Available online at www.sciencedirect.com
ScienceDirect
Journal of Computer and System Sciences 73 (2007) 442–474
JOURNAL OF COMPUTER AND SYSTEM SCIENCES
www.elsevier.com/locate/jcss

Specification and verification of data-driven Web applications ☆,☆☆

Alin Deutsch *, Liying Sui, Victor Vianu

Computer Science and Engineering, UC San Diego, La Jolla, CA 92093-0114, USA

Received 15 February 2005; received in revised form 27 March 2006

Available online 28 November 2006

Abstract

We study data-driven Web applications provided by Web sites interacting with users or applications. The Web site can access an underlying database, as well as state information updated as the interaction progresses, and receives user input. The structure and contents of Web pages, as well as the actions to be taken, are determined dynamically by querying the underlying database as well as the state and inputs. The properties to be verified concern the sequences of events (inputs, states, and actions) resulting from the interaction, and are expressed in linear or branching-time temporal logics. The results establish under what conditions automatic verification of such properties is possible and provide the complexity of verification. This brings into play a mix of techniques from logic and model checking.

Keywords: Automatic verification; Data-driven Web services and applications; Relational transducers; Infinite-state systems

1. Introduction

Interactive Web applications provide access to information as well as transactions and are typically powered by databases. They have a strong dynamic component and are governed by protocols of interaction with users or programs, ranging from the low-level input–output signatures used in WSDL [41], to high-level workflow specifications (e.g., see [7,9,12,27,40,42]). One central issue is to develop static analysis techniques to increase confidence in the robustness and correctness of complex Web applications. This paper presents new verification techniques for Web applications, and investigates the trade-off between the expressiveness of the Web application specification language and the feasibility of verification tasks.

In the scenario we consider, a Web application is provided by an interactive Web site that posts data, takes input from the user, and responds to the input by posting more data and/or taking some actions. The Web site can access an underlying database, as well as state information updated as the interaction progresses. The structure of the Web page the user sees at any given point is described by a Web page schema. The contents of a Web page is determined dynamically by querying the underlying database as well as the state. The actions taken by the Web site, and transitions from one Web page to another, are determined by the input, state, and database.

☆ A preliminary version of this paper appeared in *Proc. ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems*, 2004.

☆☆ Supported in part by the National Science Foundation under grants ITR-0225676 (SEEK), IKM-0415257 and IIS-0347968 (CAREER).

* Corresponding author.
E-mail addresses: deutsch@cs.ucsd.edu (A. Deutsch), lsui@cs.ucsd.edu (L. Sui), vianu@cs.ucsd.edu (V. Vianu).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2006.10.006

The properties we wish to verify about Web applications involve the sequences of inputs, actions, and states that may result from interactions with a user. This covers a wide variety of useful properties. As an example, for a Web site supporting an e-commerce application, it may be desirable to verify that no product is delivered before payment of the right amount is received. Or, we may wish to verify that the specification of Web page transitions is unambiguous, (the next Web page is uniquely defined at each point in a run), that each Web page is reachable from the home page, etc. To express such properties, we rely on temporal logic. Specifically, we consider two kinds of properties. The first requires that all runs must satisfy some condition on the sequence of inputs, actions, and states. To describe such properties we use a variant of linear-time temporal logic. Other properties involve several runs simultaneously. For instance, we may wish to check that for every run leading to some Web page, there exists a way to return to the home page. To capture such properties, we use variants of the branching-time logics CTL and CTL*.

Our results identify classes of Web applications for which temporal properties in the above temporal logics can be checked, and establish their complexity. For linear-time properties, the restriction needed for decidability essentially imposes a form of guarded quantification in formulas used in the specification of the Web application and the property. This is similar to the "input boundedness" restriction first introduced by Spielmann in the context of ASM transducers [36,37]. With this restriction, verification of linear-time properties is PSPACE-complete for schemas with fixed bound on the arity, and EXPSPACE otherwise. As justification for the input-boundedness restriction, we show that even slight relaxations of our restrictions lead to undecidability of verification. Thus, our decidability results are quite tight. In terms of expressiveness, it turns out that many practically relevant Web applications can be modeled with input-bounded specifications. For example, we have shown that significant portions of a Dell computer shopping Web site, Expedia, Barnes and Noble, and a Grand Prix motor racing web site can be specified within the restricted framework (see http://www.cs.ucsd.edu/~lsui/project/index.html for a demo). Furthermore, a verifier for input-bounded Web applications and linear-time properties has been implemented, with surprisingly good verification times for the four Web sites mentioned earlier. The implementation builds upon techniques developed in the present paper and is presented in [15,17].

For branching-time properties, the restrictions needed for decidability are considerably more stringent, and the complexity of verification ranges from PSPACE to 2-EXPTIME, depending on the restriction.

Our formalization of Web applications aims to be quite faithful to real high level specification tools in the style of WebML [9], and is rather complex. To simplify the technical development of our results, we use a convenient, much simpler abstract model of data-driven reactive systems. This consists of a device extending the Abstract State Machine (ASM) transducers previously studied by Spielmann [36,37]. Specifically, the device, that we call ASM^+ transducer, receives input consisting of relational tuples, and produces relations as output. As in Spielmann's ASM transducers, its control is specified using first-order queries accessing the input, a relational database, and state information consisting of additional relationals updated at each transition. Our main motivation for introducing ASM^+ transducers is that they are sufficiently powerful to simulate the considerably more complex Web application specifications we aim to verify, and thus are a convenient vehicle for developing the bulk of our verification results. Thus, we first establish verification results using ASM^+ transducers, then show how verification of Web applications can be reduced to verification of ASM^+ transducers.

1.1. Related work

Our notion of Web application is a fairly broad one. It encompasses a large class of data-intensive Web applications equipped (implicitly or explicitly) with workflows that regulate the interaction between different partners who can be users, WSDL-style Web services, Web sites, programs and databases. We address the verification of properties pertaining to the runs of these workflows.

Prior work on Web service verification has mostly focused on propositional (finite-state) abstractions of both the service workflow and the properties. These abstractions disregard the underlying database and the data values involved in the interaction. They allow one to verify for instance that *some* payment occurred before *some* shipment, but not that it involved the intended product and the right amount. [34] proposes an approach to the verification and automated composition of finite-state Web services specified using the DAML-S standard [12]. The verified properties are propositional, abstracting from the data values. They pertain to safety, liveness and deadlocks, all of which are expressible in LTL. [33] is concerned with verifying a given finite-state Web service flow specified in the standard WSFL [42] by using the explicit state model checker SPIN [26]. The properties are expressed in LTL (again abstracting

from data content). Another data-agnostic verification effort is carried out in [24,29], which describe verification techniques focusing on bugs in the control flow engendered by browser operations. The control flow is specified using a browser-action-aware calculus. The flow is verified using model checking, after abstraction to finite-state automata labeled with propositional predicates. The same automata are used for property specification.

Our model is related to WebML [9], a high-level conceptual model for data-driven Web applications, extended in [8] with workflow concepts to support process modeling. It is also related to [4], which proposes a model of peers with underlying databases. The model is a particular case of the one presented here, in which database and state access is restricted to key lookup only, so that at most one tuple is retrieved or updated at any given time. [4] does not address verification, focusing on automatic synthesis of a desired Web service by "gluing" together an existing set of services.

Other related models are high-level workflow models geared towards Web applications (e.g. [7,12,42]), and ultimately to general workflows (see [5,14,22,25,39,40]), whose focus is however quite different from ours. Non-interactive variants of Web page schemas have been proposed in prior projects such as Strudel [19], Araneus [32] and Weave [20], which target the automatic generation of Web sites from an underlying database.

More broadly, our research is related to the general area of automatic verification, and particularly reactive systems [30,31]. Directly relevant to us is Spielmann's work on Abstract State Machine (ASM) transducers [36,37]. Similarly to the earlier relational transducers of Abiteboul et al. [3], these model database-driven reactive systems that respond to input events by taking some action, and maintain state information in designated relations. Our ASM^+ transducer model extends ASM transducers with two features: (i) the ability to constrain inputs by a FO formula, and (ii) allowing access to previous user inputs. The additional features turn out to be essential for simulating the Web applications we consider. In proving our results on verification of linear-time properties of ASM^+ transducers, we build upon the techniques developed in [36,37]. However, unlike the proof of Spielmann that consists of reducing the verification problem to finite satisfiability of E + TC formulas,[1] we provide a more direct proof for ASM^+ transducers. In particular, this also yields an alternative proof of Spielmann's original result on ASM. As discussed in the paper, the more direct proof also provides the basis for the implementation of a practical verifier for linear-time properties of Web applications, described in [15,17]. For the results on branching-time properties we use a mix of techniques from finite-model theory and temporal logic (see [18]), as well as automata-theoretic model-checking techniques developed by Kupferman, Vardi, and Wolper [28].

The paper is organized as follows. Section 2 introduces our model and specification language for Web sites. Section 3 presents the variants of linear and branching time temporal logics used to specify properties of Web applications. Next, Section 4 introduces ASM^+ transducers and presents the verification results in this context. Finally, Section 5 establishes the verification results for Web applications, mostly by reduction to verification of ASM^+ transducers.

2. Web application specifications

In this section we provide our model and specification language for data-driven Web applications. In doing so, we aim to capture a significant portion of real specification languages provided by high-level tools for Web application specification, such as WebML [9]. Our model of a Web application captures the interaction of an external user[2] with the Web site, referred to as a "run." Informally, a Web application specification has the following components:

- a database that remains fixed throughout each run;
- a set of state relations that change throughout the run in response to user inputs;
- a set of Web page schemas, of which one is designated as the "home page," and another as an "error page";
- each Web page schema defines how the set of current input choices is generated as a query on the database and states. In addition, it specifies the state transition in response to the user's input, the actions to be taken, as well as the next Web page schema.

Note that the state relations can be thought of as the portion of the database that can change during a run. The distinction between states and immutable relations becomes relevant when we consider restrictions needed for decidability of verification.

[1] E + TC is existential first-order logic augmented with a transitive closure operator.

[2] We use the term "user" generically to stand for any partner interacting with the Web site, be it an actual user, program, a Web service, etc.

Intuitively, a run proceeds as follows. First, the user accesses the home page, and the state relations are initialized to empty. When accessed, each Web page generates a choice of inputs for the user, by a query on the database and states. All input options are generated by the system except for a fixed set that represents specific user information (e.g. name, password, credit card number, etc.). These are represented as constant symbols in the input schema, whose interpretations are provided by the user throughout the run as requested. The user chooses at most one tuple among the options provided for each input. In response to this choice, a state transition occurs, actions are taken, and the next Web page schema is determined, all according to the rules of the specification. As customary in verification of reactive systems, we assume that all runs are infinite (finite runs can be easily represented as infinite runs by fake loops).

We now formalize the above notion of Web application. We assume a fixed and infinite set of elements $\mathbf{dom}_\infty$. A *relational schema* is a finite set of relation symbols with associated arities, together with a finite set of constant symbols. Relation symbols with arity zero are also called propositions. A relational instance over a relational schema consists of a finite subset *Dom* of $\mathbf{dom}_\infty$, and a mapping associating to each relation symbol of positive arity a finite relation of the same arity over *Dom*, to each propositional symbol a truth value, and to each constant symbol an element of *Dom*. We use several kinds of relational schemas, with different roles in the specification of the Web application.

We assume familiarity with first-order logic (FO) over relational vocabularies. We adopt here an active domain semantics for FO formulas, as commonly done in database theory (e.g., see [2]).

Definition 2.1. A *Web application* $\mathcal{W}$ is a tuple $\langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathbf{W}, W_0, W_\epsilon \rangle$, where:

- $\mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}$ are relational schemas called database, state, input, and action schemas, respectively. The sets of relation symbols of the schemas are disjoint (but they may share constant symbols). We refer to constant symbols in $\mathbf{I}$ as *input constants*, and denote them by $\mathbf{const}(\mathbf{I})$.
- $\mathbf{W}$ is a finite set of Web page schemas.
- $W_0 \in \mathbf{W}$ is the *home page* schema, and $W_\epsilon \notin \mathbf{W}$ is the *error page* schema.

We also denote by $\mathbf{Prev_I}$ the relational vocabulary $\{prev_I \mid I \in \mathbf{I} - \mathbf{const}(\mathbf{I})\}$, where $prev_I$ has the same arity as I (intuitively, $prev_I$ refers to the input I at the previous step in the run).

A Web page schema $W \in \mathbf{W}$ is a tuple $\langle \mathbf{I}_W, \mathbf{A}_W, \mathbf{T}_W, \mathcal{R}_W \rangle$ where $\mathbf{I}_W \subseteq \mathbf{I}$, $\mathbf{A}_W \subseteq \mathbf{A}$, $\mathbf{T}_W \subseteq \mathbf{W}$ ($\mathbf{T}_W$ is a set of target Web pages). Then $\mathcal{R}_W$ is a set of rules containing the following:

- For each input relation $I \in \mathbf{I}_W$ of arity $k > 0$, an *input rule* $Options_I(\bar{x}) \leftarrow \varphi_{I,W}(\bar{x})$ where $Options_I$ is a relation of arity k, $\bar{x}$ is a k-tuple of distinct variables, and $\varphi_{I,W}(\bar{x})$ is an FO formula over schema $\mathbf{D} \cup \mathbf{S} \cup \mathbf{Prev_I} \cup \mathbf{const}(\mathbf{I})$, with free variables $\bar{x}$.
- For each state relation $S \in \mathbf{S}$, one, both, or none of the following *state rules*:
  – an insertion rule $S(\bar{x}) \leftarrow \varphi^+_{S,W}(\bar{x})$,
  – a deletion rule $\neg S(\bar{x}) \leftarrow \varphi^-_{S,W}(\bar{x})$,
  where the arity of S is k, $\bar{x}$ is a k-tuple of distinct variables, and $\varphi^\epsilon_{S,W}(\bar{x})$, $\epsilon \in \{+, -\}$, are FO formulas over schema $\mathbf{D} \cup \mathbf{S} \cup \mathbf{Prev_I} \cup \mathbf{const}(\mathbf{I}) \cup \mathbf{I}_W$, with free variables $\bar{x}$.
- For each action relation $A \in \mathbf{A}_W$, an *action rule* $A(\bar{x}) \leftarrow \varphi_{A,W}(\bar{x})$ where the arity of A is k, $\bar{x}$ is a k-tuple of distinct variables, and $\varphi_{A,W}(\bar{x})$ is an FO formula over schema $\mathbf{D} \cup \mathbf{S} \cup \mathbf{Prev_I} \cup \mathbf{const}(\mathbf{I}) \cup \mathbf{I}_W$, with free variables $\bar{x}$.
- For each $V \in \mathbf{T}_W$, a *target rule* $V \leftarrow \varphi_{V,W}$ where $\varphi_{V,W}$ is an FO sentence over schema $\mathbf{D} \cup \mathbf{S} \cup \mathbf{Prev_I} \cup \mathbf{const}(\mathbf{I}) \cup \mathbf{I}_W$.

Finally, $W_\epsilon = \langle \emptyset, \emptyset, \{W_\epsilon\}, \mathcal{R}_{W_\epsilon} \rangle$ where $\mathcal{R}_{W_\epsilon}$ consists of the rule $W_\epsilon \leftarrow \mathit{true}$.

Intuitively, the action rules of a Web page specify the actions to be taken in response to the input. The state rules specify the tuples to be inserted or deleted from state relations (with conflicts given no-op semantics, see below). If no rule is specified in a Web page schema for a given state relation, the state remains unchanged. The input rules specify a set of options to be presented to users, from which they can pick at most one tuple to input (this feature corresponds to menus in user interfaces). At every point in time, I contains the current input tuple, and $prev_J$ contains the input to J in the previous step of the run (if any). The choice of this semantics for $prev_J$ relations is somewhat arbitrary, and other choices are possible without affecting the results. For example, another possibility is to have $prev_J$ hold the

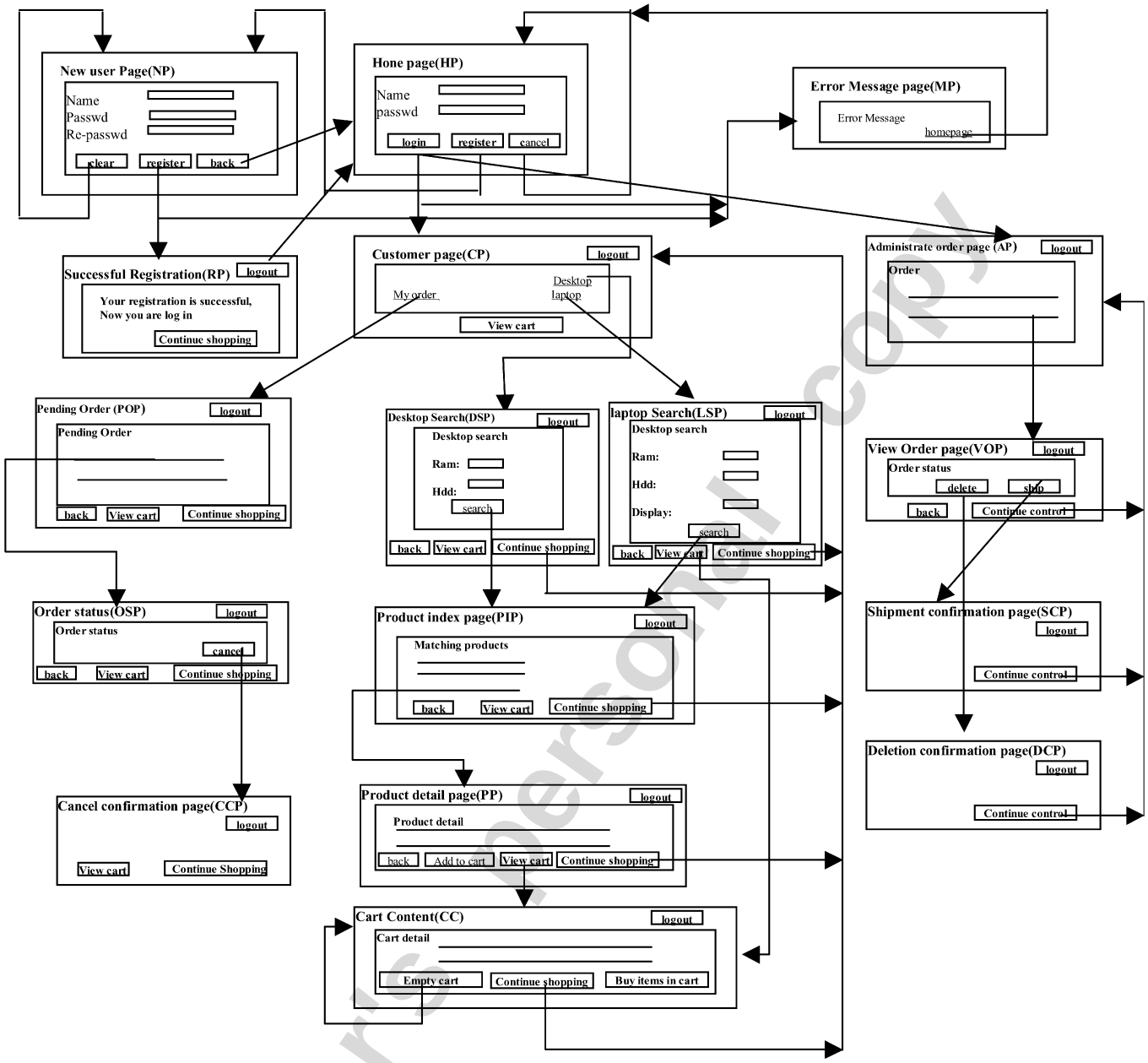

Fig. 1. Web pages in the demo.

most recent input to J occurring anywhere in the run, rather than in the previous step. Also note that $prev_J$ relations are really state relations with very specific functionality, and are redundant in the general model. However, they are very useful when defining tractable restrictions of the model.

Notation. For better readability of our examples, we use the following notation: relation R is displayed as R if it is a state relation, as R if it is an input relation, as $\underline{\mathsf{R}}$ if it is a database relation, and as $\bar{\mathsf{R}}$ if it is an action relation. In Example 2.2 below, error $\in \mathbf{S}$, $\underline{\mathsf{user}} \in \mathbf{D}$ and $\mathsf{name}, \mathsf{password}, \mathsf{button} \in \mathbf{I}$.

Example 2.2. We use as a running example throughout the paper 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 http://www.cs.ucsd.edu/~lsui/project/index.html. Figure 1 represents the Web pages of our demo, depicted in WebML style.

