Verification of Relational Transducers for Electronic Commerce

Marc Spielmann
Mathematische Grundlagen der Informatik
RWTH Aachen, D-52056 Aachen, Germany
spielmann@informatik.rwth-aachen.de

ABSTRACT

Motivated by recent work of Abiteboul, Vianu, Fordham, and Yesha [3] we investigate the verifiability of transaction protocols specifying the interaction of multiple parties via a network, where each party is equipped with an (active) database that participates in the interaction. Such transaction protocols typically occur in the context of electronic commerce applications and can be formalized as relational transducers. We introduce a class of powerful relational transducers based on Gurevich's abstract state machines and show that several verification problems related to electronic commerce applications can be solved for these transducers. Our approach is, in some sense, complementary to the approach in [3].

1. INTRODUCTION

One of the main reasons for the enormous and still accelerating growth of the World Wide Web is the strong interest of commercial enterprises in electronic commerce, i.e., in offering services on and conducting business via the Web. Electronic commerce offers challenges to various disciplines of computer science, such as cryptography (for security and authentication), databases (to support electronic transactions), and formal verification (for the design and validation of transaction protocols). A stimulating introduction to electronic commerce from the point of view of computer science can be found in [3; 4].

In this paper we investigate the verifiability of transaction protocols specifying the interaction of multiple parties via a network. Such transaction protocols typically occur in the context of electronic commerce applications, where they are also called *business models* [3]. As an example, consider a scenario where customers, presumably on remote sites, interact with the computer system of a supplier. In this setting the transaction protocol (or business model) of the supplier specifies how to react to requests of customers. Possible actions may include delivery of items, broadcasting messages to customers, updating the supplier's database, etc.

PODS 2000 Dallas, Texas USA

As a general framework to formalize business models *relational transducers* have been put forward in [3]. A relational transducer can be seen as a computational device that translates a sequence of input relations into a sequence of output relations. More precisely, a state of a relational transducer is a relational database containing the (dynamic) *memory* and the (static) *database* of the transducer. In every computation step, the transducer receives from the environment input relations (e.g., lists of orders and payments from various customers) and reacts by producing output relations (e.g., lists of bills and items to be sent to customers) and by updating its memory. The latter contains all dynamic data of the transducer necessary to keep track of ongoing transactions (e.g., a list of the currently ordered items). The role of the database of the transducer is to provide all static data that does not change during a transaction (e.g., the catalog of a supplier).

In [3] several verification problems concerning relational transducers have been identified and partially solved, among them:

(1) the problem of **verifying temporal properties** of relational transducers ("Does a business model meet its specification?"),

(2) the problem of **deciding equivalence** (**containment**) of relational transducers ("Are two business models equivalent with respect to transactions? Does a business model that was customized by a business partner still conform to the original model?"), and

(3) the **log validation** problem ("Does a transaction carried out on a customer's site actually conform to the supplier's business model?").

Although each of these problems is undecidable in general, positive results, i.e., decidability of variants of the problems, were obtained for a class of restricted relational transducers, called *Spocus transducers* [3]. Since containment (and thus equivalence) is decidable only for particular Spocus transducers—namely those with *full log*—we advocate here a stronger notion of equivalence, which we call *log equivalence*. Intuitively, two relational transducers are log equivalent if they produce the same semantically significant output whenever they are fed with the same input. We think that log equivalence can provide a useful notion of equivalence for customization of relational transducers.

In the present paper we investigate the decidability of the above verification problems (1)–(3) for relational transducers more powerful than Spocus transducers. More precisely, we consider relational transducers defined by means of simple rule-based *abstract state machine* programs. Abstract state machines (ASMs, formerly called evolving algebras) [11; 12] have become the foundation of a successful methodology for specification and verification of dynamic systems [5]. Our transducers, which we call *ASM (relational) transducers*, are more powerful than Spocus transducers due to the following two reasons:

- Rule applications are in general guarded by **first-order formulas**. In contrast, Spocus transducers are defined by means of semi-positive datalog rules, i.e., rules guarded by only conjunctive formulas (in which atomic subformulas may occur negated) [2].
- The memory relations of ASM transducers (called state relations in [3]) are **not necessarily cumulative**. Indeed, ASM transducers can be seen as active databases with immediate triggering of insertion and deletion actions [15; 2]. On the contrary, Spocus transducers have no influence on the contents of their memory, which merely accumulate all previous input.

As evidence for the computational power of ASM transducers, we show that they compute precisely the class of PSPACE-computable queries on ordered databases (without interacting with their environment).
Since none of the above problems (1)–(3) is, in its full generality, decidable for ASM transducers, we concentrate our investigations on natural restrictions of these problems. The restrictions we impose are natural in the sense that they occur often in electronic commerce applications such that solutions of the restricted problems are still of practical importance. To be more precise, we show that problems (1)–(3) are decidable for ASM transducers if one of the following two conditions is satisfied:

- **All (static) database relations of a relational transducer are known.** This restriction is applicable if, say, the catalog of a supplier does not change frequently, such that it becomes feasible to adjust verification to the currently valid catalog.
- **The maximal input flow which a relational transducer is exposed to is a priori limited.** The *maximal input flow* is, roughly speaking, the maximal amount of input data forwarded to a transducer in one computation step. This restriction should be applicable in most electronic commerce applications. It is motivated by the observation that a relational transducer running in a realistic environment will never receive 'too much' input per computation step, due to physical and technical limitations of the environment. For instance, the maximal input flow of a transducer running on an internet server is limited by the number of clients accepted on the server, the capacity of the network, the server's one-step capacity, etc.

Under the first condition we obtain decidability for all ASM transducers. Under the second condition decidability is obtained only for a class of restricted ASM transducers, called *ASM transducers with input-bounded quantification*. Transducers of the latter type are restricted in the sense that first-order quantification in the guards of their rules is bounded to the active domain of the current input.

As it turns out, the maximal arity of the relations employed by ASM transducers is a major source for the complexity of the verification problems. We show that, if the maximal arity of the employed relations is a priori bounded, then all our restricted problems are PSPACE-complete. Bounding the maximal arity should be no serious obstacle for real-life applications, since the arities of relations used in practice tend to be rather small. On the other hand, if there is no fixed upper bound on the arities of relations, then almost all our restricted problems become EXPSPACE-complete.

Related work. Our investigations are motivated by recent work of Abiteboul, Vianu, Fordham, and Yesha [3]. The general model of relational transducers and (variants of) the verification problems considered here are adapted from their framework.
ASM relational transducers are based on the ASM computation model of Gurevich [11; 12]. For applications of ASMs in database theory see [10] and [9]. The latter work employs ASMs as a formalism to specify active databases for electronic commerce applications. In this context note that ASM transducers can be viewed as active databases with immediate triggering [15; 2]. Also, a run of an ASM transducer is a temporal database in the spirit of [1]. We use first-order temporal logic [8; 1] to specify temporal properties of relational transducers (see problem (1) above).
Our decidability results are implied by a reduction to the finite satisfiability problem for existential transitive closure logic (E+TC) [7], which is decidable by results in [14; 16]. The reduction borrows elements of a construction due to Immerman and Vardi [13]. An application of the same construction in a related context can be found in [18]. Our hardness results follow from results in [19; 20; 6].

Outline. In the next section, we provide a self-contained definition of ASM relational transducers. In Section 3, we formally define the verification problems that we will be concerned with in this paper (see problems (1)–(3) above). Since none of these problems is in its general formulation decidable for ASM transducers, we introduce natural restrictions of the problems in Section 4. In Section 5, we define our weak type of ASM transducers, i.e., ASM transducers with input-bounded quantification, and present our main results concerning the verifiability of ASM transducers (of both types). In the last section, we briefly discuss some open problems.

2. ASM RELATIONAL TRANSDUCERS

We consider relational transducers as defined in [3], though we will not presuppose any familiarity with that paper. We start with a self-contained definition of a powerful model of relational transducers based on *abstract state machines* (ASMs) [11; 12]. Our transducers, which we call *ASM (relational) transducers*, are defined by means of simple rule-based ASM programs.

