WAVE: Automatic Verification of Data-Driven Web Services[*]

Alin Deutsch Victor Vianu
Department of Computer Science & Engineering
University of California, San Diego

Abstract

Data-driven Web services, viewed broadly as interactive systems available on the Web for users and programs, provide the backbone for increasingly complex Web applications. While this yields ever-increasing functionality, the added complexity renders such applications more vulnerable to bugs and failures, potentially compromising their robustness and correctness. Therefore, there is a need to develop verification techniques for such Web services. The WAVE *project at UC San Diego aims to develop new approaches for automatic verification of data-driven Web services. The work relies on a novel, highly effective marriage of model checking and database techniques. We summarize briefly the main contributions of the project, which range from theoretical foundations to the successful implementation of a prototype verifier.*

1 Verification of stand-alone data-driven Web services

We first outline our results on verification of data-driven Web services for single peers in isolation, then dicsuss extensions of the results to compositions of Web services. We focus on services interacting with external users or programs through a Web browser interface, and accessing an underlying database. Such services include e-commerce sites, scientific and other domain-specific portals, e-government, etc. These Web sites are often governed by complex, data-dependent workflows, controlled by queries. The spread of such services has been accompanied by the emergence of tools for their high-level specification. A representative, commercially successful example is WebML [1], which allows to specify a Web application using an interactive variant of the E-R model augmented with a workflow formalism. The code for the Web application is automatically generated from the WebML specification. This not only allows fast prototyping and improves programmer productivity but also provides new opportunities for automatic verification. Indeed, our WAVE prototype automatically verifies a significant class of such services. Verification leads to increased confidence in the correctness of database-driven Web applications generated from high-level specifications, by addressing the most likely source of errors (the application's specification, as opposed to the less likely errors in the automatic generator's implementation).

We focus on interactive Web sites generating Web pages dynamically by queries on an underlying database. The Web site accepts input from external users or programs, possibly subject to specified pre-conditions. It responds by taking some action, updating its internal state database, and moving to a new Web page determined

[*]Supported by the NSF under grant numbers IIS/0415257 and CAREER/0347968. Address: CSE 0404, UC San Diego, La Jolla, CA 92093-0404, USA

by yet another query. We model the queries used in the specification of the Web service as first-order queries (FO), also known as relational calculus, which can be viewed as an abstraction of the data manipulation core of SQL. A *run* is a sequence of inputs together with the Web pages, states, and actions generated by the Web service. The properties we wish to verify range from basic soundness of the specification (e.g. the next Web page to be displayed is always uniquely defined) to semantic properties (e.g. no order is shipped before a payment in the right amount is received). Such properties are expressed using an extension of *linear-time temporal logic* (LTL). Recall that LTL is propositional logic augmented with temporal operators such as **always**, **eventually, next** and **until**. The extension uses FO formulas in place of the atomic propositions of classical LTL, yielding a language called LTL-FO.

For example, the following is an LTL-FO formula stating that if a product x is paid at some point in the right amount y, then x is eventually delivered:

$$\forall x \forall y \ \ \mathbf{always}[(pay(x,y) \wedge price(x,y)) \rightarrow \mathbf{eventually}\ (deliver(x))]$$

Here *pay* is an input, *price* is a database relation, and *deliver* is an action relation.

The task of a verifier is to check that all runs of the Web service satisfy a given LTL-FO property (as usual in verification, runs are considered to be infinite). Verifiers search for counter-examples to the desired property, i.e. runs leading to a violation. A verifier is *complete* if it is guaranteed to find a counter-example whenever one exists. In the broader context of verification, a database-driven Web service is an *infinite-state* system, because the underlying database queried by the application is not fixed in advance. This poses an immediate and seemingly insurmountable challenge. Classical verification deals with finite-state systems, modeled in terms of propositions. For more expressive specifications, the traditional approach suggests the following strategy: first abstract the specification to a fully propositional one and next apply an existing model checker such as SPIN [6] to verify LTL properties of the abstracted model. This approach is unsatisfactory when the data values are first-class citizens, as in data-driven Web applications. For example, abstraction would allow checking that *some* product was delivered after *some* payment was completed. However, we could not inspect the payment and product data values to verify that the payment was for the delivered item, and in the correct amount. Conventional wisdom holds that, short of using abstraction, it is hopeless to attempt complete verification of infinite-state systems. In this respect, WAVE represents a significant departure because it is complete for a practically relevant class of infinite-state specifications. As far as we know, this is the first implementation of such a verifier.