The demo site implements the Web application $\langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathbf{W}, \mathbf{HP}, W_\epsilon \rangle$. See Fig. 1 for an overview of all Web pages of the demo, depicted in WebML style. We list here only the pages in $\mathbf{W}$ that are mentioned in the running example:

HP	the home page
RP	the new user registration page
CP	the customer page
AP	the administrator page
LSP	a laptop search page
PIP	displays the products returned by the search
CC	allows the user to view the cart contents and order items in it
MP	an error message page

The following describes the home page HP which contains two text input boxes for the customer's user name and password respectively, and three buttons, allowing customers to register, login, respectively clear the input.

```
Page HP
Inputs I_HP : name, password, button(x)
Input Rules:
  Options_button(x) ← x = "login" ∨ x = "register" ∨ x = "clear"
State Rules:
  error(x) ← ¬user(name, password) ∧ button("login") ∧ x = "failed login"
Target Web Pages T_HP: HP, RP, CP, AP, MP
Target Rules:
  HP ← button("clear")
  RP ← button("register")
  CP ← user(name, password) ∧ button("login") ∧ name ≠ "Admin"
  AP ← user(name, password) ∧ button("login") ∧ name = "Admin"
  MP ← ¬user(name, password) ∧ button("login")
End Page HP
```

Notice how the three buttons are modeled by a single input relation button, whose argument specifies the clicked button. The corresponding input rule restricts it to a login, clear or register button only. As will be seen shortly (Definition 2.3), each input relation may contain at most one tuple at any given time, corresponding to the user's pick from the set of tuples defined by the associated input rule. This guarantees that no two buttons may be clicked simultaneously. The user name and password are modeled as input constants, as their value is not supposed to change during the session. If the login button is clicked, the state rule looks up the name/password combination in the database table user. If the lookup fails, the state rule records the login failure in the state relation error, and the last target rule fires a transition to the message page MP. Notice how the "*Admin*" user enjoys special treatment: upon login, she is directed to the admin page AP, whereas all other users go to the customer page CP. Assume that the CP page allows users to follow either a link to a desktop search page, or a laptop search page LSP. We illustrate only the laptop search functionality of the search page LSP (see the online demo for the full version, which also allows users to search for desktops).

```
Page LSP
Inputs I_LSP : laptopsearch(ram, hdisk, display), button(x)
Input Rules:
  Options_button(x) ← x = "search" ∨ x = "view cart" ∨ x = "logout"
  Options_laptopsearch(r, h, d) ← criteria("laptop", "ram", r)
                                  ∧ criteria("laptop", "hdd", h) ∧ criteria("laptop", "display", d)
State Rules:
  userchoice(r, h, d) ← laptopsearch(r, h, d) ∧ button("search")
Target Web Pages T_LSP: HP, PIP, CC
```

```
Target Rules:
  HP ← button("logout")
  PIP ← ∃r∃h∃d laptopsearch(r, h, d) ∧ button("search")
  CC ← button("view cart")
End Page LSP
```

Notice how the second input rule looks up in the database the valid parameter values for the search criteria pertinent to laptops. This enables users to pick from a menu of legal values instead of providing arbitrary ones.

We next define the notion of "run" of a Web application. Essentially, a run specifies the fixed database and consecutive Web pages, states, inputs, and actions. Thus, a run over database instance D is an infinite sequence $\{\langle V_i, S_i, I_i, P_i, A_i\rangle\}_{i\geqslant 0}$, where $V_i \in \mathbf{W}$, S_i is an instance of $\mathbf{S}$, I_i is an instance of $\mathbf{I}_{V_i}$, P_i is an instance of $\mathbf{prev_I}$, and A_i is an instance of $\mathbf{A}_{V_i}$. We call $\langle V_i, S_i, I_i, P_i, A_i\rangle$ a *configuration* of the run.

The input constants play a special role in runs. Their interpretation is not fixed a priori, but is instead provided by the user as the run progresses. We will need to make sure this occurs in a sound fashion. For example, a formula may not use an input constant before its value has been provided. We will also prevent the Web application from asking the user repeatedly for the value of the same input constant. To formalize this, we will use the following notation. For each $i \geqslant 0$, κ_i denotes the set of input constants occurring in some $\mathbf{I}_{V_j}$ in the run, $j \leqslant i$, and σ_i denotes the mapping associating to each $c \in \kappa_i$ the unique $I_j(c)$ where $j \leqslant i$ and $c \in \mathbf{I}_{V_j}$.

Definition 2.3. Let $\mathcal{W} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathbf{W}, W_0, W_\epsilon\rangle$ be a Web application and D a database instance over schema $\mathbf{D}$. A *run* of $\mathcal{W}$ for database D is an infinite sequence of configurations $\{\langle V_i, S_i, I_i, P_i, A_i\rangle\}_{i\geqslant 0}$ where $V_i \in \mathbf{W}$, S_i is an instance of $\mathbf{S}$, I_i is an instance of $\mathbf{I}_{V_i}$, P_i is an instance of $\mathbf{prev_I}$, A_i is an instance of $\mathbf{A}_{V_i}$ and:

- $V_0 = W_0$, and S_0, A_0, P_0 are empty;
- for each $i \geqslant 0$, $V_{i+1} = W_\epsilon$ if one of the following holds:
  - (i) some formula used in a rule of V_i involves a constant symbol $c \in \mathbf{I}$ that is not in κ_i;
  - (ii) $\mathbf{I}_{V_i} \cap \kappa_{i-1} \neq \emptyset$;
  - (iii) there are distinct $W, W' \in \mathbf{T}_{V_i}$ for which φ_{W,V_i} and φ_{W',V_i} are both true when evaluated on D, S_i, I_i and P_i, and interpretation σ_i for the input constants occurring in the formulas;

  otherwise, V_{i+1} is the unique $W \in \mathbf{T}_{V_i}$ for which φ_{W,V_i} is true when evaluated on D, S_i, I_i, P_i and σ_i if such W exists; if not, $V_{i+1} = V_i$;
- for each $i \geqslant 0$, and for each relation R in $\mathbf{I}_{V_i}$ of arity $k > 0$, $I_i(R) \subseteq \{v\}$ for some $v \in \mathit{Options}_R$, where $\mathit{Options}_R$ is the result of evaluating φ_{R,V_i} on D, S_i, P_i and σ_i;
- for each $i \geqslant 0$, and for each proposition R in $\mathbf{I}_{V_i}$, $I_i(R)$ is a truth value;
- for each $i \geqslant 0$, and for each constant symbol c in $\mathbf{I}_{V_i}$, $I_i(c)$ is an element in $\mathbf{dom}_\infty$;
- for each $i \geqslant 0$, and for each relation prev_I in $\mathit{prev}_\mathbf{I}$, $P_i(\mathit{prev}_I) = I_{i-1}(I)$ if $I \in \mathbf{I}_{V_{i-1}}$ and $P_i(\mathit{prev}_I)$ is empty otherwise;
- for each $i \geqslant 0$, and relation S in $\mathbf{S}$, $S_{i+1}(S)$ is the result of evaluating

$$\big(\varphi^+_{S,V_i}(\bar{x}) \wedge \neg\varphi^-_{S,V_i}(\bar{x})\big) \vee \big(S(\bar{x}) \wedge \varphi^-_{S,V_i}(\bar{x}) \wedge \varphi^+_{S,V_i}(\bar{x})\big) \vee \big(S(\bar{x}) \wedge \neg\varphi^-_{S,V_i}(\bar{x}) \wedge \neg\varphi^+_{S,V_i}(\bar{x})\big)$$

  on D, S_i, I_i, P_i and σ_i, where $\varphi^\epsilon_{S,V_i}(\bar{x})$ is taken to be *false* if it is not provided in the Web page schema ($\epsilon \in \{+,-\}$). In particular, S remains unchanged if no insertion or deletion rule is specified for it;
- for each $i \geqslant 0$, and relation A in $\mathbf{A}_{V_{i+1}}$, $A_{i+1}(A)$ is the result of evaluating φ_{A,V_i} on D, S_i, I_i, P_i and σ_i.

Note that the state and actions specified at step $i+1$ in the run are those triggered at step i. This choice is convenient for technical reasons. As discussed above, input constants are provided an interpretation as a result of user input, and need not be values already existing in the database. Once an interpretation is provided for a constant symbol, it can be used in the formulas determining the run. For example, such constant symbols might include **name**, **password**, **credit-card**, etc. Note that, as a consequence of the definition of a run over database D, all configurations in a run are instances over the finite domain of D extended with interpretations for $\mathit{const}(\mathbf{I})$. In particular, each run, although infinite, has only finitely many distinct configurations.

The error Web page serves an important function, since it signals behavior that we consider anomalous. Specifically, the error Web page is reached in the following situations:

(i) the value of an input constant is required by some formula before it was provided by the user;
(ii) the user is asked to provide a value for the same input constant more than once; and
(iii) the specification of the next Web page is ambiguous, since it produces more than one Web page.

Once the error page is reached, the run loops forever in that page. We call a run *error free* if the error Web page is not reached, and we call a Web application error-free if it generates only error-free runs. Clearly, it would be desirable to verify that a given Web application is error-free. As we will see, this can be expressed in the temporal logics we consider and can be checked for restricted classes of Web applications.

3. Temporal properties of Web applications

In this section we introduce the temporal languages used for specifying properties of Web applications. These are variants of linear-time and branching time temporal logics.

3.1. Linear-time temporal logic

We begin with linear-time properties, that must be satisfied by *all* runs of a Web application. Such properties are expressed using a variant of linear-time temporal logic, adapted from [1,18,37]. We begin by defining this logic, that we denote LTL-FO (first-order linear-time temporal logic).

Definition 3.1. [1,18,37] The language LTL-FO (first-order linear-time temporal logic) is obtained by closing FO under negation, disjunction, and the following formula formation rule: If φ and ψ are formulas, then $\mathbf{X}\varphi$ and $\varphi\mathbf{U}\psi$ are formulas. Free and bound variables are defined in the obvious way. The *universal closure* of an LTL-FO formula $\varphi(\bar{x})$ with free variables $\bar{x}$ is the formula $\forall\bar{x}\,\varphi(\bar{x})$. An LTL-FO sentence is the universal closure of an LTL-FO formula.

Note that quantifiers cannot be applied to formulas containing temporal operators, except by taking the universal closure of the entire formula, yielding an LTL-FO sentence. For a given LTL-FO sentence, we refer to its maximal subformulas containing no temporal operators as the *FO components* of the sentence.

Let $\mathcal{W} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathbf{W}, W_0, W_\epsilon \rangle$ be a Web application. To express properties of runs of $\mathcal{W}$, we use LTL-FO sentences over schema $\mathbf{D} \cup \mathbf{S} \cup \mathbf{I} \cup \mathbf{Prev_I} \cup \mathbf{A} \cup \mathbf{W}$, where each $W \in \mathbf{W}$ is used as a propositional variable. The semantics of LTL-FO formulas is standard, and we describe it informally. Let $\forall\bar{x}\,\varphi(\bar{x})$ be an LTL-FO sentence over the above schema. The Web application $\mathcal{W}$ satisfies $\forall\bar{x}\,\varphi(\bar{x})$ iff every run of $\mathcal{W}$ satisfies it. Let $\rho = \{\rho_i\}_{i\geqslant 0}$ be a run of $\mathcal{W}$ for database D, and let $\rho_{\geqslant j}$ denote $\{\rho_i\}_{i\geqslant j}$, for $j \geqslant 0$. Note that $\rho = \rho_{\geqslant 0}$. Let $\mathrm{Dom}(\rho)$ be the active domain of ρ, i.e. the set of all elements occurring in relations or as interpretations for constants in ρ. The run ρ satisfies $\forall\bar{x}\,\varphi(\bar{x})$ iff for each valuation ν of $\bar{x}$ in $\mathrm{Dom}(\rho)$, $\rho_{\geqslant 0}$ satisfies $\varphi(\nu(\bar{x}))$. The latter is defined by structural induction on the formula. An FO sentence ψ is satisfied by $\rho_i = \langle V_i, S_i, I_i, P_i, A_i \rangle$ if the following hold:

- the set of input constants occurring in ψ is included in κ_i;
- the structure ρ_i' satisfies ψ, where ρ_i' is the structure obtained by augmenting ρ_i with interpretation σ_i for the input constants. Furthermore, ρ_i assigns *true* to V_i and *false* to all other propositional symbols in $\mathbf{W}$.

The semantics of Boolean operators is the obvious one. The meaning of the temporal operators $\mathbf{X}, \mathbf{U}$ is the following (where $\models$ denotes satisfaction and $j \geqslant 0$):

- $\rho_{\geqslant j} \models \mathbf{X}\varphi$ iff $\rho_{\geqslant j+1} \models \varphi$,
- $\rho_{\geqslant j} \models \varphi\mathbf{U}\psi$ iff $\exists k \geqslant j$ such that $\rho_{\geqslant k} \models \psi$ and $\rho_{\geqslant l} \models \varphi$ for $j \leqslant l < k$.

Observe that the above temporal operators can simulate all commonly used operators, including **B** (before), **G** (always) and **F** (eventually). Indeed, $\varphi \mathbf{B} \psi$ is equivalent to $\neg(\neg\varphi \mathbf{U} \neg\psi)$, $\mathbf{G}\varphi \equiv \mathit{false}\, \mathbf{B}\varphi$, and $\mathbf{F}\varphi \equiv \mathit{true}\, \mathbf{U}\varphi$. We use the above operators as shorthand in LTL-FO formulas whenever convenient.

LTL-FO sentences can express many interesting properties of a Web application. A useful class of properties pertains to the navigation between pages.

Example 3.2. The following property states that if page P is reached in a run, then page Q will be eventually reached as well:

$$\mathbf{G}(\neg \mathrm{P}) \vee \mathbf{F}(\mathrm{P} \wedge \mathbf{F}\mathrm{Q}). \tag{1}$$

Another important class of properties describes the flow of the interaction between user and application.

Example 3.3. Assume that the Web application in Example 2.2 allows the user to pick a product and records the pick in a state relation pick(*product_id*, *price*). There is also a payment page PP, with input relation pay(*amount*) and "*authorize payment*" button. Clicking this button authorizes the payment of *amount* for the product with identifier recorded in state pick, on behalf of the user whose name was provided by the constant name (recall page HP from Example 2.2). Also assume the existence of an order confirmation page OCP, containing the actions $\overline{\mathsf{conf}}$(*user_id*, *price*) (which confirms the price to the user) and $\overline{\mathsf{ship}}$(*user_id*, *product_id*) (which places the shipment order). The following property involving state, action, input and database relations requires that any shipped product be previously paid for:

$$\forall \mathit{pid}, \mathit{price}\, \big[\xi(\mathit{pid}, \mathit{price}) \mathbf{B} \neg\big(\overline{\mathsf{conf}}(\mathsf{name}, \mathit{price}) \wedge \overline{\mathsf{ship}}(\mathsf{name}, \mathit{pid})\big)\big] \tag{2}$$

where $\xi(\mathit{pid}, \mathit{price})$ is the formula

$$\mathrm{PP} \wedge \mathsf{pay}(\mathit{price}) \wedge \mathsf{button}(\text{"}\mathit{authorize\ payment}\text{"}) \wedge \mathsf{pick}(\mathit{pid}, \mathit{price}) \wedge \exists \mathit{pname}\, \underline{\mathsf{catalog}}(\mathit{pid}, \mathit{price}, \mathit{pname}). \tag{3}$$

3.2. Branching-time temporal logics

Branching-time logics allow to express temporal properties involving quantification over runs. For example, such quantification is needed to express the property "At any point in a run, there is a way to return to the shopping cart page."

We next provide the syntax and semantics of the branching-time logics CTL-FO and CTL*-FO, adapted from [1,37]. These are extensions of the well-known languages CTL and CTL* (see [18]). We also review the notion of satisfaction of a CTL$^{(*)}$ formula by a Kripke structure.

Definition 3.4. Let $\mathcal{W} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathbf{W}, W_0, W_\epsilon \rangle$ be a Web application. The set of CTL*-FO formulas over $\mathcal{W}$ is the set of state formulas defined inductively together with the set of path formulas as follows:

1. each FO formula over the vocabulary of $\mathcal{W}$ is a state formula;
2. if φ and ψ are state formulas then so are $\varphi \wedge \psi$, $\varphi \vee \psi$, and $\neg\varphi$;
3. if φ is a path formula, then $\mathrm{E}\varphi$ and $\mathrm{A}\varphi$ are state formulas;
4. each state formula is also a path formula;
5. if φ and ψ are path formulas then so are $\varphi \wedge \psi$, $\varphi \vee \psi$, and $\neg\varphi$;
6. if φ and ψ are path formulas then so are $\mathbf{X}\varphi$ and $\varphi \mathbf{U} \psi$.

The set of CTL-FO formulas over $\mathcal{W}$ is defined by replacing (4)–(6) above by the rule:

- if φ and ψ are state formulas then $\mathbf{X}\varphi$, and $\varphi \mathbf{U} \psi$ are path formulas.

The set of CTL$^{(*)}$-FO sentences consists of the universal closures of CTL$^{(*)}$-FO formulas.

Note that, as in the case of LTL-FO, first-order quantifiers cannot be applied to formulas using temporal operators or path quantifiers. The formula is closed at the very end by universally quantifying all remaining free variables,

yielding an CTL$^{(*)}$-FO sentence. The semantics of the temporal operators is the natural extension of LTL-FO with path quantifiers. Informally, $\mathbf{E}\varphi$ stands for "there exists a continuation of the current run that satisfies φ" and $\mathbf{A}\varphi$ means "every continuation of the current run satisfies φ." More formally, satisfaction of a CTL$^{(*)}$-FO sentence by a Web application $\mathcal{W}$ is defined using the tree corresponding to the runs of $\mathcal{W}$ on a given database D. The nodes of the tree consist of all prefixes of runs of $\mathcal{W}$ on D (the empty prefix is denoted *root* and is the root of the tree). A prefix π is a child of a prefix π' iff π extends π' with a single configuration. We denote the resulting infinite tree by $\mathcal{T}_{\mathcal{W},D}$. Note that $\mathcal{T}_{\mathcal{W},D}$ has only infinite branches (so no leafs) and each such infinite branch corresponds to a run of $\mathcal{W}$ on database D. Satisfaction of an CTL$^{(*)}$-FO sentence by $\mathcal{T}_{\mathcal{W},D}$ is the natural extension of the classical notion of satisfaction of CTL$^{(*)}$ formulas by infinite trees labeled with propositional variables (e.g., see [18]), and is provided below. The main difference is that propositional variables are not explicitly provided; instead, the FO components of the formulas have to be evaluated on the current configuration (last of the prefix defining the node), as described earlier. We say that a Web application $\mathcal{W}$ satisfies φ, denoted $\mathcal{W} \models \varphi$, iff $\mathcal{T}_{\mathcal{W},D} \models \varphi$ for every database D.

We review the classical notion of satisfaction of a CTL$^{(*)}$ formula by a Kripke structure (see[18]). The language CTL$^{(*)}$ is the restriction of CTL$^{(*)}$-FO where all FO formulas are propositions.

Definition 3.5. Let $AP = \{p_1, p_2, \ldots, p_n\}$ be a finite set of atomic propositional symbols. A *Kripke structure* over AP is a 4-tuple $K = (S, S^0, R, L)$ where:

- S is a finite set of states.
- $S^0 \in S$ is an initial state.
- R is a total binary relation on S ($R \subseteq S \times S$), called the *transition relation*.
- $L : S \to 2^{AP}$ assigns to each state the set of atomic propositions which are true in that state.

A *path* ρ in Kripke structure K is an infinite sequence of states $(s_0, s_1, \ldots)$ such that $(s_i, s_{i+1}) \in R$ for every $i \geqslant 0$. Let $\rho_{\geqslant i}$ denote the suffix path $(s_i, s_{i+1}, s_{i+2}, \ldots)$. The notation $K, s \models p$ indicates that a CTL* state formula p holds at state s of the Kripke structure K. Similarly, $K, \rho \models \psi$ indicates that a CTL* path formula ψ holds at a path of ρ of the Kripke structure K. We write $s \models p$ or $\rho \models \psi$ when it is obvious from the context which structure is concerned.