Syntax. A *transducer vocabulary* Υ is a quintuple ($\Upsilon_{\rm in}$, $\Upsilon_{\rm db}$, $\Upsilon_{\rm mem}$, $\Upsilon_{\rm out}$, $\Upsilon_{\rm log}$) of finite relational vocabularies (schemas) where the first four vocabularies are pairwise disjoint and $\Upsilon_{\rm log} \subseteq \Upsilon_{\rm in} \cup \Upsilon_{\rm out}$. Sometimes we also denote by Υ the relational vocabulary $\Upsilon_{\rm in} \cup \Upsilon_{\rm db} \cup \Upsilon_{\rm mem} \cup \Upsilon_{\rm out}$. The intended meaning will be clear from the context.

In the following FO stands for *first-order logic* with equality [7; 2]. (In this paper we ignore questions concerning domain independence, safety of queries, etc.)

Definition 1. Let Υ be a transducer vocabulary. An *ASM transducer program* Π over Υ is a finite set of rules of the form

$$\texttt{if } \varphi(\bar{x}) \texttt{ then } (\neg)R(\bar{x})$$

where $\varphi(\bar{x})$ (the *guard* of the rule) is a FO formula over $\Upsilon_{\text{in}} \cup \Upsilon_{\text{db}} \cup \Upsilon_{\text{mem}}$ with $\text{free}(\varphi) = \{\bar{x}\}$, and $R(\bar{x})$ is an atomic FO formula over $\Upsilon_{\text{mem}} \cup \Upsilon_{\text{out}}$ which occurs positively on the right-hand side of the rule if $R \in \Upsilon_{\text{out}}$.

The semantics of an ASM transducer program Π is similar to the one-step semantics of a datalog$^{\neg\neg}$ program [2], except that Π treats inconsistent updates as 'no operations'.

Semantics. Let Π an ASM transducer program over Υ. Π defines a relational transducer $T_{(\Upsilon,\Pi)}$ over Υ as follows.
A *state* $\mathcal{S}$ over Υ (of the relational transducer $T_{(\Upsilon,\Pi)}$) is a finite structure over the relational vocabulary Υ. For every $\Upsilon' \subseteq \Upsilon$, let $\mathcal{S}|\Upsilon'$ denote the *reduct* of $\mathcal{S}$ to Υ', i.e., the structure over Υ' obtained from $\mathcal{S}$ by removing the interpretations of the symbols in $\Upsilon - \Upsilon'$. To ease notation, we regularly write $\mathcal{S}|_{\text{db}}$ instead of $\mathcal{S}|\Upsilon_{\text{db}}$, $\mathcal{S}|_{\text{db,mem}}$ instead of $\mathcal{S}|(\Upsilon_{\text{db}} \cup \Upsilon_{\text{mem}})$, and so forth. A state $\mathcal{S}$ can be viewed as being composed of

- the *input component* $\mathcal{S}|_{\text{in}}$ (containing the current input of $T_{(\Upsilon,\Pi)}$ in state $\mathcal{S}$),
- the *database component* $\mathcal{S}|_{\text{db}}$ (containing all static data that does not change during a transaction),
- the *memory component* $\mathcal{S}|_{\text{mem}}$ (containing all temporary data that may change in every computation step), and
- the *output component* $\mathcal{S}|_{\text{out}}$ (containing the output produced during the last computation step).

Formally, the *relational transducer* $T_{(\Upsilon,\Pi)}$ over Υ defined by Π is a function which maps every state $\mathcal{S}$ over Υ to a finite structure $T_{(\Upsilon,\Pi)}(\mathcal{S})$ over $\Upsilon_{\text{mem}} \cup \Upsilon_{\text{out}}$. Intuitively, $T_{(\Upsilon,\Pi)}(\mathcal{S})$ determines the memory and output component of a successor state $\mathcal{S}'$ of $T_{(\Upsilon,\Pi)}$ in state $\mathcal{S}$. The (new) input component of $\mathcal{S}'$ will be provided by the environment. The database component of $\mathcal{S}'$ is copied from $\mathcal{S}$.
To simplify the definition of $T_{(\Upsilon,\Pi)}$, let us assume that,

- whenever $(\neg)R(\bar{x})$ is the right-hand side of a rule in Π, then the variable tuple $\bar{x}$ consists of pairwise distinct variables, and
- whenever $(\neg)R(\bar{x})$ and $(\neg)R'(\bar{y})$ are the right-hand sides of two different rules in Π and $R = R'$, then the variable tuples $\bar{x}$ and $\bar{y}$ are identical.

For each $R \in \Upsilon_{\text{mem}} \cup \Upsilon_{\text{out}}$, define the following FO formulas:

$$\begin{aligned}
\varphi_R(\bar{x}) &:= \bigvee\{\varphi(\bar{x}) : (\texttt{if } \varphi(\bar{x}) \texttt{ then } R(\bar{x})) \in \Pi\} \\
\psi_R(\bar{x}) &:= \bigvee\{\varphi(\bar{x}) : (\texttt{if } \varphi(\bar{x}) \texttt{ then } \neg R(\bar{x})) \in \Pi\} \\
\chi_R(\bar{x}) &:= (\varphi_R(\bar{x}) \wedge \neg\psi_R(\bar{x})) \vee \\
&\quad\ (\varphi_R(\bar{x}) \wedge \psi_R(\bar{x}) \wedge R(\bar{x})) \vee \\
&\quad\ (\neg\varphi_R(\bar{x}) \wedge \neg\psi_R(\bar{x}) \wedge R(\bar{x})),
\end{aligned}$$

where $\bigvee \varnothing := \mathit{false}$. Finally, for every state $\mathcal{S}$ over Υ, let the finite structure $T_{(\Upsilon,\Pi)}(\mathcal{S})$ over $\Upsilon_{\text{mem}} \cup \Upsilon_{\text{out}}$ be defined as follows:

- the universe of $T_{(\Upsilon,\Pi)}(\mathcal{S})$ is that of $\mathcal{S}$,
- for every $R \in \Upsilon_{\text{mem}}$, $R^{T_{(\Upsilon,\Pi)}(\mathcal{S})} := \chi_R^{\mathcal{S}}$, and
- for every $R \in \Upsilon_{\text{out}}$, $R^{T_{(\Upsilon,\Pi)}(\mathcal{S})} := \varphi_R^{\mathcal{S}}$,

where $\chi_R^{\mathcal{S}}$ and $\varphi_R^{\mathcal{S}}$ denote the answer relations of the queries χ_R and φ_R on $\mathcal{S}$, respectively. This concludes the definition of $T_{(\Upsilon,\Pi)}$.

Definition 2. An *ASM (relational) transducer* T is a pair (Υ, Π) consisting of a transducer vocabulary Υ and an ASM transducer program Π over Υ. ASM-T denotes the class of ASM transducers.

For the sake of brevity, we from now on blur the strict distinction between an ASM transducer $T = (\Upsilon, \Pi)$ and the relational transducer $T_{(\Upsilon,\Pi)}$ over Υ defined by Π. In particular, we denote both by T.
In this paper we focus on *infinite* runs of relational transducers over a *fixed* domain (in contrast to [3]; see also Remark 1 (b) below).

Runs. Let T be an ASM transducer over Υ. A *database* $\mathcal{D}$ appropriate for T is a finite structure over Υ_{db}. An *input sequence* $\bar{I}$ appropriate for T and $\mathcal{D}$ is an infinite sequence $(\mathcal{I}_i)_{i\in\omega}$ of finite structures over Υ_{in} where each $\mathcal{I}_i$ has the same universe as $\mathcal{D}$.
Let $\mathcal{D}$ be a database and $\bar{I}$ an input sequence, both appropriate for T (and $\mathcal{D}$). The *run* ρ of T on $\mathcal{D}$ and $\bar{I}$ is an infinite sequence $(\mathcal{S}_i)_{i\in\omega}$ of states over Υ uniquely determined by the following conditions. For every $i \in \omega$,

- $\mathcal{S}_i|_{\text{in}} = \mathcal{I}_i$,
- $\mathcal{S}_i|_{\text{db}} = \mathcal{D}$, and
- $\mathcal{S}_i|_{\text{mem,out}} = (D, \varnothing)$ if $i = 0$ (i.e., all memory and output relations are empty in the initial state $\mathcal{S}_0$); otherwise $\mathcal{S}_i|_{\text{mem,out}} = T(\mathcal{S}_{i-1})$.

