Automatic Verification of Database-Driven Systems: A New Frontier

Victor Vianu*
U.C. San Diego
La Jolla, CA 92093
USA

ABSTRACT

We describe a novel approach to verification of software systems centered around an underlying database. Instead of applying general-purpose techniques with only partial guarantees of success, it identifies restricted but reasonably expressive classes of applications and properties for which sound and complete verification can be performed in a fully automatic way. This leverages the emergence of high-level specification tools for database-centered applications that not only allow fast prototyping and improved programmer productivity but, as a side effect, provide convenient targets for automatic verification. We present theoretical and practical results on verification of database-driven systems. The results are quite encouraging and suggest that, unlike arbitrary software systems, significant classes of database-driven systems may be amenable to automatic verification. This relies on a novel marriage of database and model checking techniques, of relevance to both the database and the computer aided verification communities.

1. INTRODUCTION

Software systems centered around a database are becoming pervasive in numerous applications. They are encountered in areas as diverse as electronic commerce, e-government, scientific applications, enterprise information systems, and business process support. Such systems are often very complex and prone to costly bugs, whence the need for verification of critical properties.

Classical software verification techniques that can be applied to such systems include *model checking* and *theorem proving*. However, both have serious limitations. Indeed, model checking usually requires performing finite-state abstraction on the data, resulting in serious loss of semantics for both the system and properties being verified. Theorem proving is incomplete and requires expert user feedback.

Over the last decade, much work in the verification community has focused on extending classical model checking to infinite-state systems, of which database-driven systems are a special case (e.g., see [26] for a survey). However, in most of this work the emphasis is on studying recursive control, with data either ignored or finitely abstracted. More recent work has been focusing specifically on data as a source of infinity. This includes augmenting recursive procedures with integer parameters [18], rewriting systems with data [19, 17], Petri nets with data associated to tokens [53], automata and logics over infinite alphabets [21, 20, 58, 32, 51, 15, 17], and temporal logics manipulating data [32, 33]. However, the restricted use of data and the particular properties verified have limited applicability to database-driven systems. In particular, model checking LTL properties in the presence of data quickly becomes undecidable.

Recently, an alternative approach to verification of database-driven systems has been taking shape, at the confluence of the database and computer-aided verification areas. It aims to identify restricted but sufficiently expressive classes of database-driven applications and properties for which sound and complete verification can be performed in a fully automatic way. This approach leverages another trend in database-driven applications: the emergence of high-level specification tools for database-centered systems. A representative, commercially successful example is WebML [27, 23], which allows to specify a Web application using an interactive variant of the E-R model augmented with a workflow formalism. Non-interactive variants of Web page specifications have been proposed in Strudel [42], Araneus [54] and Weave [43], which target the automatic generation of Web sites from an underlying database. Such tools automatically generate the code for the Web application from the high-level specification. This not only allows fast prototyping and improves programmer productivity but, as a side effect, provides new opportunities for automatic verification. Indeed, the high-level specification is a natural target for verification, as it addresses the most likely source of errors (the application's specification, as opposed to the less likely errors in the automatic generator's implementation).

The theoretical and practical results obtained so far concerning the verification of such systems are quite encouraging. They suggest that, unlike arbitrary software systems, significant classes of database-driven systems may be amenable to automatic verification. This relies on a novel marriage of database and model checking techniques, and is relevant to both the database and the computer aided verification communities.

In the rest of the paper, we describe several models and results on automatic verification of database-driven systems.

*Work supported in part by the National Science Foundation under award number IIS-0415257.

ICDT 2009, March 23–25, 2009, Saint Petersburg, Russia.
Copyright 2009 ACM 978-1-60558-423-2/09/0003 ...$5.00

We begin with a brief review of the classical automata-theoretic approach to model checking, including transition systems, linear temporal logic (LTL), and model checking based on Büchi automata. We then discuss extensions needed in the framework of database-driven systems, in which finite states are replaced by database instances, and propositions by properties of instances expressed in some appropriate language such as first-order logic (FO). Finally, we specialize this general framework to the more concrete scenario of interactive Web services, and describe several theoretical results, as well as the implementation of a prototype verifier.

2. CLASSICAL MODEL CHECKING

We briefly review here the automata-theoretic approach to LTL model checking (see e.g. [28, 55]).

Classical model checking applies to finite-state transition systems. While finite-state systems may fully capture the semantics of some systems to be verified (for example logical circuits), most software systems are in fact infinite-state systems, of which a finite-state transition system represents a rough abstraction. Properties of the actual system are also abstracted, using a finite set of propositions whose truth values describe each of the finite states of the transition system.

More formally, a finite-state transition system $\mathcal{T}$ is a tuple (S, s_0, T, P, σ) where S is a finite set of *configurations* (sometimes called states), $s_0 \in S$ the initial configuration, T a transition relation among the configurations such that each configuration has at least one successor, P a finite set of propositional symbols, and σ a mapping associating to each $s \in S$ a truth assignment $\sigma(s)$ for P. $\mathcal{T}$ may be specified using various formalisms such as a non-deterministic finite-state automaton, or a Kripke structure [55]. A run ρ of $\mathcal{T}$ is an infinite sequence of configurations $s_0, s_1, \ldots$ such that $(s_i, s_{i+1}) \in T$ for each $i \geq 0$. Intuitively, the information about configurations in S that is relevant to the property to be verified is provided by the corresponding truth assignments to P. The obvious extension of σ to a run ρ is denoted by $\sigma(\rho)$. Thus, $\sigma(\rho)$ is an infinite sequence of truth assignments to P corresponding to the sequence of configurations in ρ.

Properties of runs are specified by extensions of propositional logic with temporal operators. We recall here Linear-Time Logic (LTL). A minimal set of temporal operators for LTL consists of $\mathbf{X}$ (next) and $\mathbf{U}$ (until). Consider a run $\rho = s_0, s_1, \ldots$ of $\mathcal{T}$ and let $\rho_{\geq j}$ denote the run $s_j, s_{j+1}, \ldots$, for $j \geq 0$. Satisfaction of an LTL formula by a run is defined by structural recursion as follows:

- $\rho \models p$ for $p \in P$ iff p is true in $\sigma(s_0)$;
- $\rho \models \mathbf{X}\varphi$ iff $\rho_{\geq 1} \models \varphi$; and
- $\rho \models \varphi \mathbf{U} \psi$ iff there exists $j \geq 0$ such that $\rho_{\geq j} \models \psi$ and $\rho_{\geq i} \models \varphi$ for each i, $0 \leq i < j$.

The above temporal operators can simulate other commonly used operators, including $\mathbf{F}$ (eventually), $\mathbf{G}$ (always), and $\mathbf{B}$ (before). Indeed, $\mathbf{F}\varphi \equiv \mathit{true}\,\mathbf{U}\,\varphi$ and $\mathbf{G}\varphi \equiv \neg\,\mathbf{F}\neg\varphi$. For $\mathbf{B}$ (before), we take the definition $\varphi\mathbf{B}\psi \equiv \neg(\neg\varphi\mathbf{U}\psi)$ (if ψ holds at some point, then φ must hold in a previous configuration). We use the above operators as shorthand in LTL formulas whenever convenient.

Given a transition system $\mathcal{T}$ as above and an LTL formula φ using propositions in P, the associated model checking problem is to verify whether every run of $\mathcal{T}$ satisfies φ, or equivalently, that no run of $\mathcal{T}$ satisfies $\neg\varphi$. This can be done efficiently using a key result of [68], showing that from each LTL formula φ over P one can construct an automaton B_φ on infinite sequences, called a Büchi automaton, whose alphabet consists of the truth assignments to P, and which accepts precisely the runs of $\mathcal{T}$ that satisfy φ. This reduces the model checking problem to checking the existence of a run ρ of $\mathcal{T}$ such that $\sigma(\rho)$ is accepted by $B_{\neg\varphi}$.