The notion of satisfaction of a CTL* formula by a Kripke structure is defined as follows:

1. $s \models p$ iff $p \in L(s)$.
2. $s \models \neg p$ iff $p \notin L(s)$.
3. $s \models \varphi_1 \wedge (\vee) \varphi_2$ iff $s \models \varphi_1$ and(or) $s \models \varphi_2$.
   $s \models \mathbf{E}\psi$ iff there exists an infinite path $\rho' = (s, s_1, s_2, \ldots)$ in K, starting from s, such that $\rho' \models \psi$.
   $s \models \mathbf{A}\psi$ iff for every infinite path $\rho' = (s, s_1, s_2, \ldots)$ in K starting from s, $\rho' \models \psi$.
4. $\rho_{\geqslant j} \models \psi$ iff $s' \models \psi$ where s' is the first state in $\rho_{\geqslant j}$.
5. $\rho_{\geqslant j} \models \psi_1 \wedge (\vee) \psi_2$ iff $\rho_{\geqslant j} \models \psi_1$ and(or) $\rho_{\geqslant j} \models \psi_2$.
   $\rho_{\geqslant j} \models \neg\psi$ iff $\rho_{\geqslant j} \not\models \psi$.
6. $\rho_{\geqslant j} \models \psi_1 \mathbf{U} \psi_2$ iff there exists $i \geqslant j$ such that $\rho_{\geqslant i} \models \psi_2$ and $\rho_{\geqslant k} \models \psi_1$ for all $j \leqslant k < i$.
   $\rho_{\geqslant j} \models \mathbf{X}\psi$ iff $\rho_{\geqslant j+1} \models \psi$.

A formula of CTL is also interpreted using the CTL* semantics. The complexity of checking whether a CTL$^{(*)}$ formula is satisfied by a Kripke structure (model checking) is in PTIME for CTL and PSPACE-complete for CTL*. The satisfiability problem for CTL$^{(*)}$ formulas is EXPTIME-complete for CTL and 2-EXPTIME complete for CTL*. See [18] for a concise survey on temporal logics, and further references.

Example 3.6. The following CTL*-FO sentence expresses the fact that in every run, whenever a product *pid* is bought by a user, it will eventually ship, but until that happens, the user can still cancel the order for *pid*.

$$\forall pid \forall price\, A\mathbf{G}\big(\xi(pid, price)\big) \to A\Big(\big(E\mathbf{F}\overline{\mathsf{cancel}}(\mathsf{name}, pid)\big)\mathbf{U}\big(\overline{\mathsf{ship}}(\mathsf{name}, pid)\big)\Big)$$

where ξ is the formula

$$\mathrm{PP} \wedge \mathsf{pay}(price) \wedge \mathsf{button}(\text{“}\textit{authorize payment}\text{”}) \wedge \mathrm{pick}(pid, price) \wedge \underline{\mathsf{prod\_prices}}(pid, price). \tag{4}$$

4. ASM$^+$ transducers

In this section we present an extension of Spielmann's Abstract State Machine (ASM) transducers [36,37] and prove our verification results within this framework. We denote the extended transducer model by ASM$^+$. Informally, ASM$^+$ transducers extend Spielmann's ASM transducer model with the ability to inspect the previous user input, and with input option rules constraining the choice of input by the user. The main interest of the extension is that, as we shall see, it is sufficiently powerful to simulate a wide class of Web applications. We next define ASM$^+$ transducers formally.

Definition 4.1. An ASM$^+$ transducer $\mathcal{A}$ is a tuple $\langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathcal{R} \rangle$, where:

- $\mathbf{D}$, $\mathbf{S}$, $\mathbf{I}$, $\mathbf{A}$ are relational schemas called database, state, input, and action schemas, respectively, where $\mathbf{S}$, $\mathbf{I}$, $\mathbf{A}$ contain no constant symbols. 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 of rules containing the following:
  - For each input relation $I \in \mathbf{I}$ of arity $k > 0$, an *input rule* $Options_I(\bar{x}) \leftarrow \varphi_I(\bar{x})$ where $Options_I$ is a relation of arity k, $\bar{x}$ is a k-tuple of distinct variables, and $\varphi_I(\bar{x})$ is an FO formula over schema $\mathbf{D} \cup \mathbf{S} \cup \mathbf{Prev_I}$, with free variables $\bar{x}$.
  - For each state relation $S \in \mathbf{S}$, one, both, or none of 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 $\varphi_S^\epsilon(\bar{x})$ are FO formulas over schema $\mathbf{D} \cup \mathbf{S} \cup \mathbf{Prev_I} \cup \mathbf{I}$, with free variables $\bar{x}$.
  - For each action relation $A \in \mathbf{A}$, an *action rule* $A(\bar{x}) \leftarrow \varphi(\bar{x})$ where the arity of A is k, $\bar{x}$ is a k-tuple of distinct variables, and $\varphi(\bar{x})$ is an FO formula over schema $\mathbf{D} \cup \mathbf{S} \cup \mathbf{Prev_I} \cup \mathbf{I}$, with free variables $\bar{x}$.

Note that an ASM$^+$ transducer is isomorphic to a Web application with a single Web page schema, and no input constants. The definition of runs, as well as the syntax and semantics of LTL-FO and CTL$^{(*)}$-FO formulas, are therefore inherited from Web applications. Also, any lower bounds for verification problems proven for ASM$^+$ transducers apply trivially to Web applications. To transfer the upper bounds, we will need to show appropriate reductions from Web applications to the simpler ASM$^+$ transducers.

For completeness, we briefly describe configurations and runs of ASM$^+$ transducers. Let $\mathcal{A} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathcal{R} \rangle$ be an ASM$^+$ transducer. A configuration of $\mathcal{A}$ is a tuple $\langle S, I, P, A \rangle$, where S is an instance of $\mathbf{S}$, I and P are instances of $\mathbf{I}$ and $\mathbf{Prev_I}$ consisting of at most one tuple, and A is an instance of $\mathbf{A}$. Let D be an instance of $\mathbf{D}$. A *run* of $\mathcal{A}$ on database D is an infinite sequence of configurations $\{\langle S_i, I_i, P_i, A_i \rangle\}_{i \geqslant 0}$ where:

- all relations in S_0, P_0, and A_0 are empty and all propositions are false;
- I_0 consists of at most one tuple for each input relation, belonging to the result of evaluating the corresponding input option rule; and
- for $i \geqslant 0$, $\langle S_{i+1}, I_{i+1}, P_{i+1}, A_{i+1} \rangle$ is obtained from D and $\langle S_i, I_i, P_i, A_i \rangle$ using the rules $\mathcal{R}$, as done for Web applications with a single Web page (details omitted, see Definition 2.3).

Note that each configuration in a run over database D uses values from the domain of D. In particular, every run, although infinite, has only finitely many distinct configurations.

We next present our results on verification of ASM$^+$ transducers, first for linear-time properties, then for branching-time properties.

4.1. Verification of LTL-FO properties

It is easily seen that it is undecidable if an ASM$^+$ transducer satisfies an LTL-FO formula, as shown next.

Proposition 4.2. *It is undecidable, given an ASM$^+$ transducer $\mathcal{A}$ and an LTL formula ψ, whether $\mathcal{A} \models \psi$.*

Proof. By Trakhtenbrot's theorem, finite satisfiability of FO sentences is undecidable. Let $\mathbf{D}$ be a database schema and φ an FO sentence over $\mathbf{D}$. Consider an ASM$^+$ transducer with database schema $\mathbf{D}$ and an action rule $A \leftarrow \varphi$, where A is a proposition. Clearly, φ is finitely satisfiable iff $\mathcal{A} \not\models \mathbf{G}\neg A$. □

To obtain decidability, we must restrict both the transducers and the LTL-FO sentences. We use a restriction proposed in [36,37] for ASM transducers, limiting the use of quantification in state and action rule formulas to "input-bounded" quantification, and an additional restriction on input rules. The restrictions are formulated in our framework as follows. Let $\mathcal{A} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathcal{R} \rangle$ be an ASM$^+$ transducer. The set of *input-bounded* FO formulas over the schema $\mathbf{D} \cup \mathbf{S} \cup \mathbf{I} \cup \mathbf{A} \cup \mathbf{Prev_I}$ 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 $\mathbf{I} \cup \mathbf{Prev_I}$, $\bar{x} \subseteq \mathit{free}(\alpha)$, and $\bar{x} \cap \mathit{free}(\beta) = \emptyset$ for every state or action 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 iff all formulas in state and action rules are input bounded, and all input rules use $\exists^*$FO formulas in which all state atoms are ground (note that the input option rules do not have to obey the restricted quantification formation rule above). An LTL-FO sentence over the schema of $\mathcal{A}$ is input-bounded iff all of its FO components are input-bounded.

It was shown in [36,37] that checking satisfaction of an input-bounded LTL-FO sentence by an input-bounded ASM transducer is PSPACE-complete. The lower bound, shown by reduction of quantified Boolean formula [21], transfers immediately to input-bounded ASM$^+$ transducers and LTL-FO formulas, since ASM transducers are special cases of ASM$^+$ transducers. The PSPACE upper bound is shown in [36,37] by reducing the verification problem to finite satisfiability in the logic E + TC, existential FO extended with a transitive closure operator. With some care, this proof can be adapted to ASM$^+$ transducers. However, the proof we provide is considerably more direct, circumventing the laborious reduction to E + TC and the proof of decidability of finite satisfiability for this logic. It also provides an alternative proof to Spielmann's original result on input-bounded ASM transducers. Furthermore, the construction used in our proof provides the basis for a practical implementation of a verifier for Web applications, described in [15,17].

We next present our 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 $\mathcal{A} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathcal{R} \rangle$. For simplicity of exposition, we assume that $\mathbf{I}$ consists of only one input relation (the development easily extends to multiple input relations). In particular, a configuration of $\mathcal{A}$ is a tuple $\langle S, I, P, A \rangle$ where S is an instance of $\mathbf{S}$, A is an instance of $\mathbf{A}$, 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 an input-bounded LTL-FO formula $\varphi_0 = \forall \bar{x}\, \psi_0(\bar{x})$ over the schema of $\mathcal{A}$. To check that every run of $\mathcal{A}$ satisfies φ_0 we equivalently verify that there is no run of $\mathcal{A}$ satisfying $\neg\varphi_0 = \exists \bar{x} \neg \psi_0(\bar{x})$. To do so, we would like to adapt classical model checking based on Büchi automata. We informally recall this approach (see e.g. [11] for a formal development).

4.1.1. Propositional model checking

Classical model checking applies to finite state transition systems. 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 ([11], see also Section 3.2). 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 $i \geqslant 0$. 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.

Given a transition system $\mathcal{T}$ as above and an LTL formula φ using propositions in P, the associated model checking problem is to check whether every run of $\mathcal{T}$ satisfies φ, or equivalently, that no run of $\mathcal{T}$ satisfies $\neg\varphi$. This can be done using a key result of [38], showing that from each LTL formula ϕ over P one can construct an automaton A_ϕ on infinite sequences, called a Büchi automaton, whose alphabet consists of the truth assignments to P, and which

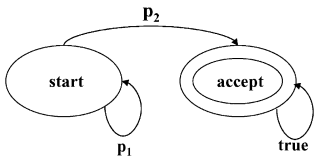

Fig. 2. Büchi automaton for $\varphi^{aux} = p_1 \mathbf{U} p_2$.

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 $A_{\neg\varphi}$.

We briefly recall Büchi automata. A Büchi automaton A is a non-deterministic finite state automaton (NFA) 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 [38] show that for every LTL formula φ there exists Büchi automaton A_φ of size exponential in φ that accepts precisely the infinite sequences of truth assignments that satisfy φ. Furthermore, given a state p of A_φ and a truth assignment σ, the set of possible next states of A_φ under input σ can be computed directly from p and φ in polynomial space [35]. This allows to generate computations of A_φ without explicitly constructing A_φ.

Example 4.3. Figure 2 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 $\mathcal{T}$ whose configurations can be enumerated in PSPACE with respect to the specification of $\mathcal{T}$, and such that, given configurations s, s', it can be checked in PSPACE whether $\langle s, s' \rangle$ is a transition in $\mathcal{T}$. Suppose φ is an LTL formula 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 $A_{\neg\varphi}$, non-deterministically extend the current run of $\mathcal{T}$ with a new configuration s, and transition to a next state of $A_{\neg\varphi}$ under input $\sigma(s)$, until an accepting state f of $A_{\neg\varphi}$ 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 $A_{\neg\varphi}$ and configuration s of $\mathcal{T}$ coincide with the current state in $A_{\neg\varphi}$ and configuration in $\mathcal{T}$, then stop and answer "yes." This shows that model checking is in non-deterministic PSPACE, and therefore in PSPACE.

4.1.2. From classical model checking to ASM$^+$ verification

There are two main obstacles to using classical model checking to verify ASM$^+$ transducers. First, LTL-FO formulas are not propositional. Second, the transition systems corresponding to ASM$^+$ transducers are not finite state, since they have infinitely many possible configurations. We next show how to overcome both obstacles.

Consider an input-bounded ASM$^+$ transducer $\mathcal{A}$ and an input-bounded 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 distinct constant symbols 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 interpretation of the constants $\bar{c}$. Let us denote $\psi(\bar{x} \leftarrow \bar{c})$ by $\psi_{\bar{c}}$. Consider now a maximal subformula ξ of $\psi_{\bar{c}}$ that contains no temporal operator, which we call an FO component of $\psi_{\bar{c}}$. Note that ξ has no free variables (as variables previously free in ξ have been replaced by the constant symbols $\bar{c}$). Thus, ξ can be evaluated to true or false in every configuration of a run of $\mathcal{A}$. This allows treating every such ξ as a proposition. More precisely, for each FO component ξ of $\psi_{\bar{c}}$, let p_ξ be a propositional symbol. Let $\psi_{\bar{c}}^{aux}$ be the LTL formula obtained by replacing in $\psi_{\bar{c}}$ every FO component ξ by p_ξ. For each configuration of $\mathcal{A}$, the truth value of p_ξ is defined as the truth value of ξ. Clearly, a run of $\mathcal{A}$ satisfies $\psi_{\bar{c}}$ iff it satisfies $\psi_{\bar{c}}^{aux}$. Specifically, for $i \geqslant 0$, let $\sigma(\rho_i)$ be the truth assignment to the propositions in $\psi_{\bar{c}}^{aux}$ such that p_ξ is true iff $\rho_i \models \xi$, and let $\sigma(\rho) = \{\sigma(\rho_i)\}_{i \geqslant 0}$. Let us denote by $A_{\psi_{\bar{c}}}$ the Büchi automaton corresponding to the propositional LTL formula $\psi_{\bar{c}}^{aux}$. A run of $A_{\psi_{\bar{c}}}$ on $\sigma(\rho)$ is an infinite sequence of states $q_0, s_0, s_1, \ldots, s_i, \ldots$ such that q_0 is the start state of $A_{\psi_{\bar{c}}}$, and $\langle q_0, \sigma(\rho_0), s_0 \rangle$, $\langle s_i, \sigma(\rho_{i+1}), s_{i+1} \rangle$ are transitions in $A_{\psi_{\bar{c}}}$ for each $i \geqslant 0$. Clearly, $\rho \models \psi_{\bar{c}}$ iff there exists a run of $A_{\psi_{\bar{c}}}$ on input $\sigma(\rho)$ that goes through some accepting state, say f, infinitely often.

Example 4.4. The following LTL-FO property referring to Example 2.2 states that any confirmed product must have previously been paid for:

$$\begin{aligned}&\forall pid, category, name, ram, hdd, display, price\\ &\quad\big(\text{UPP} \wedge \mathsf{button}(\text{“}submit\text{”}) \wedge \text{cart}(pid, price)\\ &\qquad \wedge \underline{\mathsf{products}}(pid, category, name, ram, hdd, display, price)\big)\\ &\mathbf{B}\quad \neg\overline{\mathsf{conf}}(pid, category, name, ram, hdd, display, price).\end{aligned} \tag{5}$$

Property (5) is negated to the following formula ψ:

$$\begin{aligned}&\exists pid, category, name, ram, hdd, display, price\\ &\quad\neg\big(\text{UPP} \wedge \mathsf{button}(\text{“}submit\text{”}) \wedge \text{cart}(pid, price)\\ &\qquad \wedge \underline{\mathsf{products}}(pid, category, name, ram, hdd, display, price)\big)\\ &\mathbf{U}\quad \overline{\mathsf{conf}}(pid, category, name, ram, hdd, display, price).\end{aligned} \tag{6}$$

Let $\bar{c}$ be a sequence of constants $pid_0, category_0, name_0, ram_0, hdd_0, display_0, price_0$. By replacing the existentially quantified variables with $\bar{c}$, we obtain $\psi_{\bar{c}}$:

$$\begin{aligned}&\neg\big(\text{UPP} \wedge \mathsf{button}(\text{“}submit\text{”}) \wedge \text{cart}(pid_0, price_0)\\ &\quad \wedge \underline{\mathsf{products}}(pid_0, category_0, name_0, ram_0, hdd_0, display_0, price_0)\big)\\ &\mathbf{U}\quad \overline{\mathsf{conf}}(pid_0, category_0, name_0, ram_0, hdd_0, display_0, price_0)\end{aligned}$$

which yields the propositional property $\psi_{\bar{c}}^{aux}$

$$p_1 \mathbf{U} p_2 \tag{7}$$

where p_1, p_2 are the new propositional symbols introduced for the FO formulae to the left, respectively right of the temporal operator **U** in (7). We have already seen in Fig. 2 the Büchi automaton corresponding to property (7).

In the simple example above, the FO components of $\psi_{\bar{c}}$ happen to be quantifier free. In general however, FO components may have input-bounded quantifiers.

Next, we address the harder issue of the infinite number of configurations of $\mathcal{A}$. Here we make crucial use of the input-boundedness restriction. Let $\mathcal{A} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathcal{R}\rangle$ be an input-bounded ASM$^+$ transducer, and φ an input-bounded LTL-FO formula. Let $\psi_{\bar{c}}$ be obtained from φ as described above, for some sequence $\bar{c}$ of constant symbols. Let C be the set of constant symbols in the database schema **D**. We can assume without loss of generality that $\bar{c}$ belong to C (otherwise we extend the database schema to include $\bar{c}$). Let D be a database instance of **D**, and let $\rho = \{\langle S_i, I_i, P_i, A_i\rangle\}_{i\geqslant 0}$ be a run of $\mathcal{A}$ on D. Let C_D be the set of all domain elements that are interpretations of constant symbols in C in the instance D. We say that two instances H and H' over the same database schema are C_D-isomorphic iff there exists an isomorphism from H to H' that is the identity on C_D. The C_D-isomorphism type of H consists of all instances H' that are C_D-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 $\langle S_i, I_i, P_i, A_i\rangle$ is completely determined by the restriction[3] of S_i and A_i to C_D, together with the C_D-isomorphism type of the subinstance of $\langle I_i, P_i, D\rangle$ restricted to C_D together with the elements in I_i, P_i. Since I_i and P_i contain at most one tuple each, the number of such C_D-isomorphism types is finite. Moreover, due to the input-boundedness of the state transition rules, the same information about $\langle S_{i+1}, I_{i+1}, P_{i+1}, A_{i+1}\rangle$ is determined by the corresponding information about $\langle S_i, I_i, P_i, A_i\rangle$. This will allow us to limit ourselves to inspecting a transition system whose configurations are the finitely many C_D-isomorphism types as above. This essentially reduces verification back to a classical model checking problem and, with some care, yields a PSPACE verification algorithm. We provide the details next.

For an instance K and a set T of elements, let $K|_T$ denote the restriction of K to T. Using the same notation as above, let $\rho_i = \langle S_i, I_i, P_i, A_i\rangle$ be a configuration in a run ρ of $\mathcal{A}$ on D. Let k be the arity of I, and C^i_{IP} consist of C_D together with all elements in $I_i \cup P_i$ (note that $I_i \cup P_i$ contains at most $2k$ elements). Let $D_{i,\bar{x}}$ be the restriction of D to C^i_{IP} together with witnesses to the existentially quantified variables $\bar{x}$ in the input-options formula $\exists \bar{x}\, \varphi_I(\bar{x})$ for I,

[3] 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.

satisfied by $D, P_i, I_i, S_i|C_D$. Let $\rho_i^{\downarrow} = \langle S_i|C_D, I_i, P_i, A_i|C_D, D_{i,\bar{x}}\rangle$. We refer to the sequence $\{\rho_i^{\downarrow}\}_{i\geqslant 0}$ as the *local run*[4] of ρ. We say that a sequence $\{\rho_i'\}_{i\geqslant 0}$ is a local run of $\mathcal{A}$ on D if it is the local run of some run of $\mathcal{A}$ on D.

Lemma 4.5. *Let $\mathcal{A}$ be an input-bounded ASM$^+$ transducer, φ an input-bounded LTL-FO formula, and $\bar{c}$, $\psi_{\bar{c}}$, D, and ρ be as above. Let ξ be an FO component of $\psi_{\bar{c}}$. Then for each configuration ρ_i in the run ρ, $\rho_i \models \xi$ iff $\rho_i^{\downarrow} \models \xi$.*

Proof. Let $\rho_i = \langle S_i, I_i, P_i, A_i\rangle$. We show by induction the following:

(†) for every subformula $\xi'(\bar{x})$ of ξ with free variables $\bar{x}$, and sequence $\bar{e}$ of elements in C^i_{IP} of the same arity as $\bar{x}$, $\rho_i \models \xi'(\bar{x} \leftarrow \bar{e})$ iff $\rho_i^{\downarrow} \models \xi'(\bar{x} \leftarrow \bar{e})$.