The *reduct* of ρ to some vocabulary $\Upsilon' \subseteq \Upsilon$, denoted $\rho|\Upsilon'$, is the infinite sequence $(\mathcal{S}_i|\Upsilon')_{i\in\omega}$. Again, we write $\rho|_{\text{db}}$ instead of $\rho|\Upsilon_{\text{db}}$, $\rho|_{\text{db,mem}}$ instead of $\rho|(\Upsilon_{\text{db}} \cup \Upsilon_{\text{mem}})$, etc.
The *output* and *log* produced during ρ are the infinite sequences $\rho|_{\text{out}}$ and $\rho|_{\text{log}}$, respectively. The output of an ASM transducer can be seen as a write-only data stream. Each package in that stream consists of the output relations produced during one computation step of the transducer. Logs have been introduced in [3] to capture the semantically significant input-output behavior of relational transducers (see the finite log validation problem in the next section).

Example 1. Below we define an ASM transducer T_{supp} specifying the business model of a supplier whose customers can order products, are billed for them, eventually pay, and finally receive delivery. It is inspired by an example in [3]. For readability, we display the program of T_{supp} in a slightly relaxed syntax, using nested rules (with the obvious meaning) and rules of the form $(\texttt{if } \varphi(\bar{x}, \bar{y}) \texttt{ then } (\neg)R(\bar{x}))$, where $\{\bar{x}\} \subsetneq \{\bar{x}, \bar{y}\} = \text{free}(\varphi)$. Formally, a rule of the latter form has to be replaced with $(\texttt{if } \exists\bar{y}\varphi(\bar{x}, \bar{y}) \texttt{ then } (\neg)R(\bar{x}))$.

transducer T_{supp}:

relations:
  input: $\mathit{Order}, \mathit{Pay}$
  database: $\mathit{Price}, \mathit{Available}$
  memory: $\mathit{PastOrder}$
  output: $\mathit{SendBill}, \mathit{Deliver}, \mathit{RejectPay}, \mathit{RejectOrder}$
  log: $\mathit{Pay}, \mathit{SendBill}, \mathit{Deliver}$

memory rules:
  if $\mathit{Order}(x) \wedge \mathit{Available}(x) \wedge \neg \mathit{PastOrder}(x)$ then
    $\mathit{PastOrder}(x)$
  if $\mathit{PastOrder}(x) \wedge \mathit{Pay}(x, y) \wedge \mathit{Price}(x, y)$ then
    $\neg \mathit{PastOrder}(x)$

output rules:
  if $\mathit{Order}(x) \wedge \mathit{Available}(x)$ then
    if $\neg \mathit{PastOrder}(x) \wedge \mathit{Price}(x, y)$ then
      $\mathit{SendBill}(x, y)$
    if $\mathit{PastOrder}(x)$ then
      $\mathit{RejectOrder}(x)$
  if $\mathit{Pay}(x, y)$ then
    if $\mathit{PastOrder}(x) \wedge \mathit{Price}(x, y)$ then
      $\mathit{Deliver}(x)$
    else
      $\mathit{RejectPay}(x, y)$

The following table sketches the input and output components of an initial segment of a run ρ of T_{supp}:

ρ	$\mathcal{S}_0$	$\mathcal{S}_1$	$\mathcal{S}_2$	...
in	$\mathit{Order}(a)$	$\mathit{Order}(b)$	$\mathit{Pay}(a, 5)$	
out		$\mathit{SendBill}(a, 5)$	$\mathit{SendBill}(b, 8)$	

...	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$
	$\mathit{Order}(a)$ $\mathit{Order}(b)$	$\mathit{Pay}(a, 5)$	
	$\mathit{Deliver}(a)$	$\mathit{SendBill}(a, 5)$ $\mathit{RejectOrder}(b)$	$\mathit{Deliver}(a)$

Remark 1. (a) ASM transducers are more expressive than the Spocus transducers introduced in [3]. On the one hand, every Spocus transducer program Π can easily be converted into an equivalent ASM transducer program by replacing every rule $(\gamma \leftarrow \varphi)$ in Π with (if φ then γ). On the other hand, T_{supp} in the above example is not expressible by means of Spocus transducers. Observe that the memory of T_{supp} is *not* cumulative, which enables customers to order one and the same item several times (in principle, infinitely often) during a single session. A similar behavior cannot be modeled by means of Spocus transducers, because their memory only accumulates all previous input.
(b) In [3], the authors consider *finite* runs of relational transducers over a possibly *expanding* domain. If the semantics of the guards of ASM transducer rules and the semantics of temporal formulas specifying properties of runs (see the next section) is adjusted appropriately, then our results in Section 5 should also hold in this setting.
(c) Readers familiar with the ASM computation model may have noticed that syntax and semantics of ASM transducer programs do not entirely conform to the official ASM framework [12]. This deviation from the standard definitions only serves to improve readability and allows us to simplify the presentation of technicalities. It is however not essential for the (general) ASM transducer model presented here.

Computational power of ASM transducers. We compare the computational power of ASM transducers with that of Turing machines. Since Turing machines do not interact with their environment during run-time, we restrict ourselves to ASM transducers that take a database as input and perform their computation without reading any further input.
Fix a transducer vocabulary Υ with $\Upsilon_{\text{in}} = \varnothing$ and boolean symbols *halt*, *accept* $\in \Upsilon_{\text{out}}$. Let Q be a *boolean query* over Υ_{db} (i.e., a class of finite structures over Υ_{db} closed under isomorphisms [2; 7]). We say that an ASM transducer T over Υ *computes* Q if for every database $\mathcal{D}$ appropriate for T:

1. T on $\mathcal{D}$ reaches a halting state, and

2. the first halting state reached by T on $\mathcal{D}$ is accepting iff $\mathcal{D} \in Q$.

Using well-known results from finite model theory (see, e.g., [2; 7]) the proof of the next lemma is an easy exercise.

LEMMA 1. *A boolean query is expressible in partial fixed-point logic* (FO+PFP) *(or, equivalently, in* datalog$^{\neg\neg}$*) iff it is computable by an ASM transducer. On ordered databases ASM transducers compute precisely the class of* PSPACE-*computable boolean queries.*

3. VERIFICATION PROBLEMS

In this paper we study the following three verification problems. The first two problems concern the correctness and customization of relational transducers. The third problem is related to the validation of transactions carried out by means of such transducers.

(1) **Verifying temporal properties** of relational transducers (which subsumes the problem of *goal reachability* in [3]).

(2) **Deciding log equivalence** of two relational transducers.

(3) **Finite log validation**.

Next, we formally define these problems, partly by recalling them from [3], partly by generalizing or redefining the problems in [3].

Verifying temporal properties. This problem concerns the correctness of relational transducers and thus typically emerges while designing transducers. To specify requirements on relational transducers we propose *first-order temporal logic* (FTL) [8; 1], a formalism suitable to express properties of runs of transducers. For the readers convenience we recall the definition of FTL below and use the opportunity to introduce the fragment UT of FTL, which will play a prominent role in Section 5.

Definition 3. First-order temporal logic (FTL) is obtained from FO by means of the following additional formula-formation rule:

(T) If φ and ψ are formulas, then $\mathbf{X}\varphi$ and $\varphi\mathbf{U}\psi$ are also formulas.

The *free* and *bound variables* of FTL formulas are defined in the obvious way.
Let T denote the closure of the set of FO formulas under negation, disjunction, and the above rule **(T)**. The *universal closure* of T, denoted UT, is the set of FTL formulas of the form $\forall\bar{x}\varphi$ with $\varphi \in \mathrm{T}$ and $\mathrm{free}(\varphi) \subseteq \{\bar{x}\}$.