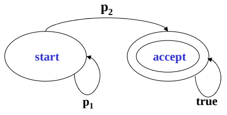


Figure 1: Büchi automaton for $\varphi = p_1\ \mathbf{U}\ p_2$

We briefly recall Büchi automata. A Büchi automaton A is defined in the same way as a nondeterministic finite state automaton, but with a special acceptance condition for infinite input sequences: a sequence is accepted iff there exists a computation of A on the sequence that reaches some accepting state f infinitely often. For the purpose of model checking, the alphabet consists of truth assignments for some given set P of propositional variables. The results of [68] show that for every LTL formula φ there exists Büchi automaton B_φ of size exponential in φ that accepts precisely the infinite sequences of truth assignments that satisfy φ. Furthermore, given a state p of B_φ and a truth assignment σ, the set of possible next states of B_φ under input σ can be computed directly from p and φ in polynomial space [63]. This allows to generate computations of B_φ without explicitly constructing B_φ.

Example 2.1. *Figure 1 shows a Büchi automaton for $p_1\mathbf{U}p_2$. Notice that the accepted infinite input sequences consist of an arbitrary-length prefix of satisfying assignments for p_1, followed by a satisfying assignment for p_2 and continued with an arbitrary infinite suffix.*

Suppose we are given a transition system $\mathcal{T}$ and an LTL formula φ over the set P of propositions of $\mathcal{T}$. The following outlines a non-deterministic PSPACE algorithm for checking whether there exists a run of $\mathcal{T}$ satisfying $\neg\varphi$: starting from the initial configuration s_0 of $\mathcal{T}$ and q_0 of $B_{\neg\varphi}$, non-deterministically extend the current run of $\mathcal{T}$ with a new configuration s, and transition to a next state of $B_{\neg\varphi}$ under input $\sigma(s)$, until an accepting state f of $B_{\neg\varphi}$ is reached. At this point, make a non-deterministic choice: (i) remember f and the current configuration s of S, or (ii) continue. If a previously remembered final state f of $B_{\neg\varphi}$ and configuration s of $\mathcal{T}$ coincide with the current state in $B_{\neg\varphi}$ and configuration in $\mathcal{T}$, then stop and answer "yes". This shows that model checking is in non-deterministic PSPACE, and therefore in PSPACE. A deterministic model checking algorithm in $O(|\mathcal{T}|2^{|\varphi|})$ is exhibited in [29].

3. FROM FINITE-STATE TO DATABASE - DRIVEN TRANSITION SYSTEMS

Adapting classical model checking to database-driven systems requires significant extensions. We informally discuss the needed extensions in general terms, then show how they specialize to some specific scenarios.

The most critical extension is that transition systems are no longer finite state. Instead, in a database-driven transition system, it is natural to model configurations as database instances. The particular data model may vary according to the application (it may be relational, XML, etc). At a minimum, the transition relation among configurations must be recursively enumerable (but can be expected to be much more restricted in realistic situations). Thus, a run of a transition system is now an infinite sequence of database instances reflecting the evolution of the system. In case of an interactive system, this includes the input received and output produced at each step.

To describe properties of such runs, a finite set of propositions is generally no longer adequate. Instead, the propositions used in LTL are replaced by formulas in some richer logic, tailored to the data model and evaluated on each configuration. For example, if the data model is relational, the logic might be FO. If it is XML, a natural logic might be based on tree patterns. Each such logic results in a different flavor of LTL with the same semantics for the temporal operators. More specifically, suppose $\mathcal{L}$ is some logic, that we leave unspecified. The LTL extension corresponding to $\mathcal{L}$, denoted LTL($\mathcal{L}$), is obtained as follows. An LTL($\mathcal{L}$) formula is obtained by taking an LTL formula φ using a set P of propositions, and replacing each proposition $p \in P$ by a formula $\pi(p)$ of $\mathcal{L}$, resulting in $\pi(\varphi) \in$ LTL($\mathcal{L}$) (see Sections 3.2 and 3.3 for examples). We refer to φ as a *propositional form* of $\pi(\varphi)$. If each $\pi(p)$ is a sentence that can be evaluated to true or false in each configuration, the semantics of $\pi(\varphi)$ is the obvious: a run of the transition system satisfies $\pi(\varphi)$ iff the sequence of truth assignments for P induced by evaluating each $\pi(p)$ on each configuration satisfies φ. As before, φ can be evaluated using the Büchi automaton B_φ, whose alphabet consists of the truth assignments of P.

One technical twist involves the use of variables in formulas of the logic. It is extremely useful to be able to refer to the same data value in different configurations. For example, when checking that every delivered product has been previously paid, one has to make sure the payment and delivery events refer to the same product, even if they occur at different times. To this end, assuming that the logic $\mathcal{L}$ uses variables to refer to data values, it is useful for the formulas used in LTL($\mathcal{L}$) to allow some of the variables to be quantified globally with respect to the entire run, rather than locally with respect to each configuration. As natural in most cases, the quantification can be assumed by default to be universal, ranging over the underlying domain.

We note that transition systems (or Kripke structures) whose configurations are relational structures, as well as first-order logic extended with modal operators, have previously been studied in the context of first-order modal logics [48, 44] (see also [41]). The emphasis in this work is mostly on sound and complete axiomatizations of first-order modal logics under various syntactic and semantic assumptions.

3.1 The Web Service Paradigm

While database-driven systems occur in many kinds of applications, they are especially prominent in the context of Web services. We will illustrate the abstract scenario of database-driven transition systems with two concrete examples of database-driven Web services: one using a relational model [36, 37], and the other XML-based [5]. Another scenario, database-driven business processes, is considered in [34]. Other work on database-driven Web services includes [8, 9], where the emphasis is on automatic synthesis of compositions rather than verification. Before describing database-driven Web services, we briefly provide some general pointers to other models for Web services.

The goal of the Web services paradigm is to enable the use of Web-hosted services with a high degree of flexibility and reliability. Web services can function in a stand-alone manner, or, more interestingly, they can be “glued” together into multi-peer *compositions* that implement complex applications. To describe and reason about Web services, various standards and models have been proposed, focusing on different levels of abstraction and targeting different aspects of the Web service (e.g., see [50] for a tutorial). At one extreme, the Web service is viewed as a black box, with its specification limited to a description of the sequences of input/output messages supported by the interface. At the other extreme, the internal logic of the Web service is available, and specified, for example, using a high-level, workflow-based formalism.

The message-based perspective of Web services is captured by the WSDL standard and has been formalized primarily using finite-state automata. Interacting Web services are modeled by distributed automata with various forms of message passing [25, 49, 10]. This uses classical techniques developed in the context of process algebras [56] and in the automata theory and verification communities [1, 59, 52, 28].

Higher-level behavioral descriptions of Web services are centered around the notions of event and activity, subject to some form of control. This has been captured in the workflows community by flowcharts, Petri nets [66, 67, 7], and state charts [47, 46, 57]. The semantic Web community has favored situation calculus [61], permitting the use of logic-based reasoning about the effects of Web services and therefore allowing the use of goal-based planning algorithms for automated constructions of compositions.

Other work approaches compositions using logic programming and description logics [13, 14, 12, 11]. Finally, another category of related models are high-level workflow models geared towards Web applications (e.g. [22, 30, 71]), and ultimately to general workflows (see [70, 45, 46, 31, 16, 69]).

3.2 Relational-Based Web Services

Proceeding to database-driven Web services, we next consider the common scenario of a service that takes input from external users and responds by producing output. The Web service can access an underlying relational database, as well as state information updated as the interaction progresses.

As a concrete verification scenario, consider Web sites specified by a high-level tool in the spirit of WebML. The contents of a Web page is determined dynamically by querying the underlying database as well as the state. The output of the Web site, transitions from one Web page to another, and state updates, are determined by the current input, state, and database, and defined by first-order queries.

Example 3.1. *We illustrate a WebML-style specification of an e-commerce Web site selling computers online. New customers can register a name and password, while returning customers can login, search for computers fulfilling certain criteria, add the results to a shopping cart, and finally* *buy the items in the shopping cart. A demo Web site implementing this example, together with its full specification, is provided at http://db.ucsd.edu/WAVE. Figure 2 depicts the Web pages of the demo in WebML style.*

A run of the above Web site starts as follows. Customers begin at the home page by providing their login name and password, and choosing one of the provided buttons (login, register, or cancel). Suppose the choice is to login. The reaction of the Web site is determined by a query checking if the name and password provided are found in the database of registered users. If the answer is positive, the login is successful and the customer proceeds to the Customer page or the Administration page depending on his status. Otherwise, there is a transition to the Error page. This continues as described by the flowchart in the figure.