In general, complete verification is easlily seen to be undecidable. Thus, completeness is only guaranteed under certain restrictions described shortly. To show that these restrictions cover a large class of applications, we have modeled a computer shopping Web site similar to the Dell site, an airline reservation application similar to Expedia, an online bookstore in the spirit of Barnes & Noble, and a sports Web site on the Motorcycle Grand Prix. We used these applications in our experimental evaluation of WAVE. If the specification and the property do not satisfy the restrictions needed for completeness, WAVE can still be used as an incomplete verifier, as typically done in software verification. The heuristics we developed remain just as effective in this case.

We now describe informally the restrictions on the Web service specifications and properties that guarantee completeness, called *input boundedness* [7, 5]. Recall that the queries we use in the specification of Web service as well as properties are FO queries. In a nutshell, input boundedness restricts the range of quantifications in FO formulas to values occurring in the input. This is natural, since interactive Web applications are input-driven. For example, to state that every payment received is in the right amount, one might use the input-bounded formula $\forall x \forall y[pay(x,y) \rightarrow price(x,y)]$, where $pay(x,y)$ is an input and $price$ is a database relation providing the price for each item.

Our main theoretical result shows the decidability of model checking for input-bounded specifications and properties. The complexity of checking that a Web service specification $\mathcal{W}$ satisfies an LTL-FO property φ is shown to be PSPACE. We briefly describe the technique underlying this result, as well as the implementation of WAVE. In our scenario, a first difficulty facing a verifier is that exhaustive exploration of all possible runs of a

Web service $\mathcal{W}$ on all databases is impossible since there are infinitely many possible databases and the length of runs is infinite. The solution lies in avoiding explicit exploration of the state space. Instead of materializing a full initial database and exploring the possible runs on it, we generate a compact representation of equivalence classes of actual runs, called *pseudo-runs*, by lazily making at each point in the run just the assumptions needed to obtain the next configuration and check satisfaction of φ. Specifically, for input-bounded $\mathcal{W}$ and φ, this can be done as follows:

(i) explicitly specify the tuples in the database that use only a small set of relevant constants C computed from $\mathcal{W}$ and φ; this is called the *core* of the database and remains unchanged throughout the run. Its size is polynomial in $\mathcal{W}$ and φ.

(ii) at each step in the run, make additional assumptions about the content of the database, needed to determine the next possible configurations. The assumptions involve only a small set of additional values.

The key point is that the local assumptions made in (ii) at each step need not be checked for global consistency. Indeed, a non-obvious consequence of the input-bounded restriction is that these assumptions are guaranteed to be globally consistent with *some* very large database which is however never explicitly constructed. Since pseudo-run configurations are of polynomial size, this yields a PSPACE verification algorithm and establishes our main theoretical result [5].

Theorem 1: Given an input-bounded Web service specification $\mathcal{W}$ and LTL-FO formula φ, it is PSPACE-complete whether $\mathcal{W}$ satisfies φ.

The PSPACE upper bound holds assuming a fixed bound on the arity of database and state relations. Otherwise, the complexity is EXPSPACE (with the arity in the exponent). It is worth noting that, in the broader context of static analysis, the PSPACE complexity is the best one can hope for. Indeed, recall that even satisfaction of a propositional LTL property by a finite-state Mealy machine is already PSPACE-complete.

The input-boundedness restriction imposed for decidability turns out to be quite tight. Indeed, we showed that even minor relaxations to these restrictions lead to undecidability. Some extensions to the model also lead to undecidability, such as allowing key constraints on the database. On the other hand, PSPACE decidability continues to hold with built-in predicates such as a dense order on the domain.

The WAVE verifier To explore the practical feasibility of our ideas, we embarked upon the implementation of the WAVE verifier. First, we developed a tool for high-level, efficient specification of data-driven Web services, in the spirit of WebML. Next, we implemented WAVE taking as input a specification of a Web service using our tool, and an LTL-FO property to be verified. The implementation is made possible by a novel coupling of classical model-checking with database optimization techniques. Interestingly, the starting point is the pseudo-run technique used to show the PSPACE upper bound. However, verification becomes practical only in conjunction with an array of additional heuristics and optimization techniques, yielding critical improvements. Chief among these is dataflow analysis, allowing to dramatically cut down the number of database cores and pseudo-runs generated in a search.