Intuitively, a formula of the form $\mathbf{X}\varphi$ (resp. $\varphi\mathbf{U}\psi$) expresses that "φ holds in the next state" (resp. that "φ holds until ψ holds"). We sketch the formal definition of the semantics of FTL formulas only with respect to runs of ASM transducers. Let Υ be a transducer vocabulary, φ a FTL formula over Υ with $\mathrm{free}(\varphi) = \{\bar{x}\}$, and $\rho = (\mathcal{S}_i)_{i\in\omega}$ a run of an ASM transducer over Υ. Simultaneously for every $i \in \omega$ and all interpretations $\bar{a}$ of the variables $\bar{x}$ (chosen from the universe of $\mathcal{S}_0$), define the *satisfaction relation* $(\rho, i, \bar{a}) \models \varphi$ by induction on the construction of φ:

$$
\begin{array}{lll}
(\rho,i,\bar{a}) \models \varphi & :\Leftrightarrow & \mathcal{S}_i \models \varphi[\bar{a}], \text{ if } \varphi \text{ is atomic} \\
(\rho,i,\bar{a}) \models \mathbf{X}\varphi & :\Leftrightarrow & (\rho,i+1,\bar{a}) \models \varphi \\
(\rho,i,\bar{a}) \models \varphi\mathbf{U}\psi & :\Leftrightarrow & \text{there is a } j \geq i \text{ such that} \\
& & (\rho,j,\bar{a}) \models \psi \text{ and for all } k, \\
& & i \leq k < j,\ (\rho,k,\bar{a}) \models \varphi
\end{array}
$$

All other cases are defined as usual. We say that ρ *satisfies* a FTL sentence φ over Υ if $(\rho, 0) \models \varphi$.
The problem of verifying temporal properties of ASM transducers can now be formalized as a decision problem. Consider a class C of ASM transducers and a fragment F of FTL. *Verifying C-transducers against F-specifications* means solving the following problem:

VERIFY(C, F): Given an ASM transducer $T \in C$ and a sentence $\varphi \in F$ over the vocabulary of T, decide whether every run of T satisfies φ.

Example 2. Recall from Example 1 the ASM transducer T_{supp} specifying the business model of a supplier. For the supplier it might be desirable to enable customers to pay an ordered item and to simultaneously reorder the same item. We can specify this requirement on T_{supp} by means of the UT formula displayed below. Intuitively, the formula expresses that an order of an item a will be rejected in the next step only if a is already ordered and is not correctly paid in the current input.

$$
\begin{aligned}
&\forall x\mathbf{G}\big[\mathbf{X}\mathit{RejectOrder}(x) \rightarrow \\
&\qquad \big(\mathit{PastOrder}(x) \wedge \neg\exists y(\mathit{Pay}(x,y) \wedge \mathit{Price}(x,y))\big)\big],
\end{aligned}
$$

where $\mathbf{G}\psi$ abbreviates $\neg(\mathit{true}\,\mathbf{U}\neg\psi)$. Let φ denote the above UT formula. Does T_{supp} meet the specification φ, i.e., is $(T_{\text{supp}}, \varphi) \in$ VERIFY(ASM-T, UT)? It is not hard to see that the answer is no.
However, one can upgrade T_{supp} to an ASM transducer T^{+}_{supp} such that $(T^{+}_{\text{supp}}, \varphi) \in$ VERIFY (ASM-T, UT). For instance, remove the conjunct $\neg\mathit{PastOrder}(x)$ from the guard of the first memory rule of T_{supp} and instead add $\neg\mathit{Order}(x)$ as a new conjunct to the guard of the second memory rule. Furthermore, replace the first (nested) output rule of T_{supp} with the following rule:

$$
\begin{array}{l}
\texttt{if } \mathit{Order}(x) \wedge \mathit{Available}(x) \texttt{ then} \\
\quad \texttt{if } \neg\mathit{PastOrder}(x) \wedge \mathit{Price}(x,y) \texttt{ then} \\
\qquad \mathit{SendBill}(x,y) \\
\quad \texttt{if } \mathit{PastOrder}(x) \texttt{ then} \\
\qquad \texttt{if } \mathit{Pay}(x,y) \wedge \mathit{Price}(x,y) \texttt{ then} \\
\qquad\quad \mathit{SendBill}(x,y) \\
\qquad \texttt{if } \neg\exists y(\mathit{Pay}(x,y) \wedge \mathit{Price}(x,y)) \texttt{ then} \\
\qquad\quad \mathit{RejectOrder}(x)
\end{array}
$$

Note that we cannot avoid the introduction of a FO quantifier in the above rule if the supplier's business model requires immediate response on any request, i.e., does not allow the transducer to postpone replies to a later point of time.

Remark 2. Although FTL does not offer past-tense temporal operators, one can mimic such operators using additional memory relations to record the history of ongoing transactions.

Finite log validation. This problem concerns fraud detection and arises if, e.g., for efficiency reasons, the relational transducer of a supplier is allowed to run on remote customer sites. In such a distributed scenario it can be vitally important for the supplier to verify that all transactions carried out on remote sites actually conform to the supplier's business model. That is, the supplier needs to check whether certain transactions are valid in the sense that—in principle—they could have been carried out by means of the transducer originally distributed.
Since in many applications not all data exchanged during a transaction is important for the supplier (e.g., inquiries about prices), the semantically significant input and output relations of a relational transducer are specified as *log relations* [3]. A record of a transaction is now a finite sequence of collections of log relations. Each collection in the sequence contains the log relations at a particular time of the transaction.
Consider an ASM transducer T over Υ, and a finite sequence $\bar{L} = (\mathcal{L}_0, \ldots, \mathcal{L}_n)$ of finite structures over $\Upsilon_{\log}$. $\bar{L}$ is called a *finite log* of T if there exists a run ρ of T such that $\bar{L}$ is an initial segment of $\rho|_{\log}$. The *finite log validation problem* for a class C of ASM transducers is the following decision problem:

FIN-LOG-VAL(C): Given an ASM transducer $T \in C$ and a finite sequence $\bar{L}$ of finite structures over the log vocabulary of T, decide whether $\bar{L}$ is a finite log of T.

Deciding log equivalence. The main motivation for considering this problem is *customization* of relational transducers. To enhance competitiveness a supplier may want to allow some customers to modify or upgrade the supplier's transducer for their convenience or to conform to their own business models. This raises the question whether a customized transducer still conforms to the original one.
Consider two ASM transducers T_1 and T_2 over the same vocabulary. We call T_1 and T_2 *log equivalent* if for every run ρ_1 of T_1 and every run ρ_2 of T_2 the following implication holds:

$$\big(\rho_1|_{\text{db,in}} = \rho_2|_{\text{db,in}}\big) \quad\Rightarrow\quad \big(\rho_1|_{\log} = \rho_2|_{\log}\big).$$

In other words, whenever T_1 and T_2 run on the same database and are fed with the same input relations, than T_1 and T_2 produce the same logs. The corresponding decision problem is defined as follows:

LOG-EQ(C): Given two ASM transducers $T_1, T_2 \in C$ over the same vocabulary, decide whether T_1 and T_2 are log equivalent.

Example 3. Verify that T_{supp} and T^{+}_{supp} in Examples 1 and 2 are not log equivalent. Observe also that adding to the program of an ASM transducer new output rules that do not affect log-output relations obviously preserves log equivalence (see also [3]). For instance, a customer may propose to add to the program of T_{supp} the following new output rule:

$$\texttt{if } \mathit{PendingBills} \wedge \mathit{PastOrder}(x) \\ \wedge \mathit{Price}(x,y) \wedge \neg \mathit{Pay}(x,y) \texttt{ then} \\ \mathit{Rebill}(x,y)$$

where *PendingBills* is a new input relation and *Rebill* a new output relation. If *Rebill* is not marked as log, then the obtained ASM transducer is clearly log equivalence to T_{supp}.

4. NATURAL RESTRICTIONS

Decidability and complexity of the three verification problems VERIFY(C,F), LOG-EQ(C), and FIN-LOG-VAL(C) obviously depend on the choice of the class $C \subseteq$ ASM-T (and the fragment $F \subseteq$ FTL). For instance, if we let $C :=$ ASM-T (and assume that F contains $\mathbf{X}\mathit{accept}$), then we will not be able to solve any of these problems, as they are undecidable by Trakhtenbrot's theorem (see, e.g., [7]). This leaves us with two options for how to proceed: we may impose restrictions on:

1. ASM transducers, or
2. the verification problems themselves.

The first approach was successfully pursued in [3] and led to the Spocus transducer model. Here we follow the second approach and attempt to solve simplified variants of VERIFY, LOG-EQ, and FIN-LOG-VAL for ASM transducers. We consider the following two types of simplifications (that are indeed restrictions):

- **The database** on which a relational transducer is supposed to run **is provided** and fixed during verification.
- **The maximal input flow** (see Definition 4 below) to which a relational transducer is exposed **is a priori limited**.