As a consequence of (†), $\rho_i \models \xi$ iff $\rho_i^{\downarrow} \models \xi$, since ξ has no free variables.

Consider (†). We can assume wlog that ξ uses only $\wedge$, $\neg$ and $\exists$. For the basis, suppose $\xi'(\bar{x})$ is an atom $R(t_1, \ldots, t_m)$ where each t_i is an element in C_D or a variable in $\bar{x}$. If R is a state or action relation, all t_i's are elements in C_D by input boundedness, so (†) holds because $\rho_i^{\downarrow}$ retains $S|C_D$ and $A|C_D$. If R is an input or database relation, then again (†) holds because $\rho_i^{\downarrow}$ retains I_i, P_i, and $D|_{C^i_{IP}}$. Consider the induction step. If $\xi' = \xi_1 \wedge \xi_2$ or $\xi' = \neg\xi_1$ and ξ_1, ξ_2 satisfy (†), it immediately follows that ξ' satisfies (†). Now suppose $\xi'(\bar{x}) = \exists y\,(R(t_1, \ldots, t_k) \wedge \varphi(\bar{x}, y))$ where R is the input or previous input relation, each variable among the t_i's is either y or in $\bar{x}$ (at least one t_i is y by input boundedness), and (†) holds for $\varphi(\bar{x}, y)$. If R is empty in ρ_i then $\xi'(\bar{x} \leftarrow \bar{e})$ is false in both ρ_i and $\rho_i^{\downarrow}$, so (†) holds. If R is not empty, then $\rho_i \models \xi'(\bar{x} \leftarrow \bar{e})$ iff there exists c occurring in R such that $\rho_i \models R(t_1, \ldots, t_k)[\bar{x} \leftarrow \bar{e}, y \leftarrow c] \wedge \varphi(\bar{x}, y)[\bar{x} \leftarrow \bar{e}, y \leftarrow c]$. By the induction hypothesis, this happens iff $\rho_i^{\downarrow} \models R(t_1, \ldots, t_k)[\bar{x} \leftarrow \bar{e}, y \leftarrow c] \wedge \varphi(\bar{x}, y)[\bar{x} \leftarrow \bar{e}, y \leftarrow c]$, so $\rho_i^{\downarrow} \models \xi'(\bar{x} \leftarrow \bar{e})$, which shows (†). □

Lemma 4.5 shows that in a configuration ρ_i, the information relevant to satisfaction of $\psi_{\bar{c}}$ is captured by $\rho_i^{\downarrow}$. In other words, a run satisfies $\psi_{\bar{c}}$ iff its local run satisfies $\psi_{\bar{c}}$. Let $C_k = C \cup \{c_1, \ldots, c_{2k}\}$ where $c_1, \ldots, c_{2k}$ are distinct new elements (recall that k is the arity of I). Let m be the number of existentially quantified variables in the input-options rule for I, and $e_1, \ldots, e_m$ be m distinct new elements. Let $C_{km} = C_k \cup \{e_1, \ldots, e_m\}$. It will be convenient to assume, without loss of generality, than $C_{km} \subset \mathbf{dom}_\infty$. We can represent the C_D-isomorphism type of $\rho_i^{\downarrow}$ by an instance whose domain is C_{km}, which we denote $\tau(\rho_i)$. Thus, Lemma 4.5 says that $\sigma(\rho_i) = \sigma(\rho_i^{\downarrow}) = \sigma(\tau(\rho_i))$. Note that the domain of $\tau(\rho_i)$ is the *fixed* set of elements C_{km}, whereas the domain of $\rho_i^{\downarrow}$ depends on i.

We wish to lift the above from individual configurations to entire runs. More precisely, we would like to be able to generate *sequences* $\{\tau_i\}_{i\geqslant 0}$ of instances using elements in C_{km} that correspond precisely to the sequences of C_D-isomorphism types of $\{\rho_i^{\downarrow}\}_{i\geqslant 0}$ for runs $\{\rho_i\}_{i\geqslant 0}$ of $\mathcal{A}$ on each database D. We formalize this using the notion of *pseudorun*.

Definition 4.6. Let $\mathcal{A} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathcal{R}\rangle$ be an input-bounded ASM$^+$ transducer, and let $\psi_{\bar{c}}$, C, C_k, and C_{km} be defined as above. A *C-pseudorun* of $\mathcal{A}$ is a sequence of instances $\{\langle S_i, I_i, P_i, A_i, D_i\rangle\}_{i\geqslant 0}$ with elements in C_{km} such that:

1. S_i, I_i, P_i, A_i, D_i are state, input, previous input, action, and database instances;
2. all D_i provide the same interpretation for the constants in C, and each constant in C is interpreted within C;
3. S_i and A_i are instances using only elements in C;
4. I_i and P_i contain at most one tuple each, using elements in C_k;
5. for each $i \geqslant 0$, $P_{i+1} = I_i$ and $D_i|_{C\cup dom(I_i)} = D_{i+1}|_{C\cup dom(P_{i+1})}$;
6. for each $i \geqslant 0$, if $I_i = \{\bar{a}\}$ and the input-options formula for I is $\exists\bar{x}\,\varphi_I(\bar{x})$, then $\langle D_i, P_i, S_i\rangle \models \varphi_I(\bar{x} \leftarrow \bar{a})$;
7. $S_0 = A_0 = P_0 = \emptyset$;
8. for each $i \geqslant 0$, $S_{i+1} = S'_{i+1}|_C$ and $A_{i+1} = A'_{i+1}|_C$, where S'_{i+1} and A'_{i+1} are the state and action relations defined from the configuration $\langle S_i, I_i, P_i, A_i\rangle$ of $\mathcal{A}$ and database D_i, according to the rules of $\mathcal{A}$.

[4] Our local run is an extension of the notion of local run introduced in [36,37].

Before proceeding, we make another useful observation showing that we can confine the search for runs of $\mathcal{A}$ satisfying $\psi_{\bar{c}}$ to *periodic* runs. Specifically, a run $\{\rho_i\}_{i\geqslant 0}$ is periodic iff there exist $n \geqslant 0$ and $p \geqslant 0$ such that $\rho_i = \rho_{i+p}$ for every $i \geqslant n$.

Lemma 4.7. *Let $\mathcal{A}$ be an input-bounded ASM$^+$ transducer, φ an input-bounded LTL-FO formula, and $\bar{c}, \psi_{\bar{c}}$ be as above. Let D be a database instance. If there exists a run of $\mathcal{A}$ on D satisfying $\psi_{\bar{c}}$ then there exists a periodic run of $\mathcal{A}$ on D satisfying $\psi_{\bar{c}}$.*

Proof. Consider a run $\rho = \{\rho_i\}_{i\geqslant 0}$ of $\mathcal{A}$ on D satisfying $\psi_{\bar{c}}$. Let $A_{\psi_{\bar{c}}}$ be the Büchi automaton corresponding to the propositional LTL formula $\psi_{\bar{c}}^{aux}$. Recall that for $i \geqslant 0$, $\sigma(\rho_i)$ denotes the truth assignment to the propositions in $\psi_{\bar{c}}^{aux}$ such that p_ξ is true iff $\rho_i \models \xi$, and $\sigma(\rho) = \{\sigma(\rho_i)\}_{i\geqslant 0}$. Since $\rho \models \psi_{\bar{c}}$, there exists a run $q_0, s_0, s_1, \ldots, s_i, \ldots$ of $A_{\psi_{\bar{c}}}$ on input $\sigma(\rho)$ that goes through some accepting state, say f, infinitely often. Since there are finitely many distinct instances ρ_i in ρ, there must exist $n < j$, such that $s_n = s_j = f$ and $\rho_n = \rho_j$. Let $p = j - n$. Consider the sequence $\rho' = \{\rho'_i\}_{i\geqslant 0}$ defined by $\rho'_m = \rho_m$ for $0 \leqslant m \leqslant j$ and $\rho'_m = \rho'_{m-p}$ for $m > j$. Clearly, ρ' is a periodic run of $\mathcal{A}$ on D and $A_{\psi_{\bar{c}}}$ accepts $\sigma(\rho')$, so $\rho' \models \psi_{\bar{c}}$. □

We next show the following key connection between pseudoruns and actual runs.

Lemma 4.8. *Let $\mathcal{A}$ be an input-bounded ASM$^+$ transducer and φ an input-bounded LTL-FO formula. Let C, $\bar{c}$, and $\psi_{\bar{c}}$ be as above. The following are equivalent*:

(i) *there exists some periodic run ρ of $\mathcal{A}$ on a database D such that $\rho \models \psi_{\bar{c}}$, and*
(ii) *there exists some periodic C-pseudorun τ of $\mathcal{A}$ such that $\tau \models \psi_{\bar{c}}$.*

Proof. Consider (i) → (ii). Let $\rho = \{\rho_i\}_{i\geqslant 0}$ be a periodic run of $\mathcal{A}$ on a database D, that satisfies $\psi_{\bar{c}}$. Recall that we assume without loss of generality that $C \subseteq \mathbf{dom}_\infty$. We can further assume that $C_D = dom(D) \cap C$ (otherwise we take an isomorphic image of D on which this is true). We first construct a C-pseudorun τ of $\mathcal{A}$ such that $\sigma(\rho) = \sigma(\tau)$. In particular, τ satisfies $\psi_{\bar{c}}$. From τ one can then easily construct a periodic C-pseudorun of $\mathcal{A}$ satisfying $\psi_{\bar{c}}$, as in Lemma 4.7.

Consider $\{\rho_i^\downarrow\}_{i\geqslant 0}$, where $\rho_i^\downarrow = \langle S_i^\rho, I_i^\rho, P_i^\rho, A_i^\rho, D_i^\rho\rangle$. We define by induction a sequence of one-to-one mappings $\{f_i\}_{i\geqslant 0}$, where f_i maps $dom(\rho_i^\downarrow)$ to C_{km} and is the identity on C_D:

- f_0 is an arbitrary one-to-one mapping from $dom(\rho_0^\downarrow)$ to C_{km} that fixes C_D and maps C_{IP}^0 to C_k;
- $f_{i+1}|dom(P_{i+1}) = f_i|dom(I_i)$ and f_{i+1} is an arbitrary extension of $f_{i+1}|dom(P_{i+1})$ to a one-to-one mapping from $dom(\rho_{i+1}^\downarrow)$ to C_{km} that is the identity on C_D and maps C_{IP}^{i+1} to C_k.

Now let $\tau_i = f_i(\rho_i^\downarrow)$ for each $i \geqslant 0$ (note that in particular the constants C are interpreted by τ_i as in D). By definition, τ_i and $\rho_i^\downarrow$ are C-isomorphic. It remains to show that $\{\tau_i\}_{i\geqslant 0}$ is a C-pseudorun of $\mathcal{A}$. Parts (1)–(6) of Definition 4.6 are obviously satisfied. Consider (7). Consider τ_i and τ_{i+1} for $i \geqslant 0$. Let R be a state relation and $R_i = S_i(R)$, $R_{i+1} = S_{i+1}(R)$. Suppose $\varphi_R^+(\bar{x})$ and $\varphi_R^-(\bar{x})$ are the input-bounded formulas of $\mathcal{A}$ defining the tuples to be inserted, respectively deleted from R. Let $\bar{e}$ be a sequence of elements in C of the same arity as $\bar{x}$. Since $\rho_i^\downarrow$ is C-isomorphic to τ_i and φ_R^+, φ_R^- are input bounded, one can show similarly to (†) in the proof of Lemma 4.5 that $\tau_i \models \varphi_R^+(\bar{e})$ iff $\rho_i^\downarrow \models \varphi_R^+(\bar{e})$, and also $\tau_i \models \varphi_R^-(\bar{e})$ iff $\rho_i^\downarrow \models \varphi_R^-(\bar{e})$. Also by (†), $\rho_i^\downarrow \models \varphi_R^+(\bar{e})$ iff $\rho_i \models \varphi_R^+(\bar{e})$ and $\rho_i^\downarrow \models \varphi_R^-(\bar{e})$ iff $\rho_i \models \varphi_R^-(\bar{e})$. It follows that $\bar{e}$ is inserted/deleted from R in the transition from ρ_i to ρ_{i+1} iff it is inserted/deleted in the transition from $\rho_i^\downarrow$ to $\rho_{i+1}^\downarrow$ iff it is inserted/deleted in the transition from τ_i to τ_{i+1} according to the state rule for R. Since by definition R is the same in ρ_i, $\rho_i^\downarrow$, and τ_i, and R is also the same in ρ_{i+1}, $\rho_{i+1}^\downarrow$, and τ_{i+1}, (7) holds for state relations. A similar argument shows that (7) also holds for action relations.

Now consider the harder (ii) → (i). Let $\tau = \{\tau_i\}_{i\geqslant 0}$ be a periodic C-pseudorun satisfying $\psi_{\bar{c}}$, where $\tau_i = \langle S_i^\tau, I_i^\tau, P_i^\tau, A_i^\tau, D_i^\tau\rangle$. We define a database D interpreting the constant symbols in C in the same way as τ, and

a periodic run $\rho = \{\langle S_i, I_i, P_i, A_i\rangle\}_{i\geqslant 0}$ of $\mathcal{A}$ on D such that for each i, $\rho_i^{\downarrow}$ is C-isomorphic to τ_i. In particular, $\sigma(\tau) = \sigma(\rho)$, so $\rho \models \psi_{\bar{c}}$.

Recall that τ is a sequence of instances using elements in C_{km} and $\mathbf{dom}_\infty$ is an infinite domain. To construct a database D and run ρ of $\mathcal{A}$ on D, we will assign values in $\mathbf{dom}_\infty$ to occurrences of the elements in C_{km} in the different configurations of τ. The challenge is to do so while using only finitely many values.

Consider the elements in $C_k - C$. Some of these elements occurring in different configurations of τ must be assigned the same value, while others are independent of each other. We denote by $\langle i, a\rangle$ the occurrence of a in τ_i, where $i \geqslant 0$ and $a \in C_k - C$. To capture the required equalities among elements in different configurations, we define the following equivalence relation $\equiv$ on occurrences $\langle i, a\rangle$. First, let $\langle i, a\rangle \approx \langle i+1, a\rangle$ iff a occurs in I_i (and therefore in P_{i+1}). Next, let $\equiv$ be the symmetric, reflexive, transitive closure of $\approx$. Let f be a mapping from the set of all occurrences of elements in C_{km} in τ to $\mathbf{dom}_\infty$ such that $f(\langle i, a\rangle) = f(\langle j, a\rangle)$ iff $\langle i, a\rangle \equiv \langle j, a\rangle$, and f is the identity on C. Note that the range of f is infinite. Consider the sequence $f(\tau) = \{f(\tau_i)\}_{i\geqslant 0}$, where $f(\tau_i) = \langle f(S_i^\tau), f(I_i^\tau), f(P_i^\tau), f(A_i^\tau), f(D_i^\tau)\rangle$. Let $D_f = \bigcup_{i\geqslant 0} f(D_i^\tau)$. We first show that $f(\tau)$ satisfies the definition of a local run of $\mathcal{A}$ on D_f, except for the requirement that D_f be finite. Given the definition of pseudorun, and since τ_i and $f(\tau_i)$ are C-isomorphic, it is enough to show that

$$D_f|dom\big(f(\tau_i)\big) = f\big(D_i^\tau\big) \quad \text{for all } i \geqslant 0. \tag{‡}$$

Consider $a \in range(f)$. The *span* of a is $\{i \mid \exists b\, f(\langle i, b\rangle) = a\}$. From the definition of f it follows that the span of each $a \notin C$ is an interval, possibly infinite to the right. Now consider (‡). Suppose towards a contradiction that $D_f|dom(f(\tau_i)) \neq f(D_i^\tau)$ for some $i \geqslant 0$. Since by definition $f(D_i^\tau) \subseteq D_f|dom(f(\tau_i))$, it follows that $f(D_i^\tau) \subset D_f|dom(f(\tau_i))$. Thus, for some database relation R, there exists a tuple t such that $dom(t) \subseteq dom(f(\tau_i))$, $R(t)$ holds in $D_f|dom(f(\tau_i))$ but $R(t)$ does not hold in $f(D_i^\tau)$. In particular, there must exist $j \neq i$ such that $R(t)$ holds in $f(D_j^\tau)$. Since $t \in dom(\tau_i) \cap dom(\tau_j)$, it follows that $i, j \in span(a)$ for each $a \in dom(t)$. However, from (4) in the definition of pseudoruns, it follows that $f(D_i^\tau)|_{dom(\tau_i)\cap dom(\tau_j)} = f(D_j^\tau)|_{dom(\tau_i)\cap dom(\tau_j)}$, so $R(t)$ holds in $f(D_i^\tau)$ iff $R(t)$ holds in $f(D_j^\tau)$. This is a contradiction. Thus, (‡) holds.

We next construct from $f(\tau)$ a local run, and then a run, whose universe is finite. Intuitively, this involves some "surgery" on $f(\tau)$, using a pumping argument. The main idea is the following. Note that f maps elements in each configuration of τ that are not in C to *new* elements in $\mathbf{dom}_\infty$, yielding the infinite universe of $f(\tau)$. It turns out that values can be assigned more economically: if two configurations τ_α and τ_β are isomorphic and far enough apart, the values assigned by f for τ_β can be reused for τ_α. Based on this observation, we can modify f so that its range has only finitely many values. We next formalize this argument.

We need to give special treatment to elements in $range(f)$ whose span is infinite. Since I contains only one tuple of arity k, it follows that at most k elements in $range(f) - C$ may have infinite span. Let R_k consist of all such elements (at most k). Let $N > 0$ be such that all elements in R_k occur in every $f(\tau_i)$ for $i \geqslant N$. From the periodicity of τ and the fact that all elements not in $R_k \cup C$ have finite span, it follows that there exist α, β, $N < \alpha < \beta$, where $\beta - \alpha$ is a sufficiently large multiple of the least period of τ, such that:

(a) $\tau_\alpha = \tau_\beta$ and $\tau_i = \tau_{i+p}$ for all $i \geqslant \alpha$, where $p = \beta - \alpha$,
(b) $f(\tau_\alpha)$ and $f(\tau_\beta)$ are $(R_k \cup C)$-isomorphic,
(c) there are no elements $a_\alpha, a_\beta, d \in range(f) - (R_k \cup C)$, such that $a_\alpha \in dom(f(\tau_\alpha))$, $a_\beta \in dom(f(\tau_\beta))$, and $span(d) \cap span(a_\alpha) \neq \emptyset$ and $span(d) \cap span(a_\beta) \neq \emptyset$.