We evaluated the verifier on a set of practically significant Web application specifications, mimicking the core features of sites such as Dell, Expedia, and Barnes and Noble. The experimental results are quite exciting: we obtained surprisingly good verification times (on the order of seconds), suggesting that automatic verification is practically feasible for large classes of properties and Web services. We describe the implementation and our experimental results in [2]. A demo of the WAVE prototype is presented in [4] and is also available at http://db.ucsd.edu/wave.

2 Extension to Web service compositions

The above results apply to the verification of single peers in isolation. We extended these results to the more challenging but practically interesting case of *compositions* of Web services. Asynchronous communication between peers adds another dimension that has to be taken into account. We briefly describe the model and results.

In a composition of Web services, peers communicate with each other by sending and receiving messages via one-way channels implemented by *message queues*. Each queue is associated with a unique sender who places messages into the queue, and a unique receiver who consumes messages from it in FIFO order (thus, we assume messages arrive in the same order they were sent). The messages can be *flat* or *nested*. Flat messages consist of single tuples, e.g. the age and social security number of a given customer. Nested messages consist of a set of tuples, e.g. the set of books written by an author.

As in the stand-alone case, each peer can receive external inputs and produce actions (sets of tuples). In a composition, each peer additionally consumes messages from its input queues, and generates output messages. A *configuration* of the composition consists of the configurations of all participating peers (the database, their local state relations, inputs, current action relations, and the message queues). A run of the composition is a sequence of consecutive configurations. We only consider serialized runs, in which at every step precisely one peer performs a transition. Properties of runs to be verified are specified in an extension of LTL-FO, where the FO statements may additionally refer to the messages currently read and sent.

In order to obtain decidability of verification, we need to extend the input-boundedness restriction introduced for single peers. Naturally, we need to also require input-boundedness of the queries defining output messages. Additional restrictions must be placed on the message channels: they may be lossy, but are required to be bounded. With these restrictions, verification is again shown to be PSPACE-complete (for fixed-arity relations, and EXPSPACE otherwise).

The above model of compositions assumes that all specifications of participating peers are available to the verifier. However, compositions may also involve autonomous parties unwilling to disclose the internal implementation details. In this case, the only information available is typically a specification of their input-output behavior. This led us to investigate *modular* verification. This consists in verifying that a subset of fully specified peers behaves correctly, subject to input-output properties of the other peers. We obtained similar decidability results for verification, subject to an appropriate extension of the input-boundedness restriction.

The results on verification of Web service compositions are described in [3].

Conclusion The results of the WAVE project obtained so far are very encouraging. They suggest that interactive applications controlled by database queries may be unusually well suited to automatic verification, and that our approach based on a mix of model checking and database optimization techniques may come to have significant practical impact.

References

[1] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing data-intensive Web applications*. Morgan-Kaufmann, 2002

[2] A. Deutsch, M. Marcus, L. Sui, V. Vianu and D. Zhou: A verifier for interactive, data-driven Web applications. *ACM SIGMOD Conference* 2005: 539-550

[3] A. Deutsch, L. Sui, V. Vianu, D. Zhou: Verification of communicating data-driven Web services. *ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems* (PODS) 2006: 90-99

[4] A. Deutsch, L. Sui, V. Vianu, D. Zhou: A system for specification and verification of interactive, data-driven Web applications (demo paper). *ACM SIGMOD Conference* 2006: 772-774

[5] A. Deutsch, L. Sui, V. Vianu: Specification and verification of data-driven Web applications. Invited to special issue of *J. Comput. Syst. Sci.* 73(3): 442-474 (2007). Extended abstract in *ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems* (PODS) 2004: 71-82

[6] G. Holzmann. *The Spin Model Checker – Primer and Reference Manual*. Addison-Wesley, 2003

[7] M. Spielmann. Verification of relational transducers for electronic commerce. *J. Comput. Syst. Sci.* 66(1):40–65 (2003). Extended abstract in *ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems* (PODS) 2000: 92-103