Both restrictions are made precise next.

Providing the database. This restriction is applicable if the database of a relational transducer changes so rarely that it becomes feasible to adjust verification to the currently valid database. As an example, consider a relational transducer T running on a notebook computer of a salesperson. The database of T may contain the supplier's catalog and may be stored on some local drive of the notebook computer. In view of the fact that T will run on this particular database only, we can as well fix the database while verifying T.

We denote by VERIFY$^{\text{db}}$ the variant of VERIFY where only runs *on a given database* are considered. Formally, VERIFY$^{\text{db}}$ is defined as follows:

VERIFY$^{\text{db}}(C,F)$: Given an ASM transducer $T \in C$, a sentence $\varphi \in F$ over the vocabulary of T, *and a database $\mathcal{D}$ appropriate for T*, decide whether every run of T *on $\mathcal{D}$* satisfies φ.

The corresponding variants of LOG-EQ and FIN-LOG-VAL, denoted LOG-EQ$^{\text{db}}$ and FIN-LOG-VAL$^{\text{db}}$, respectively, are defined similarly.

Remark 3. For many applications it suffices to solve FIN-LOG-VAL$^{\text{db}}$ instead of FIN-LOG-VAL. Clearly, every transaction by means of a relational transducer is performed with respect to *concrete* database. If we assume that the record of each transaction includes a time stamp, and that the database used during the transaction is still accessible during validation time, then deciding FIN-LOG-VAL$^{\text{db}}$ in fact suffices. This especially holds if all transactions are validated during run time.

However, for applications where the database changes frequently checking temporal properties or deciding log equivalence with respect to a fixed database is impractical or entirely useless. In such cases we propose the following restriction.

Limiting the maximal input flow. This restriction is motivated by the observation that the amount of input data forwarded to a relational transducer in one computation step is usually limited by the environment of the transducer (recall our discussion in the introduction). Hence, while verifying a relational transducer T, we may focus on runs of T where in every state the number of tuples in every input relation is a priori bounded by some natural number N (depending only on T's environment). This justifies the following definition.

Definition 4. Let $\rho = (\mathcal{S}_i)_{i\in\omega}$ be a run of an ASM transducer over Υ. The *maximal input flow* of ρ is the maximal natural number in $\{|R^{\mathcal{S}_i}| : R \in \Upsilon_{\text{in}}, i \in \omega\}$, where $|R^{\mathcal{S}_i}|$ denotes the cardinality of the input relation $R^{\mathcal{S}_i}$ (seen as a set of tuples).

Remark 4. An alternative definition of the maximal input flow could limit in every state of a run the total number of tuples in all input relations. For technical reasons we prefer the above definition.

Let N be a natural number. We denote by VERIFY$^{\text{in} \le N}$ the variant of VERIFY where only runs with maximal input flow $\le N$ are verified. Formally, VERIFY$^{\text{in} \le N}$ is defined as follows:

VERIFY$^{\text{in} \le N}(C,F)$: Given an ASM transducer $T \in C$ and a sentence $\varphi \in F$ over the vocabulary of T, decide whether every run of T *with maximal input flow $\le N$* satisfies φ.

The corresponding variants of LOG-EQ and FIN-LOG-VAL, denoted LOG-EQ$^{\text{in} \le N}$ and FIN-LOG-VAL$^{\text{in} \le N}$, respectively, are defined along the same line.

The next lemma indicates that imposing an upper bound on the maximal input flow can also be regarded as a restriction of ASM transducers. Let T be an ASM transducer whose vocabulary contains the boolean output symbol *error*. We call a run of T *error-free* if *error* does not hold in any state of the run.

LEMMA 2. *Every ASM transducer (whose vocabulary does not contain the boolean output symbol error) can be modified so that the error-free runs of the obtained ASM transducer are precisely the runs with maximal input flow $\leq N$ of the original transducer.*

Although imposing an upper bound on the maximal input flow should be no serious obstacle for most electronic commerce applications, this restriction is not strong enough to entail verifiability of all ASM transducers (see the next section).
Equipped with the terminology established so far, we can now state our main results.

5. VERIFIABILITY RESULTS

We first present our results concerning the verifiability of ASM transducers which are supposed to run on a given database only.
Let VERIFY_m denote the restriction of VERIFY to instances where only relation symbols of arity $\leq m$ occur. The corresponding restrictions of LOG-EQ and FIN-LOG-VAL are denoted by LOG-EQ_m and FIN-LOG-VAL_m, respectively.

THEOREM 1. *The following problems are* PSPACE*-complete for any natural number m:*

(1) $\text{VERIFY}_m^{\text{db}}(\text{ASM-T}, \text{UT})$.

(2) $\text{LOG-EQ}_m^{\text{db}}(\text{ASM-T})$.

(3) $\text{FIN-LOG-VAL}_m^{\text{db}}(\text{ASM-T})$.

In other words, if the maximal arity of the employed relations is a priori bounded, then

- *verifying temporal properties (expressible in the fragment* $\text{UT} \subseteq \text{FTL}$*),*
- *deciding log equivalence, and*
- *finite log validation*

are PSPACE*-complete problems for ASM transducers which are supposed to run on a given database only.*

The proof of Theorem 1 as well as the proofs of all subsequent results are sketched in the appendix. Problems (1)–(3) in the theorem are already PSPACE-hard for a fixed transducer vocabulary and a fixed database. All three problems remain in PSPACE if the arities of database and memory relations are a priori bounded, the arities of input and output relations however not. If we lift this limitation, then verification becomes more expensive.

COROLLARY 1. *The following problems are in* EXPSPACE *for any natural number $N \geq 1$:*

(1') $\text{VERIFY}^{\text{db,in} \leq N}(\text{ASM-T}, \text{UT})$.

(2') $\text{LOG-EQ}^{\text{db,in} \leq N}(\text{ASM-T})$.

(3') $\text{FIN-LOG-VAL}^{\text{db,in} \leq N}(\text{ASM-T})$.

The first two problems are even EXPSPACE*-hard.*

As pointed out in the last section, Theorem 1 provides a sufficiently general solution for applications where the design and verification of ASM transducers can be adjusted to specific databases. Since it often suffices to solve $\text{FIN-LOG-VAL}^{\text{db}}$ instead of FIN-LOG-VAL (recall Remark 3), the theorem settles the finite log validation problem for many other applications as well.
We next turn to ASM transducers that need to be verified for *all* databases because their database changes frequently or may even change during run time (see Remark 6 below).

ASM transducers with input-bounded quantification. If in problem (1') in Corollary 1 the database is not provided we immediately encounter undecidability. This is basically due to the expressive power of (unrestricted) first-order quantification in the guards of ASM transducer rules. To still obtain decidability, we propose the usage of a restricted type of first-order quantification in guards, called *input-bounded quantification*, which is specially tailored for input-driven devices like relational machines.
The main idea behind input-bounded quantification is to restrict the range of first-order quantifiers to the active domain of the current input. As an example of this type of quantification, consider the following output rule, taken from T^+_{supp} in Example 2:

$$\texttt{if } \mathit{Order}(x) \wedge \mathit{Available}(x) \wedge \mathit{PastOrder}(x) \wedge \neg\exists y(\mathit{Pay}(x,y) \wedge \mathit{Price}(x,y)) \texttt{ then } \mathit{RejectOrder}(x)$$

In this rule, the quantified variable y is 'guarded' by the input relation *Pay*. The range of y can safely be restricted to the active domain of the current input.