A WebML-style specification $\mathcal{S}$ such as above generates an infinite-state transition system $\mathcal{T}_{\mathcal{S}}$ whose configurations are relational database instances. The schema of the configurations has several components: the database of the Web application, the state relations, the input tuples, and the output relations. Given a configuration of $\mathcal{T}_{\mathcal{S}}$, the specification $\mathcal{S}$ defines a set of possible next configurations, depending on the user's input. In particular, the transition relation of $\mathcal{T}_{\mathcal{S}}$ is decidable in PTIME, and the size of each possible next configuration is polynomial in the current one.

We now turn to the specification of temporal properties. Since the underlying model here is relational, an appropriate logic for describing configurations is FO. Thus, the logic $\mathcal{L}$ is FO over the schema of configurations. For example, suppose *price* is a binary database relation providing the price of each product, *ship* is a unary action relation providing shipped products, and *pay* is a binary input relation providing a payment for a product. To say that every product that is shipped must have been previously paid for, we use the LTL formula $\mathbf{G}\ (p\ \mathbf{B}\ q)$, where q is interpreted as the formula $\mathit{ship}(x)$ and p as $\exists z(\mathit{pay}(x,z) \wedge \mathit{price}(x,z))$ (the FO formulas replacing propositions to yield an LTL(FO) formula are called FO *components* of the formula). Note that variable x is free in the above formulas, so is quantified universally at the end. This yields the LTL(FO) formula

$$\forall x\ (\mathbf{G}\ (\exists z(\mathit{pay}(x,z) \wedge \mathit{price}(x,z))\ \mathbf{B}\ \mathit{ship}(x)))$$

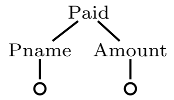
We note that variants of LTL(FO) have been introduced in [40, 3, 64]. The use of globally quantified variables is also similar in spirit to the *freeze quantifier* defined in the context of LTL extensions with data by Demri and Lazić [32, 33].

3.3 XML-Based Web Services

As a second example, we briefly mention an XML-based model of database-driven Web services, developed at INRIA [2]. Active XML (AXML) integrates the XML and Web service paradigms by allowing Web service calls to be embedded explicitly within XML documents. The Web service calls return as answers other Active XML documents, that are integrated into the caller document. The calls may be made to other services participating in a composition, or may model input from external users. Figure 3 depicts an AXML document that might arise in a mail order processing service. Its internal nodes are labeled by tags, and its leafs by tags, data values, or embedded function calls (a call to a function f is denoted by $!f$). In the example, new orders are generated by calls to the function *Mailorder*. Each mail order is a tree with root MailOrder that contains in turn calls to services *Bill*, *Deliver*, and *Reject*, triggered at appropriate times in the processing of the order. The desired sequencing is ensured by guards associated with the function calls.

A mail order evolves as follows:

1. a call to *Bill* outputs an invoice for the ordered product and returns as answer a payment, modeled as a document of the form



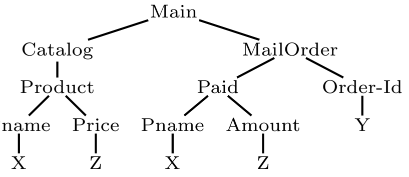
where the circles stand for data values.

2. if the payment is in the correct amount, the product is delivered (modeled as a call to *Deliver*, returning a node labeled *Delivered*); otherwise the payment is rejected.

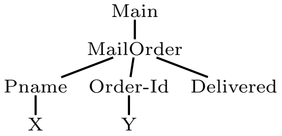
Suppose that we wish to verify the following property:

Every product for which a correct amount has been paid is eventually delivered.

To formulate the property, we use tree patterns with variables binding to data values (without going into details, let us denote such a language of tree patterns by *Tree*). The above property can be expressed in the language LTL(*Tree*) as follows. We start out with the LTL formula $\mathbf{G}(p \rightarrow \mathbf{F}q)$. The proposition p is replaced by the tree pattern



checking that the payment received for product X of order Y is in the right amount Z. The proposition q is replaced by the tree pattern



checking that product X of the same order Y is eventually delivered. Note that we wish X and Y to be the same in the tree patterns for p and q, so these are globally quantified; in contrast, Z is locally quantified. The resulting LTL(*Tree*) formula is the one in Figure 4.
The verification of LTL(*Tree*) properties of Active XML systems is studied in [5].

3.4 Model Checking Database-Driven Systems

Suppose we are given the specification $\mathcal{S}$ of a database-driven system such as above. Its semantics is a transition system $\mathcal{T}_{\mathcal{S}}$ whose configurations are database instances. Let $\mathcal{L}$ be a logic for describing these configurations, and consider a property $\varphi \in$ LTL($\mathcal{L}$). The model checking problem for $\mathcal{S}$ and φ is to verify whether $\mathcal{T}_{\mathcal{S}} \models \varphi$. Not surprisingly, this is undecidable for most familiar logics $\mathcal{L}$, such as FO. The

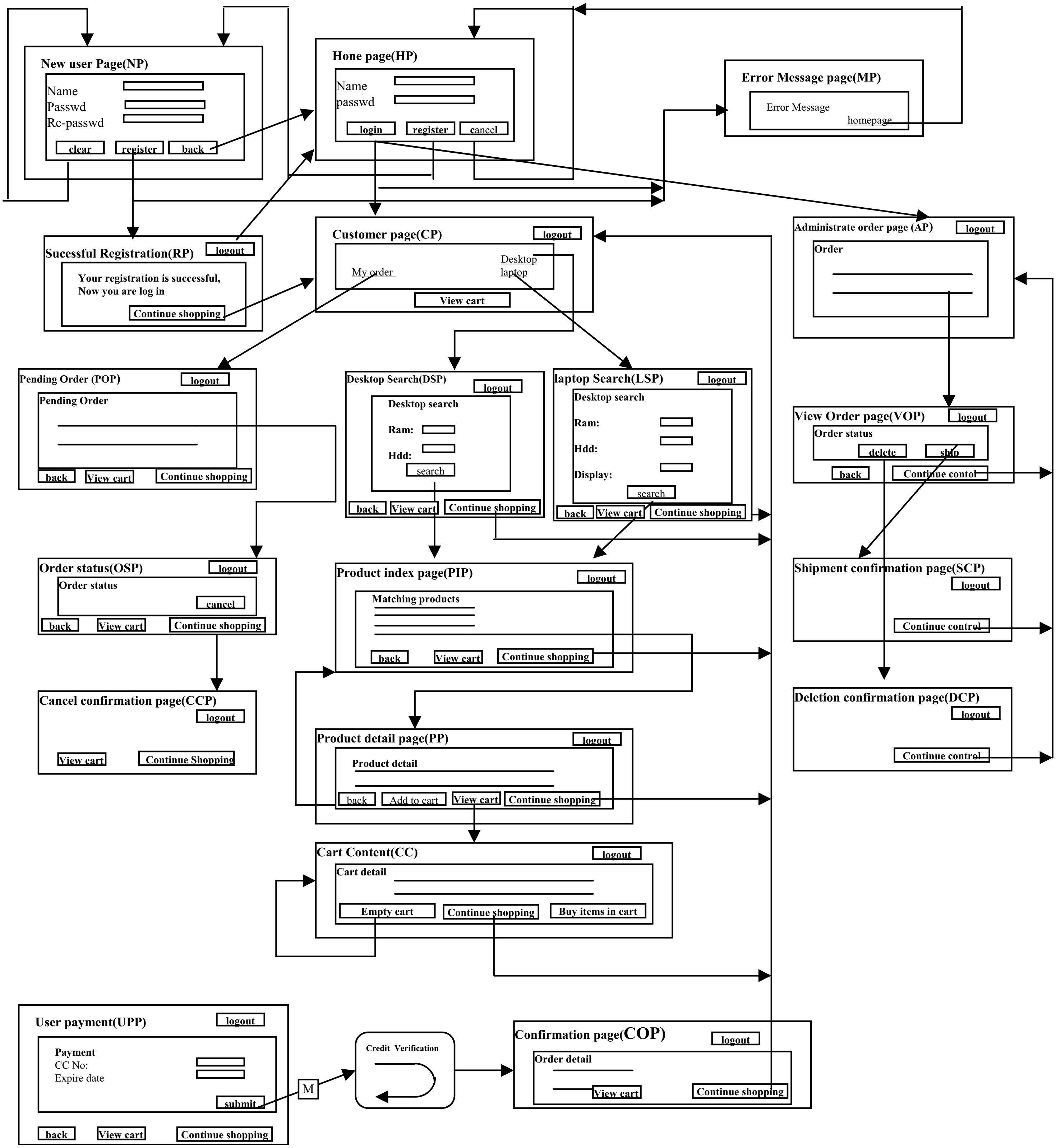


Figure 2: Web pages in the computer shopping site.