Let h be an $(R_k \cup C)$-isomorphism from $f(\tau_\alpha)$ to $f(\tau_\beta)$. Consider the sequence of configurations $f(\tau_\alpha) \ldots f(\tau_{\beta-1})$. Let $\bar{\tau}_\alpha$ be the prefix of $f(\tau_\alpha) \ldots f(\tau_{\beta-1})$ consisting of all configurations in the sequence whose domain intersects $dom(f(\tau_\alpha)) - (R_k \cup C)$, and let $\bar{\tau}_{\beta-1}$ be the suffix of $f(\tau_\alpha) \ldots f(\tau_{\beta-1})$ consisting of all configurations in the sequence whose domain intersects $dom(f(\tau_\beta)) - (R_k \cup C)$. By (c), $\bar{\tau}_\alpha$ and $\bar{\tau}_{\beta-1}$ do not overlap, so $f(\tau_\alpha) \ldots f(\tau_{\beta-1}) = \bar{\tau}_\alpha \bar{\tau} \bar{\tau}_{\beta-1}$ for some sequence of configurations $\bar{\tau}$. Let $f_h(\bar{\tau}_\alpha)$ be obtained from $\bar{\tau}_\alpha$ by replacing each element $a \in dom(f(\tau_\alpha))$ by $h(a) \in dom(f(\tau_\beta))$. Intuitively, (c) guarantees that α and β are far enough apart that replacing a by $h(a)$ in $\bar{\tau}_\alpha$ as above creates no interference. Note that the sequence $f_h(\bar{\tau}_\alpha)\bar{\tau}\bar{\tau}_{\beta-1}$ starts with $f(\tau_\beta)$. Consider the periodic sequence $\rho^{\downarrow} = \{\rho_i^{\downarrow}\}_{i\geqslant 0}$ obtained by concatenating $f_h(\bar{\tau}_\alpha)\bar{\tau}\bar{\tau}_{\beta-1}$ infinitely many times to the right of $f(\tau_0), \ldots, f(\tau_{\beta-1})$. It is easily seen that τ_i and $\rho_i^{\downarrow}$ are C-isomorphic for all $i \geqslant 0$. In particular, $\sigma(\tau) = \sigma(\rho^{\downarrow})$, so $\rho^{\downarrow} \models \psi_{\bar{c}}$. It is enough to

show that $\rho^{\downarrow}$ is a local run of $\mathcal{A}$ for some finite database. Let $\rho^{\downarrow} = \{\langle S_i, I_i, P_i, A_i, D_i\rangle\}_{i\geqslant 0}$. Let $D = \bigcup_{i\geqslant 0} D_i$. From the periodicity of $\rho^{\downarrow}$ it follows that D is finite. We claim, similarly to (‡), that

$$D|dom\big(\rho_i^{\downarrow}\big) = D_i \quad \text{for each } i \geqslant 0. \tag{$*$}$$

Suppose towards a contradiction that there exist i, j, $i \neq j$, a database relation R, and a tuple t such that $dom(t) \subseteq dom(\rho_i^{\downarrow}) \cap dom(\rho_j^{\downarrow})$, $D_i \models R(t)$, and $D_j \not\models R(t)$. Because of (‡), $\rho_i^{\downarrow}$ and $\rho_j^{\downarrow}$ cannot both be configurations already appearing in $f(\tau)$. Thus, at least one of $\rho_i^{\downarrow}$ and $\rho_j^{\downarrow}$ are configurations in $f_h(\bar{\tau}_\alpha)$. There are three cases to consider:

1. $\rho_i^{\downarrow}$ and $\rho_j^{\downarrow}$ are both in $f_h(\bar{\tau}_\alpha)$. Due to (c) above, no $b \in range(h)$ occurs in $dom(f(D_i^\tau)) \cup dom(f(D_j^\tau))$. But then h can be extended to an isomorphism (by the identity) to the entire $dom(f(D_i^\tau)) \cup dom(f(D_j^\tau))$, and h^{-1} is also an isomorphism. Thus, $dom(h^{-1}(t)) \in dom(f(\tau_i)) \cap dom(f(\tau_j))$, $f(D_i^\tau) \models R(h^{-1}(t))$, and $f(D_j^\tau) \not\models R(h^{-1}(t))$. However, this is a contradiction with (‡).
2. $\rho_i^{\downarrow}$ occurs in $f_h(\bar{\tau}_\alpha)$ and $\rho_j^{\downarrow}$ does not. Thus, $dom(t) \subseteq dom(h(f(D_i^\tau))) \cap dom(f(D_j^\tau))$, $h(f(D_i^\tau)) \models R(t)$, and $f(D_j^\tau) \not\models R(t)$. Suppose $dom(t) \cap range(h) = \emptyset$. Then $dom(t) \subseteq dom(f(D_i^\tau))$. Thus, $dom(t) \subseteq dom(f(D_i^\tau)) \cap dom(f(D_j^\tau))$, $f(D_i^\tau) \models R(t)$, and $f(D_j^\tau) \not\models R(t)$. This contradicts (‡). Thus, $dom(t) \cap range(h) \neq \emptyset$. But then $dom(f(\tau_j)) \cap dom(f(\tau_\beta)) \neq \emptyset$. From (c) it then follows that $dom(t) \subseteq range(h)$, so $dom(t) \subseteq dom(f(\tau_\beta))$. By (‡), $f(D_\beta^\tau) \not\models R(t)$ since $dom(t) \subseteq dom(f(\tau_\beta)) \cap dom(f(\tau_j))$ and $f(D_j^\tau) \not\models R(t)$. Since $dom(t) \subseteq dom(f(\tau_\beta))$, $dom(h^{-1}(t)) \subseteq dom(f(\tau_\alpha))$, so $dom(h^{-1}(t)) \subseteq dom(\tau_\alpha) \cap dom(f(\tau_i))$. Again by (‡), $f(D_\alpha^\tau) \models R(h^{-1}(t))$ because $f(D_i^\tau) \models R(h^{-1}(t))$. Thus, $f(D_\alpha^\tau) \models R(h^{-1}(t))$ and $f(D_\beta^\tau) \not\models R(t)$. However, this contradicts the fact that h is an $(R_k \cup C)$-isomorphism from $f(\tau_\alpha)$ to $f(\tau_\beta)$.
3. $\rho_j^{\downarrow}$ occurs in $f_h(\bar{\tau}_\alpha)$ and $\rho_i^{\downarrow}$ does not. The proof is similar to (2) and is omitted.

Thus, $(*)$ is proven.

Finally, let $\rho = \{\langle S_i', I_i, P_i, A_i'\rangle\}_{i\geqslant 0}$ be obtained from $\rho^{\downarrow}$ by computing for each $i \geqslant 0$, S_{i+1}' and A_{i+1}' from $\langle S_i', I_i, P_i\rangle$ and D, using the state and action rules of $\mathcal{A}$. From $(*)$, the definition of pseudorun, and the construction of $\rho^{\downarrow}$, it is clear that ρ is a run of $\mathcal{A}$ on database D, and $\rho^{\downarrow}$ is the local run of ρ. In particular, $\sigma(\tau) = \sigma(\rho)$, so $\rho \models \psi_{\bar{c}}$. Also, ρ is periodic. This completes the proof. □

Lemma 4.8 says that in order to determine whether $\mathcal{A}$ satisfies $\psi_{\bar{c}}$, it is enough to focus on periodic C-pseudoruns of $\mathcal{A}$. Summarizing the above development, we can now describe a non-deterministic PSPACE verification algorithm for input-bounded ASM$^+$ transducers and input-bounded LTL-FO properties.

The input to the algorithm is an input-bounded ASM$^+$ transducer $\mathcal{A}$ and an input-bounded LTL-FO formula φ. Let $\exists\bar{x}\,\psi(\bar{x})$ be the negation of φ and let $\bar{c}$ be a sequence of constant symbols, one for each variable in $\bar{x}$. Let C consist of $\bar{c}$ together with all constant symbols used in the specification of $\mathcal{A}$ or in φ. Guess an interpretation of the constants in C by values in C. Let $\psi_{\bar{c}} = \psi[\bar{x} \leftarrow \bar{c}]$ and let $\psi_{\bar{c}}^{aux}$ be the propositional LTL formula obtained by replacing each FO component ξ of $\psi_{\bar{c}}$ by a propositional symbol p_ξ. Let $A_{\psi_{\bar{c}}}$ be the Büchi automaton corresponding to $\psi_{\bar{c}}^{aux}$. Let $C_{km} = C \cup \{c_1, \ldots, c_{2k}\} \cup \{e_1, \ldots, e_m\}$, where the c_i's and e_j's are distinct new elements. We use the following non-deterministic PSPACE algorithms:

- Büchi-Next: on input $(\psi_{\bar{c}}^{aux}, s, \sigma)$, where s is a state of $A_{\psi_{\bar{c}}}$ and σ is a truth assignment to the propositions in $\psi_{\bar{c}}^{aux}$, the algorithm[5] returns a state s' of $A_{\psi_{\bar{c}}}$ such that $\langle s, \sigma, s'\rangle$ is a transition in $A_{\psi_{\bar{c}}}$.
- Pseudorun-Next: given as input a configuration τ in a C-pseudorun of $\mathcal{A}$, output a possible next configuration τ' in the pseudorun.

The algorithm now proceeds as follows:

1. flag := 0;
2. set τ_0 to an initial configuration of a C-pseudorun of $\mathcal{A}$;
3. set s_0 to some output of Büchi-Next$(\psi_{\bar{c}}, q_0, \sigma(\tau_0))$, where q_0 is the start state of $A_{\psi_{\bar{c}}}$;

[5] As noted earlier, the existence of such a PSPACE algorithm is a classical result shown in [35].

4. set (s, τ) to (s_0, τ_0);
5. if flag $= 0$ and s is an accepting state of $A_{\psi_{\bar{c}}}$ then non-deterministically continue or set $(\bar{s}, \bar{\tau})$ to (s, τ) and set flag $:= 1$;
6. set τ to Pseudorun-Next(τ) and s to Büchi-Next$(\psi_{\bar{c}}, s, \sigma(\tau))$;
7. if flag $= 1$ and $(s, \tau) = (\bar{s}, \bar{\tau})$ then output YES and stop; otherwise, go to 5.

Clearly, the above non-deterministic PSPACE algorithm accepts iff there exists a periodic C-pseudorun of $\mathcal{A}$ accepted by $A_{\psi_{\bar{c}}}$. Observe that if the arity of relations in the schema of $\mathcal{A}$ is not bounded, the above algorithm is in EXSPACE. This establishes the following.

Theorem 4.9. *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 *for fixed arity schemas, and* EXPSPACE *otherwise.*

Theorem 4.9 in conjunction with Lemma 5.6 complete the proof of the main result of the section, Theorem 5.3. We note that the PSPACE algorithm described above provides the basis for a practical implementation of a verifier for Web applications. Such an implementation, including additional heuristics that improve the practical performance of the algorithm, is described in [15,17]. The implementation turns out to be surprisingly effective, with verification times of under one minute in a battery of experiments.

4.2. Boundaries of decidability

One may wonder whether the input-boundedness restriction can be relaxed without affecting decidability of verification. Unfortunately, even small relaxations can lead to undecidability. Specifically, we consider the following: (i) relaxing the requirement that state atoms be ground in formulas defining input options, by allowing state atoms with variables, (ii) relaxing the input-bounded restriction by allowing a very limited form of non-input-bounded quantification in the form of state projections, (iii) allowing $prev_I$ relations to record *all* previous inputs to I rather than just the preceding one, (iv) relaxing the input-bounded restriction on properties to express functional dependencies (FDs) on the database relations,[6] and (v) extending LTL-FO formulas with path quantification.

We begin with extension (i) and show undecidability even for a fixed LTL-FO formula and input options defined by quantifier-free FO formulas using just database and state relations.

Theorem 4.10. *There exists a fixed input-bounded LTL-FO formula φ for which it is undecidable, given an input-bounded ASM$^+$ transducer $\mathcal{A}$ with input options defined by quantifier-free FO formulas over database and state relations, whether $\mathcal{A} \models \varphi$.*

Proof. The proof is by reduction of the question of whether a Turing Machine (TM) M halts on input ϵ. Let M be a deterministic TM with a left-bounded, right-infinite tape. We construct from it an ASM$^+$ transducer $\mathcal{A}$ as follows. The idea is to represent configurations of M using a 4-ary state relation T. The first two coordinates of T represent a successor relation on a subset of the active domain of the database. A tuple $T(x, y, u, v)$ says that the content of the xth cell is u, the next cell is y, and v is a state p iff M is in state p and the head is on cell x. Otherwise, v is some special symbol #. The moves of M are simulated by modifying T accordingly. M halts on input ϵ iff there exists a run of $\mathcal{A}$ on some database such that some halting state h is reached. Thus, M does not accept ϵ iff for every run, $T(x, y, u, h)$ does not hold for any x, y, u, that is, $\mathcal{A} \models \forall x \forall y \forall u\, \mathbf{G}(\neg T(x, y, u, h))$.

We now outline the construction of $\mathcal{A}$ in more detail. The database schema of $\mathcal{A}$ consists of a unary relation D and a constant *min*. The state relations are the following:

- T, a 4-ary relation;

[6] Note that any FD can be expressed as a first-order sentence f, so checking that ASM$^+$ transducer $\mathcal{A}$ satisfies property φ provided that its database satisfies FD f reduces to the standard verification problem $\mathcal{A} \models f \rightarrow \varphi$. The property $f \rightarrow \varphi$ however is not input-bounded.

- *Cell*, *Max*, and *Head*, unary relations;
- propositional states used to control the computation: *initialized*, *simul*.

The input relations are I (unary) and H (4-ary).

The first phase of the simulation constructs the initial configuration of M on input ϵ, and the tape that the current run will make available for the computation. This phase makes use of the unary input relation I. Intuitively, the role of I is to pick a new value from the active domain, that has not yet been used to identify a cell, and use it to identify a new cell of the tape. The state relation *Cell* keeps track of the values previously chosen, to prevent them from being chosen again. The state relation *Max* keeps track of the most recently inserted value.

The rules implementing the initialization are the following (the symbol b denotes the blank symbol of M and q_0 is the start state):

$$
\begin{aligned}
\mathit{Options}_I(y) &\leftarrow D(y) \wedge y \neq \mathit{min} \wedge \neg \mathit{Cell}(y) \wedge \neg \mathit{simul},\\
T(\mathit{min}, y, b, q_0) &\leftarrow I(y) \wedge \neg \mathit{initialized},\\
\mathit{Cell}(\mathit{min}) &\leftarrow \neg \mathit{initialized},\\
\mathit{Head}(\mathit{min}) &\leftarrow \neg \mathit{initialized},\\
\mathit{initialized} &\leftarrow \neg \mathit{initialized},\\
T(x, y, b, \#) &\leftarrow I(y) \wedge \mathit{Max}(x),\\
\mathit{Cell}(y) &\leftarrow I(y),\\
\neg \mathit{Max}(x) &\leftarrow \mathit{Max}(x),\\
\mathit{Max}(y) &\leftarrow I(y),\\
\mathit{simul} &\leftarrow \forall y \neg I(y).
\end{aligned}
$$

The state *simul* signals the transition to the simulation phase. Notice that this happens either if the input options for I become empty (because we have used the entire active domain) or because the input is empty at any point. In the simulation phase, T is updated to reflect the consecutive moves of M. The simulation is aborted if T runs out of tape. We illustrate the simulation with an example move. Suppose M is in state p, the head is at cell x, the content of the cell is 0, and the move of M in this configuration consists of overwriting 0 with 1, changing states from p to q, and moving right. The rules simulating this move are the following:

$$
\begin{aligned}
\mathit{Options}_H(x, y, 0, p) &\leftarrow \mathit{simul} \wedge \mathit{Head}(x) \wedge T(x, y, 0, p),\\
\neg T(x, y, 0, p) &\leftarrow \mathit{simul} \wedge H(x, y, 0, p),\\
T(x, y, 1, \#) &\leftarrow \mathit{simul} \wedge H(x, y, 0, p),\\
\neg T(y, z, u, \#) &\leftarrow \mathit{simul} \wedge H(x, y, 0, p) \wedge T(y, z, u, \#),\\
T(y, z, u, q) &\leftarrow \mathit{simul} \wedge H(x, y, 0, p) \wedge T(y, z, u, \#),\\
\neg \mathit{Head}(x) &\leftarrow \mathit{simul} \wedge H(x, y, 0, p),\\
\mathit{Head}(y) &\leftarrow \mathit{simul} \wedge H(x, y, 0, p).
\end{aligned}
$$

Such rules are included for every move of M. It is easy to see that this correctly simulates the moves of M. Note that if the input H is empty, T does not change. Finally, if the head reaches the last value provided in T, the transducer goes into an infinite loop in which, again, T stays unchanged. Thus, $T(x, y, u, h)$ holds in some run iff the computation of M on ϵ is halting. Equivalently, M does not halt on ϵ iff $\mathcal{A}$ satisfies the formula $\varphi = \forall x \forall y \forall u\, \mathbf{G}(\neg T(x, y, u, h))$. □

We next consider extension (ii): we relax input-boundedness of rules by allowing projections of state relations. We call an ASM$^+$ transducer *input-bounded with state projections* if all its formulas are input-bounded, excepting state rules that allow insertions of the form:

$$S(\bar{x}) \leftarrow \exists \bar{y}\, S'(\bar{x}, \bar{y})$$

where S and S' are state relations. We can show the following.

Theorem 4.11. *It is undecidable, given an input-bounded ASM$^+$ transducer $\mathcal{A}$ with state projections and input-bounded LTL-FO sentence φ, whether $\mathcal{A} \models \varphi$.*

Proof. The proof is by reduction of the implication problem for functional and inclusion dependencies, known to be undecidable [10]. Recall that a functional dependency (FD) over relation schema S of arity k is an expression $X \rightarrow Y$

where $X, Y \subseteq \{1, \ldots, k\}$. An instance Z over S satisfies an FD $X \to Y$ iff whenever two tuples in Z agree on X they also agree on Y. An inclusion dependency (ID) over S is an expression $[X] \subseteq [Y]$ where $X, B \subseteq \{1, \ldots, k\}$ and X and Y have the same size. An instance Z over S satisfies $[X] \subseteq [Y]$ iff for each tuple u in Z there exists a tuple v in Z such that $u|X = v|Y$. The implication problem for FDs and IDs is to determine, given a set Δ of FDs and IDs over S, and f an FD over S, whether Δ implies f (i.e., whether every instance over S satisfying Δ also satisfies f).

Let Δ be a set of FDs and IDs over a relation S, and f an FD over the same relation. We can assume without loss of generality that all FDs have singletons on the right-hand side, and we denote for simplicity $X \to \{A\}$ by $X \to A$. We construct an input-bounded ASM$^+$ transducer $\mathcal{A}$ with state projections and an input-bounded LTL-FO sentence φ such that $\Delta \models f$ iff $\mathcal{A} \models \varphi$.

Let $\mathcal{A} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathcal{R} \rangle$ where $\mathbf{D} = \{R\}$, $\mathbf{A} = \emptyset$, $\mathbf{I} = \{I, \mathit{done}\}$ where I has the same arity as S and *done* is propositional, and $\mathbf{S}$ consists of the following relations:

- the relation S;
- two propositions stop_1, stop_2;
- for each ID σ of the form $[X] \subseteq [Y]$ in Δ, a relation S_X of arity $|X|$, a relation S_Y of arity $|Y|$, a relation S^σ_X of arity $|X|$, and a proposition viol_σ;
- for each FD σ of the form $X \to A$ in $\Delta \cup \{f\}$ a relation S_{XA} of arity $|XA|$, a relation $S^\sigma_{XA_1A_2}$ of arity $|XA| + 1$, and a proposition viol_σ.

Next, let $\mathcal{R}$ be defined as follows. The input option rule for I defines the cross-product of the active domain given by the database relation R. The state rules consist of the following:

$$\begin{aligned} S(\bar{x}) &\leftarrow I(\bar{x}) \wedge \neg \mathit{stop}_1, \\ \mathit{stop}_1 &\leftarrow \mathit{done}, \\ \mathit{stop}_2 &\leftarrow \mathit{stop}_1 \end{aligned}$$

for each ID σ of the form $[X] \subseteq [Y]$ in Δ, the following rules (where $\pi_X(S)$ denotes the projection of S on X):

$$\begin{aligned} S_X &\leftarrow \pi_X(S), \\ S_Y &\leftarrow \pi_Y(S), \\ S^\sigma_X(\bar{x}) &\leftarrow S_X(\bar{x}) \wedge \neg S_Y(\bar{x}) \wedge \mathit{stop}_2, \\ \mathit{viol}_\sigma &\leftarrow \exists \bar{x}\, S^\sigma_X(\bar{x}) \end{aligned}$$

for each FD σ of the form $X \to A$ in $\Delta \cup \{f\}$, the rules:

$$\begin{aligned} S_{XA} &\leftarrow \pi_{XA}(S), \\ S^\sigma_{XA_1A_2}(\bar{x}, a_1, a_2) &\leftarrow S_{XA}(\bar{x}a_1) \wedge S_{XA}(\bar{x}a_2) \wedge a_1 \neq a_2 \wedge \mathit{stop}_2, \\ \mathit{viol}_\sigma &\leftarrow \exists \bar{x} \exists a_1 \exists a_2\, S^\sigma_{XA_1A_2}(\bar{x}, a_1, a_2). \end{aligned}$$

Intuitively, the state relation S is populated by repeated inputs, until *done* is set to true, which is remembered in the state propositions stop_1 and stop_2 (stop_2 is needed for timing reasons, to ensure that violations are not tested too early). The rules check for violations of the dependencies in Δ, so that viol_σ is set to true iff S violates σ.

Note that all rules are input bounded, except those consisting of projections of state relations. Next, let ξ be the input-bounded LTL-FO sentence

$$G(\neg \mathit{done}) \vee \left[F(\mathit{done}) \wedge \left(F\left(\bigvee_{\sigma \in \Delta} \mathit{viol}_\sigma \right) \vee G(\psi_f) \right) \right]$$

where ψ_f is the formula $\neg S^f_{XA_1A_2}(\bar{x}, a_1, a_2)$ whose universal closure states that the FD $f = X \to A$ is satisfied. Finally, let φ be the universal closure of ξ. Intuitively, φ states that either *done* is never set to true, or it is set to true and at least one of the constraints of Δ is violated, or f is satisfied. Thus, $\mathcal{A} \models \varphi$ iff $\Delta \models f$. □

We now deal with extension (iii). We say that an ASM$^+$ transducer has *lossless input* if the prev_I relations record *all* previous inputs to I in the current run.