Definition 5. Let Υ be a transducer vocabulary . An *input atom* (resp. *memory*, *output atom*) is an atomic formula of the form $R(\bar{t})$ with R in Υ_{in} (resp. Υ_{mem}, Υ_{out}).
The *input-bounded fragment* of FO, denoted FO^{I}, is obtained from FO by replacing the formula-formation rule for (unbounded) quantification with the following rule for *input-bounded quantification*:

(IBQ) If $\bar{x}$ is a tuple of variables, α an input atom with $\{\bar{x}\} \subseteq \text{free}(\alpha)$, and φ a formula such that for every memory and output atom β occurring in φ, $\text{free}(\beta) \cap \{\bar{x}\} = \varnothing$, then $\exists\bar{x}(\alpha \wedge \varphi)$ and $\forall\bar{x}(\alpha \rightarrow \varphi)$ are formulas.

An *ASM transducer with input-bounded quantification* (or *ASM^{I} transducer* for short) is an ASM transducer in whose program all rules are guarded by FO^{I} formulas. ASM^{I}-T denotes the class of ASM^{I} transducers.
Let T^{I} denote the closure of the set of FO^{I} formulas under negation, disjunction, and rule **(T)** (see Definition 3). The *universal closure* of T^{I}, denoted UT^{I}, is the set of FTL formulas of the form $\forall\bar{x}\varphi$ with $\varphi \in \text{T}^{\text{I}}$ and $\text{free}(\varphi) \subseteq \{\bar{x}\}$.

To see an example of an ASM^{I} transducer and a UT^{I} specification, recall T^+_{supp} and φ in Example 2. Also, verify that Lemma 2 remains true for ASM^{I} transducers.

Remark 5. ASM^{I} and Spocus transducers are incomparable in the following sense. There exists a run of an ASM^{I} (resp. Spocus) transducer such that no Spocus (resp. ASM^{I}) transducer can produce this run. (Recall Remark 1 (a) and note that Spocus transducers allow unrestricted projections in the guards of their output rules.)

We are now in the position to state our results concerning the verifiability of ASM transducers that need to be verified for all databases.

THEOREM 2. *The following problems are* PSPACE-*complete for any natural numbers* $m \geq 1$ *and* $N \geq 2$*:*

(4) $\textsc{verify}_m^{\mathrm{in}\,\leq\,N}(\mathrm{ASM^I\text{-}T}, \mathrm{UT^I})$.

(5) $\textsc{log-eq}_m^{\mathrm{in}\,\leq\,N}(\mathrm{ASM^I\text{-}T})$.

In other words, if the maximal arity of the employed relations and the maximal input flow are a priori bounded, then

- *verifying temporal properties (expressible in the fragment* $\mathrm{UT^I} \subseteq \mathrm{FTL}$*), and*
- *deciding log equivalence*

are PSPACE-*complete problems for ASM*^{I} *transducers.*

As before, we observe an exponential blow-up of the space complexity if there is no fixed upper bound on the arities of database and memory relations.

COROLLARY 2. *The following problems are* EXPSPACE-*complete for any natural number* $N \geq 2$*:*

(4') $\textsc{verify}^{\mathrm{in}\,\leq\,N}(\mathrm{ASM^I\text{-}T}, \mathrm{UT^I})$.

(5') $\textsc{log-eq}^{\mathrm{in}\,\leq\,N}(\mathrm{ASM^I\text{-}T})$.

Remark 6. Until now we have only considered relational transducers whose database does not change during run time. To verify an $\mathrm{ASM^I}$ transducer T with an active database, i.e., a database that can be updated during a run of T, one may proceed as follows. Suppose that $R_1, \ldots, R_n$ are the active database relations of T. Modify the program of T so that, in the first step, T copies each R_i to a new (still empty) memory relation R_i', and then, in all subsequent steps, uses R_i' instead of R_i. Every update of an R_i is now treated as input and redirected to R_i'.
There are, however, three disadvantages to this solution. Firstly, database updates may jam ordinary inputs from customers, and secondly, the number of database updates in one step is bounded (due to a necessary bound on the maximal input flow). Thirdly, by definition of input-bounded quantification we are not allowed to quantifier inside active database relations (since they are now treated as memory relations).

6. CONCLUSION

Taking the framework provided in [3] as starting point, we have investigated the verifiability of transaction protocols typically occurring in the context of electronic commerce applications. We have introduced a class of powerful relational transducers to specify such transaction protocols and showed that verification problems of practical importance can be solved for these transducers.
A still open problem is the *verification of systems of interacting relational transducers* [3]. We are optimistic that our investigations in this paper can serve as a basis for future research in this direction. Indeed, our results concerning the decidability of log equivalence is proved by means of a construction where two transducers interact by sharing part of their current state.

Another open problem concerns a class of *efficiently verifiable relational transducer.* PSPACE complexity is likely to be too expensive for large-scale applications, especially when it comes to verification with respect to a given database. The question is whether there are further restrictions of ASM transducers (with input-bounded quantification) such that the corresponding verification problems are in PTIME.

Acknowledgements. I am grateful to Erich Grädel for drawing my attention to the subject of this paper and for useful comments on an earlier draft. I would also like to thank Serge Abiteboul, Eric Rosen, and Victor Vianu for helpful comments and suggestions.

7. REFERENCES

[1] S. Abiteboul, L. Herr, and J. Van den Bussche. Temporal Versus First-Order Logic to Query Temporal Databases. In *Proceedings of 15th ACM Symposium on Principles of Database Systems (PODS '96)*, pages 49–57. ACM Press, 1996.

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

[3] S. Abiteboul, V. Vianu, B. Fordham, and Y. Yesha. Relational Transducers for Electronic Commerce. In *Proceedings of 17th ACM Symposium on Principles of Database Systems (PODS '98)*, pages 179–187. ACM Press, 1998.

[4] N. Adam and Y. Yesha. Electronic Commerce: An Overview. In N. Adam and Y. Yesha, editors, *Electronic Commerce*, volume 1028 of *Lecture Notes in Computer Science*, pages 5–12. Springer-Verlag, 1996.

[5] E. Börger and J. Huggins. Abstract State Machines 1988–1998: Commented ASM Bibliography. *Bulletin of the EATCS*, 64:105–127, February 1998.

[6] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. In *Proceedings of 12th Annual IEEE Conference on Computational Complexity (CCC '97)*, pages 82–101, 1997.

[7] H. D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer-Verlag, 1995.

[8] E. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–11072. Elsevier Science Publishers B.V., 1990.

[9] B. Fordham, S. Abiteboul, and Y. Yesha. Evolving Databases: An Application to Electronic Commerce. In *Proceedings of the International Database Engineering and Applications Symposium (IDEAS)*, August 1997.

[10] G. Gottlob, G. Kappel, and M. Schrefl. Semantics of Object-Oriented Data Models – The Evolving Algebra Approach. In J. W. Schmidt and A. A. Stogny, editors, *Next Generation Information Technology*, volume 504 of *Lecture Notes in Computer Science*, pages 144–160. Springer-Verlag, 1991.

[11] Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.

[12] Y. Gurevich. May 1997 Draft of the ASM Guide. Technical Report CSE-TR-336-97, University of Michigan, May 1997.

[13] N. Immerman and M. Vardi. Model Checking and Transitive Closure Logic. In *Proceedings of 9th International Conference on Computer-Aided Verification (CAV '97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 291–302. Springer-Verlag, 1997.

[14] A. Levy, I. Mumick, Y. Sagiv, and O. Shmueli. Equivalence, Query-Reachability, and Satisfiability in Datalog Extensions. In *Proceedings of 12th ACM Symposium on Principles of Database Systems (PODS '93)*, pages 109–122, 1993.

[15] P. Picouet and V. Vianu. Semantics and Expressiveness Issues in Active Databases. *Journal of Computer and System Sciences*, 57(3):325–355, 1998.

[16] E. Rosen. An existential fragment of second order logic. *Archive for Mathematical Logic*, 38:217–234, 1999.

[17] M. Spielmann. *Some Notes on Abstract State Machines.* PhD thesis, RWTH Aachen. In preparation.

[18] M. Spielmann. Automatic Verification of Abstract State Machines. In *Proceedings of 11th International Conference on Computer-Aided Verification (CAV '99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 431–442. Springer-Verlag, 1999.