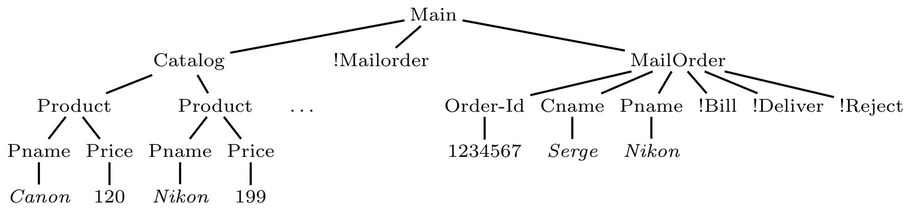


Figure 3: An AXML document

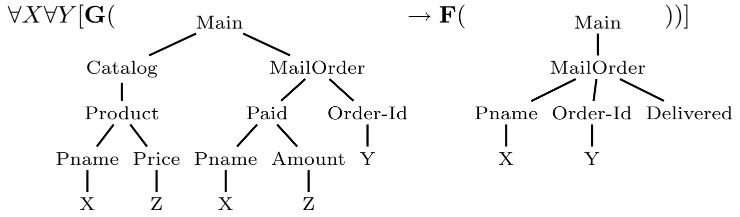


Figure 4: An LTL(*Tree*) formula

challenge then is to come up with restrictions on the systems and properties that yield decidability, while remaining sufficiently expressive to be useful in practice.

Model checking $\mathcal{S}$ with respect to φ can be viewed as a search for a counterexample run of $\mathcal{T}_{\mathcal{S}}$, i.e. a run violating φ. The immediate difficulty, compared to the classical approach, stems from the fact that $\mathcal{T}_{\mathcal{S}}$ is an infinite-state system. Consequently, the terminating procedure outlined for the finite-state case fails for two reasons. First, there are infinitely many possible initial configurations for the runs. Second, runs need not have repeating configurations, so violation of φ does not guarantee the existence of a periodic counterexample run, with a finite representation. To obtain decidability in this context, several approaches are possible. A first one is a variant of the "small model property" technique used in logic to prove decidability of satisfiability for some class of sentences. The idea is the following. Suppose one can show, for a given class of specifications and properties, that the existence of a run of a system $\mathcal{S}$ violating property φ can be checked by considering only finite prefixes of runs, whose size (length and configuration size) is smaller than a bound computable from $\mathcal{S}$ and φ. The ability to consider only finite prefixes may be a consequence of a periodicity-style property or of a restriction ensuring that the system terminates after a bounded number of transitions (extended artificially to an infinite run).

The small run property immediately yields an effective model checking procedure that consists of explicitly exploring the finite space of "small" runs. For example, this is done in [5] to show decidability of model checking for a class of AXML systems and LTL(*Tree*) properties. However, explicitly materializing runs can result in high complexity for model checking (CO-2NEXPTIME-complete in the case of AXML [5]). A more efficient alternative consists of using symbolic representations of runs. This can be effective if the size of the symbolic representation is smaller than that of the actual runs it represents. This approach is used in [36, 37, 39] to obtain a PSPACE model checking procedure for a class of relational-based systems, and is described in the next section.

4. A FORMAL MODEL OF DATABASE-DRIVEN REACTIVE SYSTEMS

We present next a formal model called *Extended Abstract State Machine Transducer*, in brief ASM$^+$, that captures in a simple way the essential features of relational database-driven reactive systems. The model is an extension of the Abstract State Machine (ASM) transducer previously studied by Spielmann [64]. Similarly to the earlier relational transducers of Abiteboul et. al. [6], ASM$^+$ transducers model database-driven reactive systems that respond to input events by producing some output, and maintain state information in designated relations. The control of the device is specified using first-order queries. The main motivation for ASM$^+$ transducers is that they are sufficiently powerful to simulate complex Web service specifications in the style of WebML. Thus, they are a convenient vehicle for developing the theoretical foundation for the verification of such systems. As we shall see, they also provide the basis for the implementation of a verifier.

4.1 ASM$^+$ Transducers

We now formalize the ASM$^+$ model. We assume a fixed and infinite set of elements $\mathbf{dom}_\infty$, equipped with a total, dense[1] order $\leq$. A *relational schema* is a finite set of relation symbols with associated arities. Relation symbols with arity zero are also called propositions. An *instance* of a relational schema consists of a mapping associating to each relation symbol of positive arity a *finite* relation of the same arity over $\mathbf{dom}_\infty$, and to each propositional symbol a truth value. The *active domain* $adom(I)$ of an instance I is the finite subset of elements of $\mathbf{dom}_\infty$ occurring in I, possibly augmented with a specified finite set of constants dependent on the context.

We assume familiarity with first-order logic (FO) over relational schemas. In addition to relations of the schema, FO formulas may use the interpreted relations $=$ and $\leq$ over $\mathbf{dom}_\infty$, as well as a finite set of *constants*, consisting of elements of $\mathbf{dom}_\infty$. As customary in relational calculus,

[1] The density assumption is used in the decidability results. It remains open whether they still hold for discrete orders.

constants are always interpreted as themselves (this differs from constants in classical logic). Unless otherwise specified, FO formulas are evaluated with respect to the infinite domain $\mathbf{dom}_\infty$. However, in order to keep results finite, we will choose to evaluate some FO formulas on an instance with respect to its active domain (this is referred to as *active domain semantics*, e.g. see [4]).

Definition 4.1. *An* ASM$^+$ *transducer $\mathcal{A}$ is a tuple*

$$\langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{O}, \mathcal{R} \rangle,$$

where:

- **D, S, I, O** *are relational schemas called database, state, input, and output schemas. We denote by* $\mathbf{Prev_I}$ *the relational vocabulary* $\{prev_I \mid I \in \mathbf{I}\}$*, where* $prev_I$ *has the same arity as I (intuitively,* $prev_I$ *refers to the input I at the previous step in the run).*
- $\mathcal{R}$ *is a set containing the following:*
  - *For each input relation* $R \in \mathbf{I}$ *of arity* $k > 0$*, a* pre-condition $\varphi_R(\bar{x})$ *where* $\varphi_R(\bar{x})$ *is an FO formula over schema* $\mathbf{D} \cup \mathbf{S} \cup \mathbf{Prev_I}$*, with k free variables* $\bar{x}$*.*
  - *For each state relation* $S \in \mathbf{S}$*, the following* state rules*:*
    - *an insertion rule* $S(\bar{x}) \leftarrow \varphi_S^+(\bar{x})$*,*
    - *a deletion rule* $\neg S(\bar{x}) \leftarrow \varphi_S^-(\bar{x})$*,*

    where the arity of S is k, $\bar{x}$ *is a k-tuple of distinct variables, and* $\varphi_S^{+/-}(\bar{x})$ *are FO formulas over schema* $\mathbf{D} \cup \mathbf{S} \cup \mathbf{Prev_I} \cup \mathbf{I}$*, with free variables* $\bar{x}$*.*
  - *For each output relation* $O \in \mathbf{O}$*, an* output rule*,* $O(\bar{x}) \leftarrow \varphi_O(\bar{x})$*, where the arity of O is k,* $\bar{x}$ *is a k-tuple of distinct variables, and* $\varphi_O(\bar{x})$ *is an FO formula over schema* $\mathbf{D} \cup \mathbf{S} \cup \mathbf{Prev_I} \cup \mathbf{I}$*, with free variables* $\bar{x}$*.*

Intuitively, the state relations designate the portion of the database that can be modified throughout the run of the transducer. Distinguishing these from the database relations that stay unchanged in a run is useful in formulating the restrictions needed for decidability. The previous inputs are introduced for the same reason. They are redundant in the general model, since they could be simulated using states. However, they become useful, once again, when formulating restrictions for decidability.