Theorem 4.12. *It is undecidable, given an input-bounded ASM$^+$ transducer $\mathcal{A}$ with lossless input and an input-bounded LTL-FO formula φ, whether $\mathcal{A} \models \varphi$.*

The proof uses a straightforward reduction of the undecidability of finite validity of FO formulas and is omitted.

We show the undecidability of extension (iv) next, proving that in the presence of FDs the verification problem becomes undecidable even for strictly input-bounded specifications and properties. Given property φ and transducer $\mathcal{A}$ with set $\mathcal{F}$ of functional dependencies on its database schema, we say that $\mathcal{A}$ satisfies φ under $\mathcal{F}$, denoted $\mathcal{A} \models_{\mathcal{F}} \varphi$, iff for every database D which satisfies $\mathcal{F}$, all runs of $\mathcal{A}$ over D satisfy φ.

Theorem 4.13. *It is undecidable, given an input-bounded ASM$^+$ transducer $\mathcal{A}$ with functional dependencies $\mathcal{F}$ on its database schema, and an input-bounded LTL-FO formula φ, whether $\mathcal{A} \models_{\mathcal{F}} \varphi$.*

Proof. By reduction from the *Post Correspondence Problem* (*PCP*). Consider a PCP instance, i.e. two sequences of length n: $\{u_i\}_{1 \leqslant i \leqslant n}, \{v_i\}_{1 \leqslant i \leqslant n}$, where all u_i, v_j are non-empty words over the alphabet $\{0, 1\}$. A *solution* to $\mathcal{P}$ is a finite non-empty sequence $\sigma \in [1, \ldots, n]^*$ such that the two strings obtained by concatenating $u_{\sigma(1)} u_{\sigma(2)} \ldots u_{\sigma(k)}$ and $v_{\sigma(1)} v_{\sigma(2)} \ldots v_{\sigma(k)}$ are identical ($\sigma(i)$ is the element at position i in σ). We say that these strings are *generated* by the solution σ. We construct ASM$^+$ transducer $\mathcal{A}$, set $\mathcal{F}$ of FDs, and property φ such that $\mathcal{P}$ has a solution iff $\mathcal{A} \not\models_{\mathcal{F}} \varphi$.

$\mathcal{A}$ simulates the search for a PCP solution as follows. The database encodes a finite string θ intended to correspond to the string generated by a solution of $\mathcal{P}$. $\mathcal{A}$ non-deterministically picks a sequence of indexes from $[1, \ldots, n]$ (by repeatedly asking an external user to pick an input among the options $[1, \ldots, n]$). Upon receiving the index i, $\mathcal{A}$ tries to match the corresponding words u_i and v_i in parallel against θ, by maintaining two cursors U and V on θ, as well as a cursor on u_i and a cursor on v_i. The cursors advance in lock-step, being incremented only if they point to the same character. Initially, U and V start from the first position in θ. The property φ is satisfied only if for all j, upon finishing to fully match u_j and v_j, U and V never meet on θ. It is easy to see that, if the database encodes a string θ, a run of $\mathcal{A}$ violates φ if and only if the sequence of indexes picked by the user is a solution to $\mathcal{P}$, which generates a prefix of θ.

θ is encoded using two binary database relations, $\underline{\mathsf{chain}}(s, t)$ (intended to contain as a subgraph a chain of directed $s \to t$ edges) and $\underline{\mathsf{char}}(i, c)$ (intended to label each node i in the chain with a character $c \in \{0, 1\}$). We pick $\mathcal{F}$ to enforce that $\underline{\mathsf{chain}}(s, t)$ satisfies the functional dependencies (FDs) $s \to t$ and $t \to s$ and $\underline{\mathsf{char}}$ satisfies the FD $i \to c$. The FDs on $\underline{\mathsf{chain}}$ ensure that nodes have in-degree and out-degree one, so $\underline{\mathsf{chain}}$ is a union of disjoint cycles and chains. The FD on $\underline{\mathsf{char}}$ will ensure that indexes are labeled uniquely, and the rules ensure that the labels are in $\{0, 1\}$ (the fact that 0 and 1 are distinct constants is stated in the property). To ensure that the cursors U and V progress along the same path without revisiting any node, we enforce that they start from the same position, a special node '$,' and never return to '$.'

In detail, the schema of $\mathcal{A}$ consists of

- $\mathbf{D} = \{\underline{\mathsf{chain}}(s, t),\ \underline{\mathsf{char}}(i, c),\ \text{'\$,' } 0, 1\}$ as described above ('$,' 0, and 1 are constants).
- $\mathbf{I} = \{\mathsf{I}(i), \mathsf{U}(x), \mathsf{V}(x)\}$. Intuitively, the user provides his pick of a word index in I, and U and V are the cursors on θ. The options provided to the user contain the immediate successors in $\underline{\mathsf{chain}}$ of the cursors at the previous input $prev_{\mathsf{U}}, prev_{\mathsf{V}}$. Of course, there is at most one successor due to the FDs on $\underline{\mathsf{chain}}$.
- $\mathbf{S}$ contains the following propositional states:
  - for each $1 \leqslant i \leqslant n$, each $1 \leqslant j \leqslant |u_i|$ and each $1 \leqslant k \leqslant |v_i|$, state U_i^j and state V_i^k (these play the role of cursors in the u_i and v_i words);
  - state done_u, set to true only when a full u_i word is matched; begun_u which, when set to false, signals that the matching of u_i words has not yet begun; similarly, states done_v and begun_v.
- $\mathbf{A} = \emptyset$.

$\mathcal{A}$ contains

- the input rules

$$Options_{\mathsf{I}}(i) \leftarrow (i = 1 \vee i = 2 \vee \cdots \vee i = n) \wedge (\neg \mathrm{begun}_u \wedge \neg \mathrm{begun}_v \vee \mathrm{done}_u \wedge \mathrm{done}_v);$$

$$Options_{\mathsf{U}}(t) \leftarrow (\neg \mathrm{begun}_u \wedge t = \text{'\$'}) \vee \mathrm{begun}_u \wedge \neg \mathrm{done}_u \wedge \exists s \exists c\, prev_{\mathsf{U}}(s) \wedge \underline{\mathsf{chain}}(s, t) \wedge t \neq \text{'\$'} \wedge \underline{\mathsf{char}}(t, c) \wedge \left(\bigvee_{i,j} prev_{\mathsf{I}}(i) \wedge c = u_i(j) \wedge \mathrm{U}_i^j \right);$$

$$Options_{\mathsf{V}}(t) \leftarrow (\neg \text{begun}_v \wedge t = \text{`\$'}) \vee \text{begun}_v \wedge \neg \text{done}_v \wedge \exists s \exists c\, prev_{\mathsf{V}}(s) \wedge \underline{\mathsf{chain}}(s,t) \wedge t \neq \text{`\$'} \wedge \underline{\mathsf{char}}(t,c) \wedge \left(\bigvee_{i,k} prev_{\mathsf{I}}(i) \wedge c = v_i(k) \wedge \mathrm{V}_i^k \right);$$

- the state rules

$$\begin{aligned}
\text{begun}_u &\leftarrow \neg \text{begun}_u \wedge \exists t\ \mathsf{U}(t),\\
\text{begun}_v &\leftarrow \neg \text{begun}_v \wedge \exists t\ \mathsf{V}(t),\\
\text{done}_u &\leftarrow \exists t\ \mathsf{U}(t) \wedge \left(\bigvee_{i=1}^{n} \mathrm{U}_i^{|u_i|-1} \right),\\
\neg \text{done}_u &\leftarrow \text{done}_u \wedge \exists x\ \mathsf{I}(x),\\
\text{done}_v &\leftarrow \exists t\ \mathsf{V}(t) \wedge \left(\bigvee_{i=1}^{n} \mathrm{V}_i^{|v_i|-1} \right),\\
\neg \text{done}_v &\leftarrow \text{done}_v \wedge \exists x\ \mathsf{I}(x).
\end{aligned}$$

  Moreover, for $1 \leqslant i \leqslant n$,

$$\begin{aligned}
\mathrm{U}_i^1 &\leftarrow \mathsf{I}(i),\\
\mathrm{U}_i^j &\leftarrow \mathrm{U}_i^{j-1} \wedge \exists t\ \mathsf{U}(t) \quad \text{for } 1 < j \leqslant |u_i|,\\
\neg \mathrm{U}_i^j &\leftarrow \mathrm{U}_i^j \wedge \exists t\ \mathsf{U}(t) \quad \text{for } 1 \leqslant j \leqslant |u_i|,\\
\mathrm{V}_i^1 &\leftarrow \mathsf{I}(i),\\
\mathrm{V}_i^j &\leftarrow \mathrm{V}_i^{j-1} \wedge \exists t\ \mathsf{V}(t) \quad \text{for } 1 < j \leqslant |v_i|,\\
\neg \mathrm{V}_i^j &\leftarrow \mathrm{V}_i^j \wedge \exists t\ \mathsf{V}(t) \quad \text{for } 1 \leqslant j \leqslant |v_i|.
\end{aligned}$$

$\mathcal{F}$ consists of FDs $t \to s$, $s \to t$ on $\underline{\mathsf{chain}}$ and $i \to c$ on $\underline{\mathsf{char}}$.

The (input-bounded) property φ is

$$\forall t\, 0 \neq 1 \wedge \mathbf{G} \neg \big(prev_{\mathsf{U}}(t) \wedge prev_{\mathsf{V}}(t) \wedge \text{done}_u \wedge \text{done}_v \big). \qquad \square$$

Finally, we address the undecidability of extension (v).

Theorem 4.14. *It is undecidable, given an input-bounded ASM$^+$ transducer $\mathcal{A}$ and input-bounded CTL-FO sentence φ, whether $\mathcal{A} \models \varphi$.*

Proof. Using path quantifiers, one can easily simulate first-order quantification by considering runs that provide values for the quantified variables as inputs. This allows to use a reduction of finite validity of FO sentences to the above verification problem. We illustrate the reduction for FO sentences of the form $\exists x \forall y\, \alpha(x, y)$ where α is a quantifier free formula over relational vocabulary $\{R\}$. Let $\mathcal{A} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathcal{R} \rangle$ be an ASM$^+$ transducer where $\mathbf{D} = \{R\}$, $\mathbf{I} = \{X, Y\}$ (X, Y are unary relations), $\mathbf{A} = \emptyset$, $\mathbf{S} = \{S_X, S_Y, done_x, true_\alpha\}$ (S_X, S_Y are unary and the other states are propositional). The input option rules are:

$$\begin{aligned}
Options_X(x) &\leftarrow \big(\psi_{dom}(x) \wedge \neg done_x\big) \vee \big(done_x \wedge S_X(x)\big),\\
Options_Y(y) &\leftarrow done_x \wedge \psi_{dom}(y),
\end{aligned}$$

where $\psi_{dom}(x)$ defines the active domain provided by R. The state rules are the following:

$$\begin{aligned}
S_X(x) &\leftarrow X(x),\\
done_x &\leftarrow \neg done_x,\\
true_\alpha &\leftarrow \exists x \exists y \big(X(x) \wedge Y(y) \wedge \alpha(x, y)\big).
\end{aligned}$$

Note that a path in $\mathcal{T}_\mathcal{A}$ starts at *root*, then proceeds to the start configuration of a run on some database D. The first input provided is a value of x, which is remembered in the state relation S_X. In the next configuration, $done_x$ is true, the same value of x as previously chosen is provided again via input X, and an arbitrary value is provided for y by the input relation Y. In the following configuration $true_\alpha$ is true if $\alpha(x, y)$ is satisfied for the chosen values of x, y.

Let φ be the CTL-FO sentence EXAXAX($true_\alpha$). Clearly, $\mathcal{A} \models \varphi$ iff $\exists x \forall y\, \alpha(x, y)$ is valid. Note that $\mathcal{A}$ and φ are input bounded (in fact φ is propositional, so in CTL). This proof is easily extended along the same lines to the general case. □

The proof further shows that a single alternation of path quantifiers is sufficient to yield undecidability, since one alternation is enough to express validity of FO sentences in the prefix class $\exists^*\forall^*$FO, known to be undecidable [6].

4.3. Verification of branching-time properties

In this section we consider the verification of branching-time temporal properties of ASM$^+$ transducers. As noted in the previous section, the decidability results for input-bounded ASM$^+$ transducers do not extend to CTL$^{(*)}$-FO sentences, even if they are restricted to be input bounded (by requiring every FO subformula to be input bounded). We next consider several restrictions leading to decidability of the verification problem for CTL$^{(*)}$-FO sentences.

4.3.1. Propositional input-bounded ASM$^+$ transducers

The first restriction further limits input-bounded ASM$^+$ transducers by requiring all states to be propositional. Furthermore, no rules can use $\mathbf{Prev_I}$ atoms. We call such ASM$^+$ transducers *propositional*. In a propositional ASM$^+$ transducer, inputs can still be parameterized. The CTL* formulas we consider are propositional and use only state symbols. For a given ASM$^+$ transducer $\mathcal{A} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathcal{R} \rangle$, we denote by $\Sigma_\mathcal{A}$ the propositional vocabulary $\mathbf{S}$. We first show the following:

Theorem 4.15. *Given a propositional, input-bounded ASM$^+$ transducer $\mathcal{A}$ and a CTL* formula φ over $\Sigma_\mathcal{A}$, it is decidable whether $\mathcal{A} \models \varphi$. The complexity of the decision procedure is* CO-NEXPTIME *if φ is in CTL, and* EXPSPACE *if φ is in CTL*.*

Proof. The proof has two stages. First, we show that there is a bound on the size of databases that need to be considered when checking for violations of φ (or equivalently, satisfaction of $\neg\varphi$). Second, we prove that for a given database D there exists a Kripke structure $K_{\mathcal{A},D}$ over alphabet $\Sigma_\mathcal{A}$, of size exponential in $\Sigma_\mathcal{A}$, such that $\mathcal{T}_{\mathcal{A},D} \models \neg\varphi$ iff $K_{\mathcal{A},D} \models \neg\varphi$. This allows us to use known model-checking techniques for CTL$^{(*)}$ on Kripke structures to verify whether $\mathcal{T}_{\mathcal{A},D} \models \neg\varphi$.

We start with the following:

Lemma 4.16. *Let $\mathcal{A}$ be a propositional, input bounded ASM$^+$ transducer, and φ a CTL* formula over $\Sigma_\mathcal{A}$. Then $\mathcal{A} \not\models \varphi$ iff there exists a database instance D of size exponential in $\mathcal{A}$, such that $\mathcal{T}_{\mathcal{A},D} \models \neg\varphi$.*

Proof. Let $\mathcal{A} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathcal{R} \rangle$ be a propositional, input-bounded ASM$^+$ transducer and φ a CTL* formula over $\Sigma_\mathcal{A}$. For each configuration ρ of $\mathcal{A}$, we denote by $\lambda(\rho)$ the set of states true in ρ. We also denote by λ the extension of this mapping to trees of configurations $\mathcal{T}_{\mathcal{A},D}$, where $\lambda(root) = \emptyset$. Obviously, if $\lambda(\mathcal{T}_{\mathcal{A},D_1}) = \lambda(\mathcal{T}_{\mathcal{A},D_2})$ then $\mathcal{T}_{\mathcal{A},D_1}$ and $\mathcal{T}_{\mathcal{A},D_2}$ satisfy the same CTL* formulas over $\Sigma_\mathcal{A}$. Now suppose that $\mathcal{A} \not\models \varphi$, so there is some D such that $\mathcal{T}_{\mathcal{A},D} \models \neg\varphi$. We show that there exists a database D_0 of size exponential in $\mathcal{A}$, such that $\lambda(\mathcal{T}_{\mathcal{A},D}) = \lambda(\mathcal{T}_{\mathcal{A},D_0})$, so $\mathcal{T}_{\mathcal{A},D_0} \models \neg\varphi$. This is done by showing that $\lambda(\mathcal{T}_{\mathcal{A},D}) = \lambda(\mathcal{T}_{\mathcal{A},D_0})$ iff D_0 satisfies a particular FO sentence ξ in the prefix class $\exists^*\forall^*$FO with a number of variables exponential in $\mathcal{A}$. Since ξ is satisfied by D, it is satisfiable. But this implies that ξ has a model D_0 whose domain has a number of elements equal to the number of existential variables of ξ, so exponential in $\mathcal{A}$ (see [6]). Thus, the size of D_0 is also exponential in $\mathcal{A}$ (for bounded database schema arity).

We next describe ξ. Note that, because states are propositional, the sets of propositions true in successors of a configuration ρ of a run of $\mathcal{A}$ on D depend only on D and $\lambda(\rho)$. Thus, $\lambda(\rho)$ uniquely determines the set $\{\lambda(\bar\rho) \mid \langle \rho, \bar\rho \rangle \in \mathcal{T}_{\mathcal{A},D}\}$. Consider a pair $\langle \Sigma, \bar\Sigma \rangle = \langle \lambda(\rho), \lambda(\bar\rho) \rangle \in \lambda(\mathcal{T}_{\mathcal{A},D})$. Let $I_1, \ldots, I_k$ be the input predicates in $\mathbf{I}$, and let $\varphi_1 \ldots, \varphi_k$ be the $\exists^*$FO formulas defining the input options for $I_1, \ldots, I_k$. We construct a quantifier-free FO sentence $\varphi_{\langle \Sigma, \bar\Sigma \rangle}(\bar x_1, \ldots, \bar x_k)$ on $\mathbf{D}$ such that $\lambda(\bar\rho) = \bar\Sigma$ whenever $\bar\rho$ is the next configuration from ρ resulting from the choice of inputs $\bar x_1, \ldots, \bar x_k$ from the options available for $I_1, \ldots, I_k$. For simplicity, we show the construction for the case when all user inputs are non-empty. The construction can be easily adapted to account for empty inputs.

Thus, $\langle \Sigma, \bar{\Sigma} \rangle \in \lambda(\mathcal{T}_{\mathcal{A},D_0})$ iff $D_0 \models \exists \bar{x}_1 \ldots \exists \bar{x}_k [\varphi_1(\bar{x}_1) \wedge \cdots \wedge \varphi_k(\bar{x}_k) \wedge \varphi_{\langle \Sigma, \bar{\Sigma} \rangle}(\bar{x}_1, \ldots, \bar{x}_k)]$. To ensure that only valid pairs $\langle \Sigma, \bar{\Sigma} \rangle$ occur in $\lambda(\mathcal{T}_{\mathcal{A},D_0})$, it must also be the case that $D_0 \models \forall \bar{x}_1, \ldots, \forall \bar{x}_k [(\varphi_1(\bar{x}_1) \wedge \cdots \wedge \varphi_k(\bar{x}_k)) \rightarrow \bigvee_{\bar{\Sigma}} \varphi_{\langle \Sigma, \bar{\Sigma} \rangle}(\bar{x}_1, \ldots, \bar{x}_k)]$.

Then ξ is the conjunction of all such formulas for all pairs in $\lambda(\mathcal{T}_{\mathcal{A},D})$, yielding a formula in the prefix class $\exists^*\forall^*$FO. Since there can be exponentially many such pairs, ξ is exponential in $\mathcal{A}$.

In order to define the sentence $\varphi_{\langle \Sigma, \bar{\Sigma} \rangle}$ we need the following notation. For each FO sentence ψ let ψ_Σ be the sentence obtained by replacing in ψ every proposition $p \in \Sigma$ by *true* and $p \notin \Sigma$ by *false*. Further, for each input-bounded formula ψ let the quantifier-free version of ψ , denoted ψ^{qf}, be defined as follows. Intuitively, ψ^{qf} eliminates the quantifiers by taking advantage of the fact that each input I consists, after the user's choice, of at most a single tuple $\bar{x}_I$ ($\bar{x}_I$ is a sequence of m distinct variables, where m is the arity of I). The formula ψ^{qf} reformulates ψ using these tuples. Specifically, let ψ' be obtained by replacing each input-bounded quantification $\exists \bar{x}\, (\alpha \wedge \beta)$ and $\forall \bar{x}\, (\alpha \rightarrow \beta)$ by $\alpha \wedge \beta$.