[19] M. Vardi. Complexity of relational query languages. In *Proceedings of 14th ACM Symposium on Theory of Computing (STOC '82)*, pages 137–146, 1982.

[20] M. Vardi. On the Complexity of Bounded-Variable Queries. In *Proceedings of 14th ACM Symposium on Principles of Database Systems (PODS '95)*, pages 266–276. ACM Press, 1995.

APPENDIX

We sketch the proofs of our main results. Subsequently, we refer to the verification problems in Theorems 1 and 2, and Corollaries 1 and 2 by their respective (primed) numbers.

PROPOSITION 1. *(i)* FIN-LOG-VAL$^{\mathrm{db}}$(ASM-T) *is polynomial-time reducible to the complement of* VERIFY$^{\mathrm{db}}$(ASM-T, UT). *(ii)* LOG-EQ(ASM-T) *is polynomial-time reducible to* VERIFY(ASM-T, UT).

The proposition remains true if ASM-T and UT is replaced with ASM$^{\mathrm{I}}$-T and UT$^{\mathrm{I}}$, respectively. It also holds for various combinations of restrictions of VERIFY, LOG-EQ, and FIN-LOG-VAL. In particular, one can show that problems (2) and (3) reduce to problem (1), problems (2') and (3') reduce to problem (1'), problem (5) reduces to problem (4), and problem (5') reduces to problem (4'). Hence, to prove our containment assertions it suffices to consider problems (1), (1'), (4) and (4'). For hardness we may concentrate on problems (2), (2'), (3), (5), and (5').

Hardness of problems (2), (3) and (5) can be proved by reduction of the PSPACE-complete satisfiability problem for quantified boolean formulas. Hardness of problems (2') and (5') follows by reduction of the EXPSPACE-complete model checking problem for partial fixed-point logic (FO+PFP). The complexity of the latter problem is also known as the combined complexity of the query language (FO+PFP) (see [20; 19; 6]).
The remainder of the appendix is devoted to the proofs of our containment assertions. We will particularly be interested in the following problem:

RUN-SAT(C, F): Given an ASM transducer $T \in C$ and a sentence $\varphi \in F$ over the vocabulary of T, decide whether there *exists* a run of T satisfying φ.

Obviously, $(T, \varphi) \in$ VERIFY(C, F) iff $(T, \neg\varphi) \notin$ RUN-SAT$(C, \neg F)$, where $\neg F$ denotes the set of negated F-formulas.
Recall the definition of the fragment T of FTL (see Definition 3). For any complexity class K closed under complementation and polynomial-time reductions, the above equivalence implies VERIFY$(C, \mathrm{UT}) \in K$ iff RUN-SAT$(C, \mathrm{T}) \in K$, and VERIFY$(C, \mathrm{UT}^{\mathrm{I}}) \in K$ iff RUN-SAT$(C, \mathrm{T}^{\mathrm{I}}) \in K$, presuming that database vocabularies may now contain constant symbols. This observation will justify to consider RUN-SAT in place of VERIFY in the upcoming construction that shows containment of problems (1), (1'), (4), and (4').
We present the construction in three steps. The first step outlines the general direction of the construction in form of a polynomial-time reduction of RUN-SAT(ASM-T, T) to FIN-SAT(FO+TC), i.e, the finite satisfiability problem for transitive closure logic [7]. In the second and third step, we then show that suitable refinements of the reduction imply the desired containment assertions.

Step 1: Reduction of RUN-SAT(ASM-T, T). We call a run $\rho = (\mathcal{S}_i)_{i\in\omega}$ of an ASM transducer *periodic* if there exist $s, p \in \omega$ with $p \geq 1$ such that $\mathcal{S}_i = \mathcal{S}_{i+p}$ for every $i \geq s$.

LEMMA 3. *Let T be an ASM transducer and φ a* T *sentence over the vocabulary of T. If there exists a run of T satisfying φ, then there also exists a periodic run of T satisfying φ.*

Let Υ be a transducer vocabulary. A (finite or infinite) sequence $\sigma = (\mathcal{S}_i)_{i\in\kappa}$ of states over Υ is called *consistent* if $\mathcal{S}_i|_{\mathrm{db}} = \mathcal{S}_j|_{\mathrm{db}}$ for all $i, j \in \kappa$. We encode finite consistent sequences (presumably representing periodic runs) as finite structures. To this end, suppose that $\sigma = (\mathcal{S}_0, \dots, \mathcal{S}_q)$ is consistent. Let D be the domain of $\mathcal{S}_0|_{\mathrm{db}}$ and set $I = \{0, \dots, q\}$. W.l.o.g., we may assume that D and I are disjoint. Obtain Υ^+ from Υ by (i) adding to Υ two new set symbols, say, D' and I', and (ii) increasing the arity of every relation symbol in $\Upsilon_{\mathrm{in}} \cup \Upsilon_{\mathrm{mem}} \cup \Upsilon_{\mathrm{out}}$ by one. Now define the finite structure $\mathcal{A}_\sigma$ over Υ^+ (encoding σ) as follows:

- Let the universe of $\mathcal{A}_\sigma$ be $D \cup I$.
- Let D' and I' be interpreted as D and I, respectively.
- Let every $R \in \Upsilon_{\mathrm{db}}$ be interpreted as in $\mathcal{S}_0|_{\mathrm{db}}$.
- For every k-ary $R \in \Upsilon_{\mathrm{in}} \cup \Upsilon_{\mathrm{mem}} \cup \Upsilon_{\mathrm{out}}$, $\bar{a} \in D^k$, and $i \in I$, let $(\bar{a}, i) \in R^{\mathcal{A}_\sigma} :\Leftrightarrow \bar{a} \in R^{\mathcal{S}_i}$.

To ease notation we may subsequently use one and same letter to denote both a free variable of a formula and an interpretation of that variable. The intended meaning will be clear from the context.

PROPOSITION 2. *Let Υ and Υ^+ be as above and T an ASM transducer over Υ. From T one can obtain in polynomial time a formula $\chi_T(i,i') \in \mathrm{FO}(\Upsilon^+)$ with free$(\chi_T) = \{i,i'\}$ such that, whenever σ and $\mathcal{A}_\sigma$ are as above and $i,i' \in \{0,\dots,q\}$, then*

$$\mathcal{A}_\sigma \models \chi_T[i,i'] \quad \Leftrightarrow \quad T(\mathcal{S}_i) = \mathcal{S}_{i'}|_{\mathrm{mem,out}}.$$

As a first step toward a proof of our containment assertions, we display a polynomial-time reduction of RUN-SAT(ASM-T, T) to FIN-SAT(FO+TC). This reduction will serve as a guideline in steps two and three later on.

Consider an instance (T,φ) of RUN-SAT(ASM-T, T) with T and φ over Υ. We are going to define a sentence $\chi_{T,\varphi} \in (\mathrm{FO{+}TC})(\Upsilon^+)$ which has a finite model iff there exists a run of T satisfying φ. (The construction closely follows a construction due to Immerman and Vardi that was first presented in [13] as a translation of the branching-time logic CTL* into (FO+TC); see also [18].)

In the following we regard $\varphi\mathbf{B}\psi$ as be a well-formed FTL formula whose semantics is given by $\varphi\mathbf{B}\psi := \neg(\neg\varphi\mathbf{U}\neg\psi)$. In this extended syntax we can, w.l.o.g., assume that φ is in negation normal form, i.e., negations in φ occur only in front of atomic subformulas. Let $\mathrm{cl}(\varphi)$ be the set of those subformulas of φ whose occurrence is not strictly inside some FO subformula of φ. For every $\theta \in \mathrm{cl}(\varphi)$, choose a boolean variable b_θ. If θ has the form $\psi\mathbf{U}\chi$, then choose an additional boolean variable m_θ, different from b_θ. Let $\bar{b}$ (resp. $\bar{m}$) denote a tuple consisting of all boolean variables b_θ (resp. m_θ) in some arbitrary but fixed order.

Every FO formula $\theta \in \mathrm{cl}(\varphi)$ is a sentence over Υ. Obtain $\theta^+(i) \in \mathrm{FO}(\Upsilon^+)$ from θ as follows: (i) replace every input, memory, and output atom of the form $R(\bar{t})$ with $R(\bar{t},i)$, and (ii) replace every FO quantifier with the corresponding D'-bounded quantifier (e.g., replace $\exists x$ with $(\exists x \in D')$ and $\forall x$ with $(\forall x \in D')$).