We next define the notion of "run" of an ASM$^+$ transducer. Essentially, a run specifies the fixed database and a sequence of consecutive configurations, consisting of the current states, inputs, previous inputs, and outputs . Thus, a run over database instance D is an infinite sequence

$$\{\langle S_i, I_i, P_i, O_i \rangle\}_{i \geq 0},$$

where S_i is an instance of $\mathbf{S}$, I_i is an instance of $\mathbf{I}$, P_i is an instance of $\mathbf{Prev_I}$, and O_i is an instance of $\mathbf{O}$. We call $\langle S_i, I_i, P_i, O_i \rangle$ a *configuration* of the run. We next define runs formally.

Definition 4.2. *Let* $\mathcal{A} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{O}, \mathcal{R} \rangle$ *be an* ASM$^+$ *transducer and D a database instance over schema* $\mathbf{D}$*. A* run *of* $\mathcal{A}$ *for database D is an infinite sequence of configurations* $\{\langle S_i, I_i, P_i, O_i \rangle\}_{i \geq 0}$ *where for each* $i \geq 0$*:*

- S_i, I_i, P_i, O_i *are instances of* $\mathbf{S}$*,* $\mathbf{I}$*,* $\mathbf{Prev_I}$*, and* $\mathbf{O}$*, respectively;*
- S_0, O_0, P_0 *are empty;*
- *for each relation R in* $\mathbf{I}$ *of arity* $k > 0$*,* $I_i(R)$ *consists of at most one tuple* $\bar{v}$ *that satisfies the pre-condition* φ_R *evaluated on D,* S_i*, and* P_i*, with domain* $\mathbf{dom}_\infty$ *;*
- *for each proposition R in* $\mathbf{I}$*,* $I_i(R)$ *is a truth value;*
- *for each relation R in* $\mathbf{I}$*,* $P_{i+1}(prev_R) = I_i(R)$*.*
- *for each relation S in* $\mathbf{S}$*,* $S_{i+1}(S)$ *is the result of evaluating*

  $$(\varphi_S^+(\bar{x}) \wedge \neg\varphi_S^-(\bar{x})) \vee$$
  $$(S(\bar{x}) \wedge \neg\varphi_S^-(\bar{x}) \wedge \neg\varphi_S^+(\bar{x}))$$

  on D, S_i*,* I_i*, and* P_i*, with active domain semantics.*
- *for each relation* $O \in \mathbf{O}$*,* $O_{i+1}(A)$ *is the result of evaluating* φ_O *on* D, S_i*,* I_i*, and* P_i*, again with active domain semantics.*

Note that the state and outputs specified at step $i+1$ in the run are those triggered at step i. Note also that all database, state, and output relations in a run are finite, because formulas in the corresponding rules are evaluated with active domain semantics. However, there may be infinitely many inputs satisfying the input pre-conditions, since these are evaluated with respect to $\mathbf{dom}_\infty$. Consequently, inputs may introduce infinitely many new values in the course of a run. Finally, observe that inputs are allowed to be empty. This matches practical situations in which some inputs are optional. For instance, in an e-commerce application such as that in Example 3.1, a user may be shown several drop-down menus, but a choice in just one of the menus may be sufficient to cause a transition to a new page. As it turns out, this semantics also has subtle consequences for verification.

4.2 Verification of LTL(FO) properties

In order to specify properties of ASM$^+$ transducers, we use the language LTL(FO). More specifically, for an ASM$^+$ transducer $\mathcal{A} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{O}, \mathcal{R} \rangle$, the FO components of LTL(FO) formulas are over relational vocabulary

$$\langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{Prev_I}, \mathbf{O} \rangle.$$

Satisfaction of LTL(FO) formulas by $\mathcal{A}$ is defined as described in the more general setup of Section 3.

It is easily seen that it is undecidable if an ASM$^+$ transducer satisfies an LTL(FO) formula, as a direct consequence of Trakhtenbrot's theorem (undecidability of finite satisfiability of FO sentences [65]). To obtain decidability, we must restrict both the transducers and the LTL(FO) sentences. To this end, we adapt a restriction proposed in [64] for ASM transducers, called "input boundedness". The core idea of input boundedness is that quantifications used in formulas of the specification and property are guarded by input atoms. For example, the LTL(FO) formula

$$\forall x \; (\mathbf{G} \; (\exists z (pay(x,z) \wedge price(x,z)) \; \mathbf{B} \; ship(x)))$$

is input bounded, since the quantification $\exists z$ is guarded by $pay(x,z)$ and pay is an input relation. This restriction matches naturally the intuition that the system modeled by the transducer is input driven. The actual restriction is quite technical, but provides an appealing package. First, it turns out to be tight, in the sense that even small relaxations lead to undecidability. Second, as argued in [36, 37], it remains sufficiently rich to express a significant class of practically relevant applications and properties. As a typical example, the e-commerce Web application in Example 3.1 can be modeled under this restriction, and many relevant natural properties can be expressed. Third, the complexity of verification is PSPACE (for fixed-arity schemas), which is about as low as one can hope for, given that model checking is already PSPACE-complete in the finite-state case [62]. Moreover, the proof technique developed to show decidability in PSPACE provides the basis for the implementation of an actual verifier.

Of course, the model and input-bounded restrictions also has many limitations. For example, the model includes no numerical domain with arithmetic operations. To deal with such limitations, one can use a form of abstraction, at the cost of losing completeness. For example, one can model arithmetic operations as "black box" relations, ignoring their semantics. This preserves soundness of verification (a system certified as correct is indeed so) but may yield false negatives (a generated counterexample may violate the semantics of the arithmetic operations).

We next discuss in more detail the input bounded restriction and the proof of decidability of verification. The input-bounded restriction is formulated as follows. Let

$$\mathcal{A} \;=\; \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{O}, \mathcal{R} \rangle$$

be an ASM$^+$ transducer. The set of *input-bounded* FO formulas over the schema $\mathbf{D} \cup \mathbf{S} \cup \mathbf{I} \cup \mathbf{O} \cup \mathbf{Prev_I}$ is obtained by replacing in the definition of FO the quantification formation rule by the following:

- if φ is a formula, α is a current or previous input atom using a relational symbol from $\mathbf{I} \cup \mathbf{Prev_I}$, $\bar{x} \subseteq free(\alpha)$, and $\bar{x} \cap free(\beta) = \emptyset$ for every state or output atom β in φ, then $\exists \bar{x}(\alpha \wedge \varphi)$ and $\forall \bar{x}(\alpha \rightarrow \varphi)$ are formulas.

An ASM$^+$ transducer is input-bounded if all formulas in state and output rules are input bounded, and all input preconditions are $\exists^*$FO formulas in which all state atoms are ground, i.e. use only constants and no variables (note that the input pre-conditions do not have to obey the restricted quantification formation rule above). An LTL(FO) sentence over the schema of $\mathcal{A}$ is input-bounded if all of its FO components are input-bounded.

The input-bounded restriction explains some of the choices made in the definition of ASM$^+$ transducers. Note how state and database relations are subject to different treatment under the restriction. For example, state atoms can only appear ground in input pre-conditions, whereas database atoms are unrestricted. This explains the distinction made in the definition of ASM$^+$ between fixed database relations and updatable state relations. The ground state atom requirement also explains the need for explicit access to previous inputs. While these could be inserted in states, they cannot be accessed by ground state atoms. Thus, the availability of previous inputs increases the power of input-bounded ASM$^+$ transducers. The additional expressiveness turns out to be very useful in modeling practical applications. For instance, in Example 3.1, this allows generating a list of matching products in response to previous input from the user specifying desired characteristics.