Next, let ψ^{qf} be obtained by first bringing ψ' to DNF (disjunctions of conjunctions), then applying to each disjunct δ the following procedure yielding δ'. Let *eq* (*neq*) be the (in)equalities occurring in δ. For each input relation I occurring in δ and each i, $1 \leqslant i \leqslant m$, let $\theta(I, i)$ be the set of terms occurring in the ith position of I in a positive occurrence $I(\bar{z})$ in δ. Let $\equiv$ be the reflexive, transitive closure of the following relation on the terms of δ: $\{(x, y) \mid x = y \in eq\} \cup \{(x, y) \mid x, y \in \theta(I, i)$ for some I and $i\}$. If for some x, y it is the case that $x \equiv y$ and $x \neq y$ is in *neq*, then $\delta' = \mathit{false}$. Otherwise, define the following equivalence relation on the pairs (I, i) of input atoms I and positions i of I: $(I, i) \equiv (J, j)$ iff there exist terms x, y so that $x \equiv y$, $x \in \theta(I, i)$, and $y \in \theta(J, j)$. For each variable y in δ, let $\nu(y)$ be one arbitrarily chosen $(x_I)_i$ for which $y \in \theta(I, i)$. Let δ' be obtained as follows:

1. add to δ the conjunction of all equalities $(x_I)_i = (y_J)_j$ where $(I, i) \equiv (J, j)$, $c = c'$ where c, c' are constants and $c \equiv c'$, and $(x_I)_i = c$ for some arbitrarily chosen $c \in \theta(I, i)$, if such exists;
2. for each negative occurrence $\neg I(z_1, \ldots, z_m)$ of an input atom, add the conjunct consisting of the disjunction $\bigvee_{i=1}^{m} (\nu(z_i) \neq (x_I)_i)$;
3. delete all input atoms;
4. replace each variable y by $\nu(y)$ in the remaining atoms.

Finally, ψ^{qf} is the disjunction of all resulting δ'.

We can now define $\varphi_{\langle \Sigma, \bar{\Sigma} \rangle}$. This is constructed using the rules of $\mathcal{A}$. Consider a proposition p in $\mathbf{S}$. We associate to p and $\neg p$ formulas β_p and $\beta_{\neg p}$ defined using the rules for $(\neg)p$. If $p \leftarrow \gamma$ and $\neg p \leftarrow \delta$ are in $\mathcal{R}$ then β_p is τ_Σ^{qf}, where $\tau = (\gamma \wedge \neg\delta) \vee (p \wedge \neg\delta)$, and $\beta_{\neg p}$ is π_Σ^{qf} where $\pi = (\neg p \wedge \neg\gamma) \vee (\neg p \wedge \gamma \wedge \delta) \vee (\delta \wedge \neg\gamma)$. Finally, $\varphi_{\langle \Sigma, \bar{\Sigma} \rangle}$ is the quantifier-free formula $\bigwedge_{p \in \bar{\Sigma}} \beta_p \wedge \bigwedge_{p \in (\Sigma_V - \bar{\Sigma})} \beta_{\neg p}$. □

The next stage towards the proof of Theorem 4.15 is to reduce the verification problem for a fixed database to a model checking problem of a CTL$^{(*)}$ formula on a Kripke structure. We therefore show the following.

Lemma 4.17. *For each ASM$^+$ transducer $\mathcal{A}$ over database schema* **D***, each database instance D over* **D***, and each CTL$^{(*)}$ formula φ over $\Sigma_\mathcal{A}$, one can construct, in time polynomial in D and exponential in $\mathcal{A}$, a Kripke structure $K_{\mathcal{A},D}$ over $\Sigma_\mathcal{A}$, of size exponential in $\Sigma_\mathcal{A}$, such that $\mathcal{T}_{\mathcal{A},D} \models \varphi$ iff $K_{\mathcal{A},D} \models \varphi$.*

Proof. The Kripke structure $K_{\mathcal{A},D}$ has one node labeled for each set of propositions $\Sigma \subseteq \Sigma_\mathcal{A}$ labeling a node in $\lambda(\mathcal{T}_{\mathcal{A},D})$. There is an edge $\langle \Sigma, \bar{\Sigma} \rangle$ iff there is a node labeled Σ with a child labeled $\bar{\Sigma}$ in $\lambda(\mathcal{T}_{\mathcal{A},D})$. Clearly, $K_{\mathcal{A},D}$ can be obtained by expanding $\lambda(\mathcal{T}_{\mathcal{A},D})$ until no new labels are found. Each edge involves evaluating the formulas of $\mathcal{A}$ on Σ and D, which is polynomial in Σ and D and exponential in $\mathcal{A}$. The maximum number of edges is exponential in $\Sigma_\mathcal{A}$. □

Lemmas 4.16 and 4.17 provide the proof of Theorem 4.15: to check that $\mathcal{A} \not\models \varphi$, first guess a database D of size exponential in $\mathcal{A}$, then construct from D and $\mathcal{A}$, in time exponential in $\mathcal{A}$, the Kripke structure $K_{\mathcal{A},D}$. Finally, checking that $K_{\mathcal{A},D} \models \neg\varphi$ is in polynomial time with respect to $K_{\mathcal{A},D}$ and $\neg\varphi$ if φ is in CTL, and in polynomial space if φ is in CTL*. Overall, checking $\mathcal{A} \models \varphi$ is in CO-NEXPTIME if φ is in CTL, and in EXPSPACE if φ is in CTL*. □

A special case of interest involves ASM$^+$ transducers that are entirely propositional. Thus, the database plays no role in the specification: inputs, states, and actions are all propositional, and the rules do not use the database. Let us call such a transducer *fully propositional*. We can show the following:

Theorem 4.18. *Given a fully propositional ASM$^+$ transducer $\mathcal{A}$ and a CTL* formula φ over $\Sigma_{\mathcal{A}}$, it is decidable in* PSPACE *whether* $\mathcal{A} \models \varphi$.

Proof. In the case of a fully propositional ASM$^+$ transducer $\mathcal{A}$, the Kripke structure $K_{\mathcal{A},D}$ is independent of D (let us denote it by $K_{\mathcal{A}}$). However, $K_{\mathcal{A}}$ is exponential with respect to $\mathcal{A}$ so cannot be constructed in PSPACE. We therefore need a more subtle approach, that circumvents the explicit construction of $K_{\mathcal{A}}$. To do so, we adopt techniques developed in the context of model checking for concurrent programs (modeled by propositional transition systems). Specifically, the model checking algorithm developed by Kupferman, Vardi and Wolper in [28] can be adapted to fully propositional transducers. The algorithm uses a special kind of tree automaton, called *hesitant alternating tree automaton* (HAA) (see [28] for the definition). As shown in [28], for each CTL* formula φ one can construct an HAA A_φ accepting precisely the trees (with degrees in a specified finite set) that satisfy φ. In particular, for a given Kripke structure K, one can construct a product HAA $K \times A_\varphi$ that is non-empty iff $K \models \varphi$. The non-emptiness test can be rendered efficient using the crucial observation that non-emptiness of $K \times A_\varphi$ can be reduced to the non-emptiness of a corresponding word HAA over a 1-letter alphabet, which is shown to be decidable in linear time, unlike the general non-emptiness problem for alternating tree automata. Finally, it is shown that $K \times A_\varphi$ need not be constructed explicitly. Instead, its transitions can be generated on-the-fly from K and φ, as needed in the non-emptiness test for the 1-letter word HAA corresponding to $K \times A_\varphi$. This yields a model checking algorithm of space complexity polynomial in φ and polylogarithmic in K. We refer to [28] for details.

In our case, K is $K_{\mathcal{A}}$, and the input consists of φ and $\mathcal{A}$ instead of φ and $K_{\mathcal{A}}$. The previous approach can be adapted by pushing further the on-the-fly generation of $K_{\mathcal{A}} \times A_\varphi$ by also generating on-the-fly the relevant edges of $K_{\mathcal{A}}$ from $\mathcal{A}$ when needed. This yields a polynomial space algorithm for checking whether $\mathcal{A} \models \varphi$, similar to the algorithm with the same complexity obtained in [28] for model checking of concurrent programs. □

5. Verification of Web applications

We finally present our verification results for Web applications. Most of the results are shown by reducing the verification problem for Web applications to corresponding verification problems for ASM$^+$ transducers. We begin with linear-time properties.

5.1. Linear-time properties of Web applications

As for ASM$^+$ transducers, the decidability results for verification of linear-time properties of Web applications require the input-boundedness restriction. This extends naturally from ASM$^+$ transducers to Web applications.

Definition 5.1. A Web application is *input-bounded* if all formulas used in state, action, and target rules are input bounded, and formulas used in input option rules are $\exists^*$FO formulas in which all state atoms are ground.

Example 5.2. All rules on pages HP, LSP in Example 2.2 are input-bounded. Property (1) in Example 3.2 is trivially input-bounded, as it contains no quantifiers. Property (2) in Example 3.3, however, is not input-bounded because *pname* appears in no input atom. We turn this into an input-bounded property by modeling the catalog database relation with two relations prod_prices($pid, price$) and prod_names($pid, pname$). We can now rewrite property (2) to the input-bounded sentence

$$\forall pid, price \left[\xi'(pid, price)\mathbf{B}\neg\left(\overline{\mathsf{conf}}(\mathsf{name}, price) \wedge \overline{\mathsf{ship}}(\mathsf{name}, pid)\right)\right] \tag{8}$$

where $\xi'(pid, price)$ is short for

$$\mathrm{PP} \wedge \mathsf{pay}(price) \wedge \mathsf{button}(\text{“}authorize\ payment\text{”}) \wedge \mathrm{pick}(pid, price) \wedge \mathrm{prod\_prices}(pid, price). \tag{9}$$

We show the following result on the verification of linear-time properties. Recall from Section 2 that a Web application is error-free if no run ever leads to the special error page.

Theorem 5.3. *The following are decidable*:

(i) *given an input-bounded Web application $\mathcal{W}$, whether it is error free*;
(ii) *given an error-free Web application $\mathcal{W}$ with input-bounded rules and an input-bounded LTL-FO sentence φ over the schema of $\mathcal{W}$, whether $\mathcal{W}$ satisfies φ.*

Furthermore, both problems are PSPACE*-complete for schemas with fixed bound on the arity, and in* EXPSPACE *for schemas with no fixed bound on the arity.*

The lower bound follows immediately from the PSPACE lower bound for ASM^+ transducers. The upper bound is more involved, and requires a reduction to the verification problem for ASM^+ transducers. To begin, we note that part (i) of Theorem 5.3 can be reduced to part (ii).

Lemma 5.4. *For each Web application $\mathcal{W}$ with input-bounded rules there exists an error-free Web application $\mathcal{W}'$ with input-bounded rules, of size quadratic in $\mathcal{W}$, such that $\mathcal{W}$ is error free iff $\mathcal{W}' \models \varphi$, for some fixed input-bounded LTL-FO sentence φ.*

Proof. Let $\mathcal{W} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathbf{W}, W_0, W_\epsilon \rangle$ be a Web application with input-bounded rules. Intuitively, we wish to construct a Web application $\mathcal{W}'$ with a new Web page schema W'_ϵ that is reached according to the rules of the application (and without generating an error), exactly when the error page W_ϵ would be reached in the original Web application. Then it is enough to verify that W'_ϵ is never reached in any run of $\mathcal{W}'$. To this end, we define $\mathcal{W}' = \langle \mathbf{D}, \mathbf{S}', \mathbf{I}, \mathbf{A}, \mathbf{W}', W'_0, W_\epsilon \rangle$ as follows. For each input constant c of $\mathcal{W}$, let p_c be a new propositional symbol, and $\mathbf{S}' = \mathbf{S} \cup \{p_c \mid c \text{ is an input constant of } \mathcal{W}\}$. $\mathbf{W}'$ contains a new Web page schema W'_ϵ defined identically to W_ϵ, and for each Web page schema $W = \langle \mathbf{I}_W, \mathbf{A}_W, \mathbf{T}_W, \mathcal{R}_W \rangle$ of $\mathbf{W}$ different from W_0 and W_ϵ, a Web page schema $W' = \langle \mathbf{I}_W, \mathbf{A}_W, \mathbf{T}'_W, \mathcal{R}'_W \rangle$, where $\mathbf{T}'_W = \mathbf{T}_W \cup \{W'_\epsilon\}$. $\mathcal{R}'_W$ consists of the following rules. The state, input, and action rules of $\mathcal{R}_W$ remain unchanged, except for the addition of one state rule $p_c \leftarrow \mathit{true}$ for each input constant $c \in \mathbf{I}_W$. Before defining the target rules, let ψ_W be $\psi_1 \vee \psi_2 \vee \psi_3$, where:

- ψ_1 is the disjunction of all formulas $\varphi_{V,W} \wedge \varphi_{V',W}$ where $V \neq V'$ and $V \leftarrow \varphi_{V,W}$, $V' \leftarrow \varphi_{V',W}$ are target rules in $\mathcal{R}_W$,
- ψ_2 is the disjunction of all formulas $\varphi_{V,W} \wedge \neg p_c$ where $V \leftarrow \varphi_{V,W}$ is a target rule in $\mathcal{R}_W$ and c is an input constant occurring in some input rule in V but not in $\mathbf{I}_W$, or occurring in some other rule of V, but not in $\mathbf{I}_W \cup \mathbf{I}_V$, and
- ψ_3 is the disjunction of the formulas $\varphi_{V,W} \wedge p_c$ where $V \leftarrow \varphi_{V,W}$ is a target rule in $\mathcal{R}_W$ and c occurs in $\mathbf{I}_V - \mathbf{I}_W$, and $\varphi_{V,W}$ if $c \in \mathbf{I}_W \cap \mathbf{I}_V$.

Intuitively, ψ_W states that the original target rules of W are ambiguous (stated by ψ_1) or the next Web page uses some input constant not yet provided (formula ψ_2), or the next Web page requires as input some constant already provided (stated by ψ_3). The target rules of W' make use of ψ_W:

- each target rule $V \leftarrow \varphi_{V,W}$, where $V \in \mathbf{T}_W$, is replaced by $V \leftarrow \varphi_{V,W} \wedge \neg\psi_W$,
- $W'_\epsilon \leftarrow \psi_W$ is a new target rule.

Finally, W'_0 is a special case. It is defined as above if input rules of W_0 contain no input constants, and the other formulas contain only input constants in $\mathbf{I}_{W_0}$; otherwise, it is defined as $\langle \emptyset, \emptyset, \{W'_\epsilon\}, \{W'_\epsilon \leftarrow \mathit{true}\} \rangle$.

It is easily verified that $\mathcal{W}'$ is error free and $\mathcal{W}'$ is input bounded if $\mathcal{W}$ is input bounded. Also, $\mathcal{W}$ is error free iff the page W'_ϵ is never reached in any run of $\mathcal{W}'$, i.e. $\mathcal{W}'$ satisfies the input-bounded LTL-FO sentence $\mathbf{G}\,\neg W'_\epsilon$. □

The following shows that checking that a Web application is error-free is already PSPACE-hard.

Lemma 5.5. *Checking whether an input-bounded Web application is error free is* PSPACE*-hard.*

Proof. The proof is by reduction from quantified Boolean formula (QBF), known to be PSPACE-complete [21]. Let φ be a quantified Boolean formula (we can assume φ uses just $\vee, \neg, \exists$). Consider the Web application $\mathcal{W}_\varphi = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathbf{W}, W_0, W_\epsilon \rangle$ where:

- $\mathbf{D} = \{R : 1, 0, 1\}$, $\mathbf{S} = \emptyset$, $\mathbf{I} = \{I_0 : 1, I_1 : 1\}$, $\mathbf{A} = \emptyset$, $\mathbf{W} = \{W_0, W_1, W_2\}$.
- $W_0 = \langle \{I_0, I_1\}, \emptyset, \{W_1, W_2\}, \mathcal{R}_{W_0} \rangle$ where $\mathcal{R}_{W_0}$ consists of the input rules

  $$Options_{I_i}(x) \leftarrow R(x),$$

  for $i \in \{0, 1\}$ and the target rules

  $$W_i \leftarrow I_0(0) \wedge I_1(1) \wedge 0 \neq 1 \wedge \varphi', \quad i \in \{1, 2\},$$

  where φ' is defined from φ as follows:
  - each propositional variable x is replaced by $(x = 1)$;
  - disjunction and negation remain unchanged;
  - $\exists x\, \psi$ becomes $\exists x\, ((I_0(x) \vee I_1(x)) \wedge \psi)$.
- $\{W_1, W_2\}$ are arbitrary.

Clearly, $\mathcal{W}_\varphi$ is input-bounded and of size polynomial in φ, and it is error free iff there is no run for which $I_0 = \{0\}$, $I_1 = \{1\}$, and φ' is true. Obviously, there exists a run for which $I_0 = \{0\}$ and $I_1 = \{1\}$. But then φ' has the same value as φ. Therefore, $\mathcal{W}$ is error-free iff φ is false. □

We next reduce the verification of error-free Web applications to verification of ASM$^+$ transducers.

Lemma 5.6. *Let $\mathcal{W} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathbf{W}, W_0, W_\epsilon \rangle$ be an error-free, input-bounded Web application and φ an LTL-FO or CTL$^{(*)}$-FO sentence over the schema of $\mathcal{W}$. There exists an ASM$^+$ transducer $\mathcal{A}$, of size linear in $\mathcal{W}$, such that $\mathcal{W} \models \varphi$ iff $\mathcal{A} \models \varphi$.*

Proof. In brief, the reduction has to overcome two obstacles: (i) simulating the multiple Web schemas of $\mathcal{W}$, and (ii) eliminating the constants from the input schema of $\mathcal{W}$. It is easy to deal with (i): we just simulate the behavior of different Web pages and transitions using new propositional state variables corresponding to the Web pages. Overcoming (ii) makes essential use of the assumption that $\mathcal{W}$ is error free. Indeed, this guarantees that the value of each input constant is only provided once, and that no formula makes use of such constants before they are provided. This allows to assume that the input constants are provided prior to the run, as part of the database.

More precisely, let $\mathcal{A} = \langle \mathbf{D}', \mathbf{S}', \mathbf{I}', \mathbf{A}', \mathcal{R} \rangle$ where:

- $\mathbf{D}' = \mathbf{D} \cup const(\mathbf{I})$, where $const(\mathbf{I})$ denotes the set of input constant symbols in $\mathbf{I}$;
- $\mathbf{S}' = \mathbf{S} \cup \mathbf{W}$, where each $W \in \mathbf{W}$ is taken to be a propositional symbol;
- $\mathbf{I}' = \mathbf{I} - const(\mathbf{I})$;
- $\mathbf{A}' = \mathbf{A}$.

The set of rules $\mathcal{R}$ of $\mathcal{A}$ is defined as follows. For each relational input I of $\mathbf{I}$ we add to $\mathcal{R}$ the input rule $Options_I(\bar{x}) \leftarrow \xi$, where ξ is the disjunction of all formulas $\varphi_{I,W}(\bar{x}) \wedge W$ for which $Options_I(\bar{x}) \leftarrow \varphi_{I,W}(\bar{x})$ is an input rule of the page W in $\mathcal{W}$. We define the state rules next. For each state rule $(\neg)S(\bar{x}) \leftarrow \varphi^{\epsilon}_{S,W}(\bar{x})$ of $\mathbf{W}$, we add a state rule $(\neg)S(\bar{x}) \leftarrow \varphi^{\epsilon}_{S,W}(\bar{x}) \wedge W$ to $\mathcal{R}$. In addition, for each target rule $V \leftarrow \varphi_{V,W}$ of $\mathbf{W}$ we add to $\mathcal{R}$ the state rules $V \leftarrow \varphi_{V,W} \wedge W$ and, if $V \neq W$, $\neg W \leftarrow \varphi_{V,W} \wedge W$. The action rules of $\mathcal{R}$ consist of all rules $A(\bar{x}) \leftarrow \varphi(\bar{x}) \wedge W$ for which $A(\bar{x}) \leftarrow \varphi(\bar{x})$ is an action rule of Web page schema W in $\mathbf{W}$. □

Theorem 5.3(ii) and the PSPACE upper bound (EXPSPACE with no fixed bound on arities) now follow from Lemma 5.6 and Theorem 4.9.

The undecidability results developed in Section 4.2 carry over to verification of Web applications due to the reduction provided by Lemma 5.6. The following is a corollary of Lemma 5.6 and Theorems 4.10–4.13.

Corollary 5.7.

1. *There exists a fixed input-bounded LTL-FO sentence φ for which it is undecidable, given an error-free, input-bounded Web application $\mathcal{W}$ with input options defined by quantifier-free FO formulas over database and state relations, whether $\mathcal{W} \models \varphi$.*
2. *It is undecidable, given an error-free, input-bounded Web application $\mathcal{W}$ with state projections and input-bounded LTL-FO sentence φ, whether $\mathcal{W} \models \varphi$.*
3. *It is undecidable, given an error-free, input-bounded Web application $\mathcal{W}$ with lossless input and an input-bounded LTL-FO sentence φ, whether $\mathcal{W} \models \varphi$.*
4. *It is undecidable, given an error-free, input-bounded Web application $\mathcal{W}$ with functional dependencies $\mathcal{F}$ on its database schema, and an input-bounded LTL-FO sentence φ, whether $\mathcal{W} \models_{\mathcal{F}} \varphi$.*

5.2. Branching-time properties of Web applications

Lemma 5.6 and Theorem 4.14 imply the following undecidability result:

Corollary 5.8. *It is undecidable, given an error-free, input-bounded Web application $\mathcal{W}$ and input-bounded CTL-FO sentence φ, whether $\mathcal{W} \models \varphi$.*

We therefore consider next several restrictions leading to decidability of the verification problem for CTL$^{(*)}$-FO sentences. Some of the results mirror directly those obtained for ASM$^+$ transducers in Section 4.3, while others require some development specific to the Web application formalism.