We next outline the proof that verification of input-bounded ASM$^+$ transducers can be done in PSPACE assuming a fixed bound on the arity of relations, and in EXPSPACE otherwise. Consider an ASM$^+$ transducer $\mathcal{A} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{O}, \mathcal{R} \rangle$. For simplicity of exposition, we assume that $\mathbf{I}$ consists of only one input relation (this easily extends to multiple input relations). In particular, a configuration of $\mathcal{A}$ is a tuple $\langle S, I, P, O \rangle$ where S is an instance of $\mathbf{S}$, O is an instance of $\mathbf{O}$, and I and P are instances of the unique input relation in $\mathbf{I}$, each consisting of at most one tuple.

Consider an input-bounded ASM$^+$ transducer $\mathcal{A}$ and LTL(FO) formula $\varphi_0 = \forall \bar{x} \psi_0(\bar{x})$. Let $\psi = \neg \psi_0$ and $\varphi = \neg \varphi_0 = \exists \bar{x} \psi(\bar{x})$. Let $\bar{c}$ be a tuple of constants of the same arity as $\bar{x}$. Verifying that all runs of a transducer $\mathcal{A}$ satisfy φ_0 is equivalent to checking that no run satisfies $\psi(\bar{x} \leftarrow \bar{c})$ (the formula obtained by substituting $\bar{c}$ for $\bar{x}$ in $\psi(\bar{x})$) for any choice of constants $\bar{c}$. Let us denote $\psi(\bar{x} \leftarrow \bar{c})$ by $\psi_{\bar{c}}$. Note that the FO components of $\psi_{\bar{c}}$ have no free variables (as variables previously free have been replaced by the constants in $\bar{c}$). We need to check whether there exists some run of $\mathcal{A}$ satisfying $\psi_{\bar{c}}$.

As discussed in Section 3.4, the core difficulty of verifying the above is that $\mathcal{A}$ is an infinite-state system (as it has infinitely many configurations), rather than a finite-state system as in classical model checking. Thus, exhaustive exploration of all possible runs of $\mathcal{A}$ is impossible. 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 symbolic representations of equivalence classes of actual runs, called *pseudoruns*, in which every configuration retains just the information needed to check satisfaction of $\psi_{\bar{c}}$. Moreover, this information is sufficient to obtain the same information about the *next* configuration (so it is a "closed" representation system with respect to transitions of $\mathcal{A}$). This makes crucial use of the input-bounded restriction.

We next outline in more detail the pseudorun technique. Let $\mathcal{A}, \varphi, \bar{c}$, and $\psi_{\bar{c}}$ be as above. Let D be a database instance of $\mathbf{D}$, and let $\rho = \{\langle S_i, I_i, P_i, O_i \rangle\}_{i \geq 0}$ be a run of $\mathcal{A}$ on D. Let C be the set of all constants used in $\mathcal{R}$ and $\psi_{\bar{c}}$ (this includes $\bar{c}$). We henceforth include C in the active domain of all instances considered below. We say that two instances H and H' over the same database schema are C-isomorphic iff there exists an isomorphism from H to H' that is the identity on C and preserves $\leq$. The C-isomorphism type of H consists of all instances H' that are C-isomorphic to H. The critical observation is that, due to input-boundedness, the truth value of each FO component of $\psi_{\bar{c}}$ in a configuration $\rho_i = \langle S_i, I_i, P_i, O_i \rangle$ is completely determined by the restriction[2] of S_i and O_i to C, together with the C-isomorphism type of the subinstance of $\langle I_i, P_i, D, \leq \rangle$ restricted to $adom(I_i \cup P_i)$. Since I_i and P_i contain at most one tuple each, the number of such C-isomorphism types is finite. Our pseudoruns, defined shortly, will essentially represent such C-isomorphism types. This will allow us to limit ourselves to inspecting a transition system whose configurations are the finitely many C-isomorphism types as above and essentially reduces verification back to a classical model checking problem and, with some care, yields a PSPACE ver-

[2] The restriction of a database instance K to a set T of domain elements is the instance consisting of the tuples in K using only elements in T.

ification algorithm.

We next develop a symbolic representation for local runs, leading to our notion of pseudorun. For a configuration $\rho_i = \langle S_i, I_i, P_i, O_i \rangle$, let us denote

$$\rho_i^{\downarrow} = \langle S_i|C, I_i, P_i, O_i|C, D_i, \leq_i \rangle,$$

where D_i and $\leq_i$ are the restrictions of D and $\leq$ to $adom(I_i \cup P_i)$. We refer to $\rho_i^{\downarrow}$ as the *local configuration* corresponding to ρ_i, and to $\{\rho_i^{\downarrow}\}_{i \geq 0}$ as the *local run* of $\{\rho_i\}_{i \geq 0}$. The idea is to represent local configurations using a fixed, finite set of symbols. Intuitively, symbols must be "reused" in such a representation, so the same symbol occurring in different symbolic configurations may correspond to different domain elements in a real local run.

Let $k = 2 \cdot arity(R)$, where R is the unique input relation. Let $V_k = C \cup \{v_1, \ldots, v_k\}$, where $v_1, \ldots, v_k$ are distinct new symbols. Intuitively, the $v_1, \ldots, v_k$ are used to represent input values that are not in C. Thus, $v_1, \ldots, v_k$ function as variables: they may represent different values in different configurations; in contrast, the elements in C have a fixed interpretation (as themselves). We can clearly represent the C-isomorphism type of $\rho_i^{\downarrow}$ by an instance whose domain is V_k. To do so, it is enough to fix some injective mapping f_i from $adom(\rho_i^{\downarrow})$ to V_k that fixes C, and consider the instance $\tau_i = f_i(\rho_i^{\downarrow})$ over V_k. By definition, τ_i is C-isomorphic to $\rho_i^{\downarrow}$. A pseudorun of $\mathcal{A}$ is an extension of this representation to an entire local run of $\mathcal{A}$. The extension takes into account the connection between consecutive local configurations, induced by the fact that $I_i = P_{i+1}$ for each $i \geq 0$. Specifically, f_i and f_{i+1} must agree on the elements in I_i. Thus, $\tau = \{\tau_i\}_{i \geq 0}$ is a symbolic representation of the local run $\rho^{\downarrow} = \{\rho_i^{\downarrow}\}_{i \geq 0}$, using only elements in V_k, and $\tau \models \psi_{\bar{c}}$ iff $\rho^{\downarrow} \models \psi_{\bar{c}}$. We are on the right track, but this is not quite sufficient. Indeed, the symbolic runs would not be useful if one would need local runs to generate them. Fortunately, there is a way around this. Symbolic runs using elements in V_k can be generated independently, by directly applying the transition rules of $\mathcal{A}$. These are called pseudoruns. However, not all pseudoruns correspond to local runs on a *finite* database – some require an infinite database. To filter out the latter undesired pseudoruns, an additional criterion must be used, consisting of a certain periodicity property (rather technical and omitted here). The most subtle part of the proof, complicated by the presence of the order on the domain[3] is showing that this criterion works: there exists a local run satisfying $\psi_{\bar{c}}$ iff there exists a "periodic" pseudorun satisfying $\psi_{\bar{c}}$. The verification algorithm then non-deterministically searches for a "periodic" pseudorun satisfying $\psi_{\bar{c}}$. This can be done in PSPACE because configurations of pseudoruns are of size polynomial in $\mathcal{A}$, states of the Büchi automaton corresponding to $\psi_{\bar{c}}$ are of size polynomial in $\psi_{\bar{c}}$, and non-deterministic transitions in both $\mathcal{A}$ and the Büchi automaton can be computed in PSPACE. Finally, the "periodicity" property ensures that acceptance can be checked using a finite prefix of the pseudorun (recall also the discussion in Section 3.4). This leads to the desired PSPACE verification algorithm.

THEOREM 4.3. *It is decidable, given an input-bounded ASM$^+$ transducer $\mathcal{A}$ and an input-bounded LTL(FO) formula φ, whether every run of $\mathcal{A}$ satisfies φ. Furthermore, the complexity of the decision problem is* PSPACE*-complete for fixed arity schemas, and* EXPSPACE *otherwise.*

As mentioned earlier, the input-boundedness restrictions on transducers and properties are quite tight. We mention a few small relaxations of the restrictions or of the model that lead to undecidability:

- relaxing the input-bounded quantification rule by allowing state projections;
- allowing non-ground state atoms in input pre-conditions (this holds even with just a single unary state);
- allowing previous input relations $prev_R$ to hold *all* previous inputs to R rather than just the most recent one;
- requiring non-empty inputs at each transition;
- quantifying global variables in LTL(FO) formulas existentially rather than universally;
- extending LTL(FO) by allowing path quantifiers (one alternation is sufficient).

The proofs are by reduction from the Post Correspondence Problem [60] or from implication for functional and inclusion dependencies (e.g., see [4]), both known to be undecidable.

Verification also becomes undecidable in the presence of database constraints such as functional dependencies (even for binary relations with just one key attribute). As in the case of arithmetic operations, partial verification can still be carried out in this case, but completeness is lost. Thus, a counterexample run produced by the algorithm could be a false negative, because the database it uses violates the functional dependencies.

4.3 The WAVE Verifier

While the PSPACE upper bound obtained for verification in the input-bounded case is encouraging from a theoretical viewpoint, it does not provide any indication of its practical feasibility. Fortunately, it turns out that the pseudorun technique described above also provides a good basis for the efficient implementation of a verifier. Indeed, this technique lies at the core of the WAVE verifier, targeted at data-driven Web services of the WebML flavor [39, 35].

The verifier, as well as its target specification framework, are both implemented from scratch. Thus, we first 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 starting point for the implementation is the pseudorun technique. Indeed, the verifier basically carries out a search for counterexample pseudoruns. 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 prune the search for counterexample pseudoruns.

The verifier was evaluated on a set of practically significant Web application specifications, mimicking the core features of sites such as Dell, Expedia, and Barnes and Noble.

[3] A first proof without order is provided in [37], while the extension to an ordered domain is shown in [34], in the framework of data-driven business processes.

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 significant classes of properties and Web services. The implementation and experimental results are described in [35]. A demo of the WAVE prototype is presented in [38] and is also available at http://db.ucsd.edu/wave.

4.4 Compositions of ASM$^+$ Transducers

The verification results discussed so far apply to single ASM$^+$ transducers in isolation. These results were extended in [39] to the more challenging but practically interesting case of *compositions* of ASM$^+$ transducers, modeling compositions of database-driven Web services. Asynchronous communication between transducers adds another dimension that has to be taken into account. We briefly describe the model and results of [39].

In an ASM$^+$ composition, the transducers communicate with each other by sending and receiving messages via one-way channels modeled as *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, it is assumed 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 transducer can receive external inputs and produce outputs (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 output 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 transducer performs a transition. Properties of runs to be verified are specified in an extension of LTL(FO) where the FO components may additionally refer to the messages currently read and received.

In order to obtain decidability of verification, we need to extend the input-boundedness restriction. 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 proof is by reduction to the single transducer case. In particular, flat messages are simulated in the single transducer by new input relations, and nested messages by additional state relations. The nondeterministic choice of which ASM$^+$ moves in each transition of the composition is also simulated with an additional input.

As in the case of single transducers, verification becomes undecidable if some of the restrictions are relaxed. Not suprisingly, verification is undecidable with unbounded queues (this already happens for finite-state systems [24]). More interestingly, lossiness of channels is essential: verification becomes undecidable under the assumption that channels are *perfect*, i.e. messages are never lost (the proof is by reduction of the Post Correspondence Problem). Another subtle distinction involves how messages are observed. With lossy channels, there are two possibilities. Under the *observer-at-sender* semantics, messages are observed when sent. Under the *observer-at-recipient* semantics, they are observed when received. The semantics used in our model is the latter. It turns out that verification becomes undecidable if observer-at-sender semantics is used instead.

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 leads to an investigation of *modular* verification. It consists in verifying that a subset of fully specified transducers behaves correctly, subject to input-output properties of the other transducers. Similar decidability results are obtained in [39] for verification, subject to an appropriate extension of the input-boundedness restriction.

5. CONCLUSIONS

Database-driven systems are increasingly prevalent, particularly in the context of Web services. They provide the backbone of complex applications for which verification is critically important. A fortunate development facilitating this task is the emergence of high-level specification tools centered around database queries, that provide a natural target for verification. The results we described suggest that verification may indeed be feasible for significant classes of database-driven systems so specified. The theoretical results, as well as the implementation of an actual verifier exhibiting suprisingly good performance, are made possible by a novel coupling of techniques from database theory and model checking. The encouraging results suggest that this approach is quite promising, and may be just the starting point of a fruitful marriage between the database and computer-aided verification areas.

Verification is just one of the static analysis problems that arise in the context of database-driven systems. In the specific context of Web services, *synthesis* of compositions, as well as *orchestration* of services, are important issues that are very challenging even for finite-state systems and remain largely unexplored in the presence of data. Another interesting class of problems concerns *abstraction*, used either as a tool in verification or as a means to provide different high-level views of a system, aimed at different classes of users. These are important directions for future research.

Acknowledgement I wish to thank Serge Abiteboul, Alin Deutsch, Luc Segoufin and Moshe Vardi for providing very useful comments and suggestions on a draft of this paper.

6. REFERENCES

[1] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In *Proc. ICALP Conference*, 1989.

[2] S. Abiteboul, O. Benjelloun, and T. Milo. The Active XML project: an overview. *VLDB Journal*, 17(5):1019–1040, 2008.

[3] S. Abiteboul, L. Herr, and J. V. den Bussche. Temporal versus first-order logic to query temporal

databases. In *Proc. ACM Symp. on Principles of Database Systems (PODS)*, pages 49–57, 1996.

[4] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[5] S. Abiteboul, L. Segoufin, and V. Vianu. Static analysis of Active XML systems. In *Proc. ACM Symp. on Principles of Database Systems (PODS)*, pages 221–230, 2008.

[6] S. Abiteboul, V. Vianu, B. Fordham, and Y. Yesha. Relational transducers for electronic commerce. *Journal of Computer and System Sciences (JCSS)*, 61(2):236–269, 2000. Extended abstract in PODS 98.

[7] N. Adam, V. Atluri, and W. Huang. Modeling and analysis of workflows using Petri nets. *Journal of Intelligent Information Systems*, 10(2):131–158, 1998.

[8] D. Berardi, D. Calvanese, G. Giacomo, R. Hull, and M. Mecella. Automatic composition of transisiton-based semantic web services with messaging. In *Proc. Int'l. Conf. on Very Large Databases (VLDB)*, 2005.

[9] D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, M. Lenzerini, and M. Mecella. Modeling data & processes for service specifications in Colombo. In *EMOI-INTEROP*, 2005.

[10] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Proc. Int'l. Conf on Service-Oriented Computing (ICSOC)*, pages 43–58, 2003.

[11] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella. E-service composition by description logics based reasoning. In *Description Logics*, 2003.

[12] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella. A tool for automatic composition of services based on logics of programs. In *Technologies for E-Services, 5th International Workshop (TES)*, pages 80–94, 2004.

[13] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella. Automatic service composition based on behavioral descriptions. *Int. J. Cooperative Inf. Syst.*, 14(4):333–376, 2005.

[14] D. Berardi, G. D. Giacomo, M. Lenzerini, M. Mecella, and D. Calvanese. Synthesis of underspecified composite *e*-services based on automated reasoning. In *Proc. Int'l. Conf on Service-Oriented Computing (ICSOC)*, pages 105–114, 2004.

[15] M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *Proc. IEEE Conf. on Logic in Computer Science (LICS)*, pages 7–16, 2006.

[16] A. J. Bonner and M. Kifer. An overview of transaction logic. *Theor. Comput. Sci.*, 133(2):205–265, 1994.