5.2.1. Propositional input-bounded Web applications

The first restriction we consider for Web applications is an extension of propositional input-bounded ASM$^+$ transducers. The restriction limits input-bounded Web applications by requiring all states and actions to be propositional. Furthermore, no rules can use $\mathbf{Prev_I}$ atoms. We also call such Web applications *propositional*. As for ASM$^+$ transducers, in a propositional Web application, inputs can still be parameterized in the Web application specification. The CTL* formulas we consider are propositional and use input, action, state, and Web page symbols, viewed as propositions (recall that the CTL* formulas used for propositional ASM$^+$ transducers used only the states). Satisfaction of such a CTL* formula by a Web application is defined as for CTL*-FO, where truth of propositional symbols in a given configuration $\langle V, S, I, A\rangle$ is defined as follows: a Web page symbol is true iff it equals V, a state symbol s is true iff $s \in S$, an input symbol J is true iff $J \in \mathbf{I}_V$, and an action symbol a is true iff $a \in A$.

Example 5.9. CTL$^{(*)}$-FO is particularly useful for specifying navigational properties of Web applications. Note that these applications do not necessarily have to be propositional; we could abstract their predicates to propositional symbols, thus concentrating only on reachability properties. This is in the spirit of program verification, where program variables are first abstracted to booleans [13,23], in order to check CTL* properties such as liveness. For our running example, abstracting all non-input atoms to propositions, we could ask whether from any page it is possible to navigate to the home page HP using the following CTL sentence:

$$A\mathbf{G}E\mathbf{F}(\mathrm{HP}).$$

The following CTL property states that, after login, the user can reach a page where he can authorize payment for a product:[7]

$$A\mathbf{G}\big((\mathrm{HP} \wedge \mathsf{button}(\text{“}\textit{login}\text{”})) \rightarrow E\mathbf{F}(\mathsf{button}(\text{“}\textit{authorize payment}\text{”}))\big)$$

where $\mathsf{button}(\text{“}\textit{login}\text{”})$, $\mathsf{button}(\text{“}\textit{authorize payment}\text{”})$ denote the corresponding propositions. In the specification of the abstracted application, we can still allow in the home page HP a state rule that checks successful login:

$$\mathrm{logged\_in} \leftarrow \underline{\mathsf{users}}(\mathsf{name}, \mathsf{password}) \wedge \mathsf{button}(\text{“}\textit{login}\text{”}).$$

[7] The most important property in electronic commerce ☺.

For a given Web application $\mathcal{W} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathbf{W}, W_0, W_\epsilon \rangle$, we denote by $\Sigma_\mathcal{W}$ the propositional vocabulary consisting in all symbols in $\mathbf{S} \cup \mathbf{I} \cup \mathbf{A} \cup \mathbf{W}$. By abuse of notation, we use the same symbol for a relation R in the vocabulary of $\mathcal{W}$ and for the corresponding propositional symbol in $\Sigma_\mathcal{W}$.

Theorem 5.10. *Given a propositional, input-bounded, error-free Web application $\mathcal{W}$ and a CTL* formula φ over $\Sigma_\mathcal{W}$, it is decidable whether $\mathcal{W} \models \varphi$. The complexity of the decision procedure is* CO-NEXPTIME *if φ is in CTL, and* EXPSPACE *if φ is in CTL*.*

Theorem 5.10 is a consequence of Theorem 4.15 on ASM$^+$ transducers together with the following.

Lemma 5.11. *For each propositional, input-bounded, error-free Web application $\mathcal{W}$ and CTL* formula φ over $\Sigma_\mathcal{W}$, one can construct in linear time a propositional, input-bounded ASM$^+$ transducer $\mathcal{A}$ such that $\Sigma_\mathcal{A} \supseteq \Sigma_\mathcal{W}$ and $\mathcal{W} \models \varphi$ iff $\mathcal{A} \models \varphi$.*

Proof. The proof is similar to that of Lemma 5.6. In order for the states of $\mathcal{A}$ to contain all propositions in $\Sigma_\mathcal{W}$, one has to introduce, in addition to the states for Web pages introduced in the proof of Lemma 5.6, new states for all actions and inputs, that are true precisely when the corresponding propositional symbol in $\Sigma_\mathcal{W}$ evaluates to true in the semantics of CTL* formulas over $\Sigma_\mathcal{W}$ described above. This is straightforward and details are omitted. □

The complexity of the decision problem of Theorem 4.15 can be decreased under additional assumptions. The following result focuses on verification of navigational properties of Web sites, expressed by CTL* formulas over alphabet $\mathbf{W}$.

Corollary 5.12. *Let $\mathbf{S}$ be a fixed set of state propositions and $\mathbf{D}$ a fixed database schema. Given a propositional, input-bounded, error-free Web application $\mathcal{W}$ with states $\mathbf{S}$ and database schema $\mathbf{D}$, and a CTL* formula φ over $\mathbf{W}$, it is decidable in* PSPACE *whether $\mathcal{W} \models \varphi$.*

Proof. The decision procedure is similar to that for Theorem 4.15. Since $\mathbf{S}$ is fixed and φ refers only to $\mathbf{W}$, it is enough to retain, in labels of $\lambda(\mathcal{T}_{\mathcal{W},D})$ only the states and Web page names. Since $\mathcal{W}$ is error free, there is exactly one Web page name per label. It follows that the number of pairs $\langle \Sigma, \bar{\Sigma} \rangle$ occurring in $\lambda(\mathcal{T}_{\mathcal{W},D})$ is quadratic in $\mathbf{W}$, so the formula ξ has polynomially many variables, and the size of the database D_0 is polynomial in $\mathcal{W}$. The Kripke structure $K_{\mathcal{W},D_0}$ can now be constructed in PSPACE with respect to $\mathcal{W}$, and checking φ can be done in PSPACE with respect to $K_{\mathcal{W},D_0}$ and φ. Altogether, checking that $\mathcal{W} \models \varphi$ is done in PSPACE with respect to $\mathcal{W}$ and φ. □

Another special case of interest, as for ASM$^+$ transducers, involves Web applications that are entirely propositional. Thus, the database plays no role in the specification: inputs, states, and actions are all propositional, and the rules do not use the database. Such Web application are called *fully propositional*. We can show the following, which is a direct consequence of Lemma 5.6 and Theorem 4.18.

Theorem 5.13. *Given a fully propositional, error-free Web application $\mathcal{W}$ and a CTL* formula φ over $\Sigma_\mathcal{W}$, it is decidable in* PSPACE *whether $\mathcal{W} \models \varphi$.*

One may wonder if the restrictions of Theorem 5.10 can be relaxed without compromising the decidability of verification. In particular, it would be of interest if one could lift some of the restrictions on the propositional nature of states and actions. Unfortunately, we have shown that allowing parameterized actions leads to undecidability of verification, even for CTL formulas whose only use of action predicates is to check emptiness. The proof is by reduction of the implication problem for functional and inclusion dependencies. We omit the details.

5.2.2. Web applications with input-driven search

The restrictions considered so far require states of a Web application to be propositional, and do not allow the use of $\mathbf{Prev_I}$ atoms. Although adequate for some verification tasks, this is a serious limitation in many situations, since no values can be passed on from one Web page to another. We next alleviate some of this limitation by considering

Web applications that allow limited use of $\mathbf{Prev_I}$ atoms. This can model commonly arising applications involving a user-driven search, going through consecutive stages of refinement. More formally:

Definition 5.14. A *Web application with input-driven search* is an input-bounded Web application $\mathcal{W} = \langle \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathbf{W}, W_0, W_\epsilon \rangle$ where:

- $\mathbf{I}$ consists of a single unary relation I;
- $\mathbf{S}$ consists of propositional states including *not-start*;
- $\mathbf{A}$ is propositional;
- $\mathbf{D}$ includes a constant symbol i_0 and a designated binary relation R_I;
- the state rule for *not-start* is *not-start* $\leftarrow \neg$*not-start*;
- the input option rule for I is in all Web pages of the form

$$\mathit{Options}_I(y) \leftarrow (\neg \mathit{not\text{-}start} \wedge y = i_0) \vee \big(\mathit{not\text{-}start} \wedge \exists x \big(\mathit{prev}_I(x) \wedge R_I(x, y)\big) \wedge \varphi(y)\big)$$

where $\varphi(y)$ is a quantifier-free formula over $\mathbf{D} \cup \mathbf{S}$ with free variable y.

Note that *not-start* is false at the start of the computation and true thereafter. To initialize the search, the first input option is the constant i_0. Subsequently (when *not-start* is true), if x was the previously chosen input, the allowed next inputs are the y's for which $R_I(x, y) \wedge \varphi(y)$ holds, where R_I is the special input search relation and φ places some additional condition on y involving the database and the propositional states.

Example 5.15. Consider a variation of a computer-selling Web site which does not just partition its products into desktops and laptops, but rather uses the more complex classification depicted in Fig. 3. The user can search the hierarchy of categories, and will only see a certain category if it is currently in stock, as reflected by the database. The propositional state new is set on the page which offers the choice between new and used products. The page schemas for new and old computers are reused, so when generating the options, the Web site must consult state new to distinguish among new and old products. We can abstract this Web site as a Web application with input-driven search, in which the binary database relation R_I is a graph which contains as a subgraph the one in Fig. 3, and in which the unary database relations such as newDesktop, usedDesktop, usedLaptop contain the in-stock products. Here is the input rule corresponding to the desktop search page:

$$\begin{aligned}\mathit{Options}_I(y) \leftarrow{}& (\neg \mathit{not\text{-}start} \wedge y = i_0) \vee \mathit{not\text{-}start} \wedge \exists x \big(\mathit{prev}_I(x) \wedge R_I(x, y)\big) \\ &\wedge \big(\mathrm{new} \wedge \underline{\mathsf{newDesktop}}(y) \vee \neg \mathrm{new} \wedge \underline{\mathsf{usedDesktop}}(y)\big). \qquad (10)\end{aligned}$$

We can show the following.

Theorem 5.16. *Given a Web application with input-driven search $\mathcal{W}$ and a CTL* formula φ, it is decidable whether $\mathcal{W} \models \varphi$ in* EXPTIME *if φ is in CTL, and* 2-EXPTIME *if φ is in CTL*.*

Proof. We reduce the problem of checking whether $\mathcal{W} \models \varphi$ to the satisfiability problem for CTL$^{(*)}$ formulas. As mentioned in Section 3.2, this is known to be EXPTIME-complete for CTL and 2-EXPTIME complete for CTL*. We consider Kripke structures over the alphabet $\Sigma_\mathcal{W} \cup \mathbf{D}$. Intuitively, each node of the Kripke structure represents a configuration, and its label represents the relevant information about the configuration: the set of propositions in $\Sigma_\mathcal{W}$ that hold, and the type of the current input with respect to the database, i.e. the set of relations Q in $\mathbf{D} - \{R_I\}$ for which

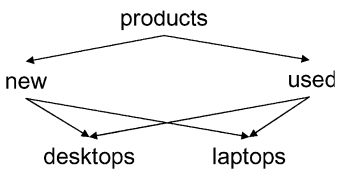

Fig. 3. Fragment of R_I for Example 5.15.

$y^k \in Q$, where k is the arity of Q and y the current input. Note that the types of different inputs are independent of each other because inputs are unary, so every Kripke structure can be viewed as representing an input choice relation R_I together with type assignments for the elements of R_I. In addition, in order for a Kripke structure to represent an actual run of $\mathcal{W}$, the assignments of literals of $\Sigma_{\mathcal{W}}$ to nodes has to be consistent with the rules of $\mathcal{W}$. However, this can be easily expressed by a CTL formula ρ computable in polynomial time from $\mathcal{W}$. It follows that $\mathcal{W} \models \varphi$ iff $\rho \wedge \neg\varphi$ is unsatisfiable. The latter is a CTL formula if φ is in CTL, and a CTL* formula if φ is in CTL*. □

6. Conclusions

We have identified a practically appealing and fairly tight class of Web applications and linear-time temporal formulas for which verification is decidable. The complexity of verification is PSPACE-complete (for fixed database arity). This is quite reasonable as static analysis goes.[8] For branching-time properties, we identified decidable restrictions for which the complexity of verification ranges from PSPACE to 2-EXPTIME. To obtain these results, we used a mix of techniques from logic and model checking.

6.0.3. Verification of Web service compositions

An important aspect of Web applications is their interaction and the composition of multiple services required for complex applications (e.g., see [27]). In [16] we extend the present study of verification of single Web applications by considering verification of *compositions* of Web service peers that interact asynchronously by exchanging messages. We consider two formalisms for specification of correctness properties of compositions, namely LTL-FO and conversation protocols. For both formalisms, we map the boundaries of verification decidability. We also address modular verification, in which the correctness of a composition is predicated on the properties of its environment.

6.0.4. Implementation

We have implemented a verifier for input-bounded LTL-FO properties and Web applications, by coupling the pseudorun technique described in Section 4.1 with various database heuristics. The verifier is described in [15,17]. Verification times obtained were surprisingly good, often on the order of seconds. We plan to pursue this investigation in order to establish the boundary of practical feasibility of our approach, and extend the implementation to compositions of Web services.

6.0.5. Future work

Other interesting aspects of Web application verification could not be addressed in this paper and are left for future work. We mention two issues that deserve further study: sessions and multiple users.

In practical Web applications it is not always realistic to assume that verification applies to *all* possible runs of the application. This may be due to various reasons: there may be a need to verify properties of complex applications in a modular fashion, the restrictions needed for decidability may only hold for certain portions of runs, etc. Let us call portions of runs to be verified *sessions*. Some sessions can be specified implicitly within the temporal formula to be verified, while others may require explicit definition by other means. It is of interest to understand what types of sessions can be verified by our approach. For instance, in our running example, the default assumption is that sessions consist of single-user runs beginning at login and ending at logout. However, other types of sessions can be fit to our restrictions, including certain kinds of multi-user sessions.

Acknowledgments

We thank Caroline Cruz and Dayou Zhou for their help in implementing the demo Web site for our running example, and Ioana Manolescu, Monica Marcus, and Marc Spielmann for comments on this and a previous version of this paper. We also thank the anonymous referees for their careful reading of the manuscript and useful comments and suggestions that resulted in an improved paper.

[8] Recall that even testing inclusion of two conjunctive queries is NP-complete!

References

[1] S. Abiteboul, L. Herr, J.V. den Bussche, Temporal versus first-order logic to query temporal databases, in: Proc. ACM PODS, 1996, pp. 49–57.
[2] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison–Wesley, 1995.
[3] S. Abiteboul, V. Vianu, B. Fordham, Y. Yesha, Relational transducers for electronic commerce, J. Comput. System Sci. 61 (2) (2000) 236–269, extended abstract in PODS 98.
[4] D. Berardi, D. Calvanese, G. Giacomo, R. Hull, M. Mecella, Automatic composition of transition-based semantic Web services with messaging, in: VLDB, 2005.
[5] A.J. Bonner, M. Kifer, An overview of transaction logic, Theoret. Comput. Sci. 133 (2) (1994) 205–265.
[6] E. Borger, E. Gradel, Y. Gurevich, The Classical Decision Problem, Springer, 1997.
[7] BPML.org., Business process modeling language, http://www.bpmi.org.
[8] M. Brambilla, S. Ceri, S. Comai, P. Fraternali, I. Manolescu, Specification and design of workflow-driven hypertexts, J. Web Eng. 1 (1) (2002).
[9] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, M. Matera, Designing Data-Intensive Web Applications, Morgan Kaufmann, 2002.
[10] A.K. Chandra, M. Vardi, The implication problem for functional and inclusion dependencies is undecidable, SIAM J. Comput. 14 (3) (1985) 671–677.
[11] E.M. Clarke, O. Grumberg, D.A. Peled, Model Checking, MIT Press, 2000.
[12] DAML-S Coalition, A. Ankolekar, et al., DAML-S: Web service description for the semantic Web, in: The Semantic Web—ISWC, 2002, pp. 348–363.
[13] S. Das, D.L. Dill, S. Park, Experience with predicate abstraction, in: Computer Aided Verification (CAV), 1999, pp. 160–171.
[14] H. Davulcu, M. Kifer, C.R. Ramakrishnan, I.V. Ramakrishnan, Logic based modeling and analysis of workflows, in: Proc. ACM PODS, 1998, pp. 25–33.
[15] A. Deutsch, L. Sui, V. Vianu, D. Zhou, A system for specification and verification of interactive, data-driven web applications, in: ACM SIGMOD International Conf. on Management of Data, 2006, demo.
[16] A. Deutsch, L. Sui, V. Vianu, D. Zhou, Verification of communicating data-driven Web services, in: ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS), 2006.
[17] A. Deutsch, M. Marcus, L. Sui, V. Vianu, D. Zhou, A verifier for interactive, data-driven web applications, in: ACM SIGMOD International Conf. on Management of Data, 2005.
[18] E.A. Emerson, Temporal and modal logic, in: J.V. Leeuwen (Ed.), Handbook of Theoretical Computer Science, vol. B: Formal Models and Sematics, North-Holland/MIT Press, 1990, pp. 995–1072.
[19] M.F. Fernández, D. Florescu, A.Y. Levy, D. Suciu, Declarative specification of web sites with Strudel, VLDB J. 9 (1) (2000) 38–55.
[20] D. Florescu, K. Yagoub, P. Valduriez, V. Issarny, WEAVE: A data-intensive web site management system (software demonstration), in: Proc. of the Conf. on Extending Database Technology (EDBT), 2000.
[21] M.R. Garey, D.S. Johnson, Computers and Intractability, Freeman, 1979.
[22] D. Georgakopoulos, M.F. Hornick, A.P. Sheth, An overview of workflow management: From process modeling to workflow automation infrastructure, Distrib. Parallel Dat. 3 (2) (1995) 119–153.
[23] S. Graf, H. Saidi, Construction of abstract state graphs with pvs, in: Computer Aided Verification (CAV), 1997, pp. 72–83.
[24] P.T. Graunke, R.B. Findler, S. Krishnamurthi, M. Felleisen, Modeling web interactions, in: European Symposium on Programming, 2003.
[25] D. Harel, On the formal semantics of statecharts, in: Proc. LICS, 1987, pp. 54–64.
[26] G. Holzmann, The Spin Model Checker—Primer and Reference Manual, Addison–Wesley, 2003.
[27] R. Hull, M. Benedikt, V. Christophides, J. Su, E-services: A look behind the curtain, in: Proc. ACM PODS, 2003, pp. 1–14.
[28] O. Kupferman, M. Vardi, P. Wolper, An automata-theoretic approach to branching-time model checking, J. ACM 47 (2) (2000) 312–360.
[29] D.R. Licata, S. Krishnamurthi, Verifying interactive web programs, in: IEEE International Symposium on Automated Software Engineering, 2004.
[30] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems, Springer, 1991.
[31] Z. Manna, A. Pnueli, Temporal Verification of Reactive Systems: Safety, Springer, 1995.
[32] G. Mecca, P. Merialdo, P. Atzeni, Araneus in the era of XML, IEEE Data Eng. Bull. 22 (3) (1999) 19–26.
[33] S. Nakajima, Verification of Web service flows with model-checking techniques, in: International Symposium on Cyber Worlds (CW), 2002, pp. 378–385.
[34] S. Narayanan, S.A. McIlraith, Simulation, verification and automated composition of Web services, in: International World Wide Web Conference (WWW), 2002, pp. 77–88.
[35] A.P. Sistla, M.Y. Vardi, P. Wolper, The complementation problem for Buchi automata with applications to temporal logic, Theoret. Comput. Sci. 49 (1987) 217–237.
[36] M. Spielmann, Abstract state machines: Verification problems and complexity, PhD thesis, RWTH, Aachen, 2000.
[37] M. Spielmann, Verification of relational transducers for electronic commerce, J. Comput. System Sci. 66 (1) (2003) 40–65, extended abstract in PODS 2000.
[38] M.Y. Vardi, P. Wolper, An automata-theoretic approach to automatic program verification, in: Symp. on Logic in Computer Science, 1986.
[39] D. Wodtke, G. Weikum, A formal foundation for distributed workflow execution based on state charts, in: Proc. ICDT, 1997, pp. 231–246.
[40] Workflow management coalition, http://www.wfmc.org, 2001.
[41] Web services description language (WSDL 1.1), http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 2001.
[42] Web services flow language (WSFL 1.0), http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, 2001.