[17] A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *Fundamentals of Computation Theory, 16th International Symposium (FCT)*, volume 4639 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2007.

[18] A. Bouajjani, P. Habermehl, and R. Mayr. Automatic verification of recursive procedures with one integer parameter. *Theoretical Computer Science*, 295:85–106, 2003.

[19] A. Bouajjani, Y. Jurski, and M. Sighireanu. A generic framework for reasoning about dynamic networks of infinite-state processes. In *TACAS'07*, volume 4424 of *Lecture Notes in Computer Science*, pages 690–705. Springer, 2007.

[20] P. Bouyer. A logical characterization of data languages. *Information Processing Letters*, 84(2):75–85, 2002.

[21] P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Information and Computation*, 182(2):137–162, 2003.

[22] BPML.org. Business process modeling language. http://www.bpmi.org.

[23] M. Brambilla, S. Ceri, S. Comai, P. Fraternali, and I. Manolescu. Specification and design of workflow-driven hypertexts. *Journal of Web Engineering*, 1(1), 2002.

[24] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. of the ACM*, 30(2):323–342, 1983.

[25] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: a new approach to design and analysis of e-service composition. In *World Wide Web Conference*, pages 403–410, 2003.

[26] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification of infinite structures. In *Handbook of Process Algebra*, pages 545–623. Elsevier Science, 2001.

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

[28] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.

[29] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal methods in system design*, 1:275–288, 1992.

[30] DAML-S Coalition (A. Ankolekar et al).DAML-S: Web service description for the semantic Web. In *The Semantic Web - ISWC*, pages 348–363, 2002.

[31] H. Davulcu, M. Kifer, C. R. Ramakrishnan, and I. V. Ramakrishnan. Logic based modeling and analysis of workflows. In *Proc. ACM Symp. on Principles of Database Systems (PODS)*, pages 25–33, 1998.

[32] S. Demri and R. Lazić. LTL with the Freeze Quantifier and Register Automata. In *Proc. IEEE Conf. on Logic in Computer Science (LICS)*, pages 17–26, 2006.

[33] S. Demri, R. Lazić, and A. Sangnier. Model checking freeze LTL over one-counter automata. In *Proc. 11th Int'l. Conf. on Foundations of Software Science and Computation Structures (FoSSaCS'08)*, pages 490–504, 2008.

[34] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. *This proceedings.*

[35] A. Deutsch, M. Marcus, L. Sui, V. Vianu, and D. Zhou. A verifier for interactive, data-driven web applications. In *Proc. ACM SIGMOD Int'l. Conf. on Management of Data (SIGMOD)*, pages 539–550, 2005.

[36] A. Deutsch, L. Sui, and V. Vianu. Specification and

verification of data-driven web services. In *Proc. ACM Symp. on Principles of Database Systems (PODS)*, pages 71–82, 2004.

[37] A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web services. *Journal of Computer and System Sciences (JCSS)*, 73(3):442–474, 2007.

[38] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. A system for specification and verification of interactive, data-driven Web applications. In *Proc. ACM SIGMOD Int'l. Conf. on Management of Data (SIGMOD)*, pages 772–774, 2006.

[39] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven Web services. In *Proc. ACM Symp. on Principles of Database Systems (PODS)*, pages 90–99, 2006.

[40] E. A. Emerson. Temporal and modal logic. In J. V. Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pages 995–1072. North-Holland Pub. Co./MIT Press, 1990.

[41] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about knowledge*. MIT Press, 1996.

[42] M. F. Fernández, D. Florescu, A. Y. Levy, and D. Suciu. Declarative specification of web sites with Strudel. *VLDB Journal*, 9(1):38–55, 2000.

[43] D. Florescu, K. Yagoub, P. Valduriez, and V. Issarny. WEAVE: A data-intensive web site management system(software demonstration). In *Proc. of the Conf. on Extending Database Technology (EDBT)*, 2000.

[44] J. Garson. Quantification in modal logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, pages 249–307. Reidel, 1977.

[45] D. Georgakopoulos, M. F. Hornick, and A. P. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[46] D. Harel. On the formal semantics of statecharts. In *Proc. IEEE Conf. on Logic in Computer Science (LICS)*, pages 54–64, 1987.

[47] D. Harel. Statecharts: A visual formulation for complex systems. *Sci. Comput. Program*, 8(3):231–274, 1987.

[48] G. Hughes and M. Cresswell. *An introduction to modal logic*. Methuen, 1968.

[49] R. Hull, M. Benedikt, V. Christophides, and J. Su. E-services: a look behind the curtain. In *Proc. ACM Symp. on Principles of Database Systems (PODS)*, pages 1–14, 2003.

[50] R. Hull and J. Su. Tools for design of composite web services. In *Proc. ACM SIGMOD Int'l. Conf. on Management of Data (SIGMOD)*, pages 958–961, 2004.

[51] M. Jurdzinski and R. Lazić. Alternation-free modal mu-calculus for data trees. In *Proc. IEEE Conf. on Logic in Computer Science (LICS)*, pages 131–140, 2007.

[52] O. Kupferman and M. Vardi. Synthesizing distributed systems. In *Proc. IEEE Conf. on Logic in Computer Science (LICS)*, 2001.

[53] R. Lazić, T. Newcomb, J. Ouaknine, A. Roscoe, and J. Worrell. Nets with tokens which carry data. In *ICATPN'07*, volume 4546 of *Lecture Notes in Computer Science*, pages 301–320. Springer, 2007.

[54] G. Mecca, P. Merialdo, and P. Atzeni. Araneus in the era of XML. *IEEE Data Engineering Bulletin*, 22(3):19–26, 1999.

[55] S. Merz. Model checking: a tutorial overview. In *Modeling and verification of parallel processes*. Springer-Verlag New York, 2001.

[56] R. Milner. *Communicating and mobile systems: the π calculus*. Cambridge University Press, 1999.

[57] W. Mok and D. Paper. Using harel's statecharts to model business workflows. *J. of Database Management*, 13(3):17–34, 2002.

[58] F. Neven, T. Schwentick, and V. Vianu. Finite State Machines for Strings Over Infinite Alphabets. *ACM Transactions on Computational Logic*, 5(3):403–435, 2004.

[59] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. IEEE Symp. on Foundations of Computer Science (FOCS)*, 1990.

[60] E. L. Post. Recursive unsolvability of a problem of Thue. *J. of Symbolic Logic*, 12:1–11, 1947.

[61] R. Reiter. *Knowledge in action:logical foundations for specifying and implementing dynamical systems*. MIT Press, 2001.

[62] A. Sistla and E. Clarke. The complexity of propositional linear temporal logic. *J. of the ACM*, 32:733–749, 1985.

[63] A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[64] M. Spielmann. Verification of relational transducers for electronic commerce. *Journal of Computer and System Sciences (JCSS)*, 66(1):40–65, 2003. Extended abstract in PODS 2000.

[65] B. Trankhtenbrot. The impossibility of an algorithm for the decision problem for finite models. *Doklady Academii Nauk SSSR*, 70:569–572, 1950.

[66] W. M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[67] W. M. P. van der Aalst and A. H. M. ter Hofstede. Workflow patterns: On the expressive power of (petri-net-based) workflow languages. In *Proc. of the Fourth International Workshop on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, August 28-30, 2002 / Kurt Jensen (Ed.)*, pages 1–20. Technical Report DAIMI PB-560, Aug. 2002. InternalNote: Submitted by: hr available online: http://www.daimi.aau.dk/CPnets/workshop02/cpn/papers/.

[68] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. IEEE Conf. on Logic in Computer Science (LICS)*, pages 332–344, 1986.

[69] D. Wodtke and G. Weikum. A formal foundation for distributed workflow execution based on state charts. In *Proc. of Intl. Conf. on Database Theory (ICDT)*, pages 231–246, 1997.

[70] Workflow management coalition, 2001.
http://www.wfmc.org.

[71] Web Services Flow Language(WSFL 1.0), 2001.
http://www-3.ibm.com/ software/ solutions
/webservices/pdf/WSFL.pdf.