We come to the definition of $\chi_{T,\varphi}$. Let $next \in \mathrm{FO}(\Upsilon^+)$ with free$(next) \subseteq \{i, i', \bar{b}, \bar{b}', \bar{m}, \bar{m}'\}$ be defined by:

$$next \quad := \quad (i \in I') \wedge \chi_T(i,i') \wedge \bigwedge\nolimits_{\theta \in \mathrm{cl}(\varphi)} next_\theta,$$

where $\chi_T(i,i')$ is obtained from T according to Proposition 2, and $next_\theta$ is defined as follows: if $\theta \in \mathrm{FO}$, then $next_\theta := b_\theta \to \theta^+(i)$; if $\theta = \psi \vee(\wedge)\chi$, then $next_\theta := b_\theta \to (b_\psi \vee(\wedge) b_\chi)$; if $\theta = \mathbf{X}\psi$, then $next_\theta := b_\theta \to b'_\psi$; if $\theta = \psi\mathbf{B}\chi$, then $next_\theta := b_\theta \to (b_\chi \wedge (b_\psi \vee b'_\theta))$; and if $\theta = \psi\mathbf{U}\chi$, then

$$next_\theta \quad := \quad \big(b_\theta \to (b_\chi \vee (b_\psi \wedge b'_\theta))\big) \wedge \big(m'_\theta \to (m_\theta \vee b_\chi)\big).$$

Finally,

$$\begin{aligned}
\chi_{T,\varphi} \quad &:= \quad \exists i\bar{b}, i'\bar{b}', \bar{m}' \big(initial \wedge run\big) \\
initial \quad &:= \quad \bigwedge\nolimits_{R \in \Upsilon_{\mathrm{mem}} \cup \Upsilon_{\mathrm{out}}} \big((\forall \bar{x} \in D')\neg R(\bar{x},i)\big) \\
run \quad &:= \quad [\mathrm{TC}_{i\bar{b}\bar{m},i'\bar{b}'\bar{m}'}\ next](i\bar{b}\bar{0}, i'\bar{b}'\bar{0}) \ \wedge \\
&\qquad\ [\mathrm{TC}^S_{i\bar{b}\bar{m},i'\bar{b}'\bar{m}'}\ next](i'\bar{b}'\bar{0}, i'\bar{b}'\bar{m}') \ \wedge \\
&\qquad\ b_\varphi \wedge \bigwedge\nolimits_{\psi\mathbf{U}\chi \in \mathrm{cl}(\varphi)} (b'_{\psi\mathbf{U}\chi} \to m'_{\psi\mathbf{U}\chi}),
\end{aligned}$$

where TC^S denotes the *strict version* of the transitive closure operator TC (which is definable in (FO+TC)).

Correctness: It is not difficult to show that $(T,\varphi) \in$ RUN-SAT (ASM-T, T) iff $\chi_{T,\varphi} \in$ FIN-SAT(FO+TC). (For the "if" direction unwind the essential part of a model of $\chi_{T,\varphi}$ to obtain a periodic run of T satisfying φ. For the "only-if" direction use Lemma 3.)

Polynomial-time computability: $\chi_T(i,i')$ is polynomial-time computable from T by Proposition 2. Since the cardinality of $\mathrm{cl}(\varphi)$ is bounded by the length of φ, *next* and thus *run* can be obtained from T and φ in polynomial time. If we assume that the arities of the relation symbols in Υ are encoded in Υ, then *initial* can be obtained from Υ in polynomial time. Altogether, this shows polynomial-time computability of $\chi_{T,\varphi}$.

Next, we refine this reduction to a polynomial-time reduction of $\text{RUN-SAT}^{\mathrm{in} \le N}$ ($\mathrm{ASM}^{\mathrm{I}}$-T, T^{I}) to a decidable subproblem of FIN-SAT(FO+TC).

Step 2: Reduction of $\text{RUN-SAT}^{\mathrm{in} \le N}(\mathrm{ASM}^{\mathrm{I}}\text{-T}, \mathrm{T}^{\mathrm{I}})$. For any structure $\mathcal{A}$ with universe A, and any $X \subseteq A$ containing all constants in $\mathcal{A}$, we denote by $\mathcal{A}^X$ the substructure of $\mathcal{A}$ induced by X. Let T be an ASM transducer over Υ, $\sigma = (\mathcal{S}_i)_{i\in\omega}$ an infinite consistent sequence of states over Υ, and X the set of constants in $\mathcal{S}_0$. We call σ a *local run* of T if

1. $(\mathcal{S}_0|_{\mathrm{mem,out}})^X = (X, \varnothing)$, i.e., all memory and output relations in $\mathcal{S}_0^X$ are empty, and
2. for every $i \in \omega$, $T(\mathcal{S}_i)^X = (\mathcal{S}_{i+1}|_{\mathrm{mem,out}})^X$.

LEMMA 4. *Let T be an $\mathrm{ASM}^{\mathrm{I}}$ transducer and φ a T^{I} sentence over the vocabulary of T. If there exists a local run of T satisfying φ, then there also exists a (genuine) run of T satisfying φ.*

In favor of a succinct formulation of the next proposition, we introduce a new type of bounded quantification. A *witness set* W is a finite set of variables and constant symbols. For a witness set W and a variable x not in W, let $(x \in W)$ abbreviate $(\bigvee_{\nu \in W} x = \nu)$. Intuitively, the 'formula' $(x \in W)$ holds iff the interpretation of x matches the interpretation of some symbol in W.

Obtain the *witness-bounded fragment* of FO, denoted FO^{W}, from FO by replacing the formula-formation rule for (unbounded) quantification with the following rule for *witness-bounded quantification*:

(WBQ) If W is a witness set, x a variable not in W, and φ a formula, then $(\exists x \in W)\varphi$ and $(\forall x \in W)\varphi$ are formulas.

The free and bound variables of FO^{W} formulas are defined as usual. In particular, x occurs bound in $(\exists x \in W)\varphi$ and $(\forall x \in W)\varphi$, whereas all variables in the witness set W occur free.

We view FO^{W} as a fragment of FO, where formulas of the form $(\exists x \in W)\varphi$ and $(\forall x \in W)\varphi$ are mere abbreviations for $\exists x(x \in W \wedge \varphi)$ and $\forall x(x \in W \to \varphi)$, respectively. It is easy to see that FO^{W} is as expressive as the quantifier-free fragment of FO. However, FO^{W} allows us to represent (certain) quantifier-free formulas exponentially more succinct (unless PSPACE = NP).

From now on we tacitly assume that with every transducer vocabulary Υ there comes an arbitrary but fixed order on Υ_{in}. Let N be a natural number ≥ 1, Υ a transducer vocabulary, and $\sigma = (\mathcal{S}_0,\dots,\mathcal{S}_q)$ a consistent sequence of states over Υ with maximal input flow $\le N$, i.e., $|R^{\mathcal{S}_i}| \le N$ for every $R \in \Upsilon_{\mathrm{in}}$ and $i \in \{0,\dots,q\}$. (Think of σ as a representation of a periodic local run of an $\mathrm{ASM}^{\mathrm{I}}$ transducer.)

Because N is an upper bound on the maximal input flow of σ, the input component $\mathcal{S}_i|_{\rm in}$ of each state $\mathcal{S}_i$ can be encoded as a tuple of domain elements whose length depends only on N and $\Upsilon_{\rm in}$. We next fix such an encoding.
Let $R_1, \ldots, R_n$ be the enumeration of $\Upsilon_{\rm in}$ according to the order on $\Upsilon_{\rm in}$, k_j the arity of R_j, and D the domain of $\mathcal{S}_0|_{\rm db}$. We call $\bar{d} \in D^{N(1+k_j)}$ an *encoding* of $R_j^{\mathcal{S}_i}$ if for every $l \in \{1, \ldots, N\}$ there exist $b_l \in D$ and $\bar{c}_l \in D^{k_j}$, such that $\bar{d} = (b_1\bar{c}_1, \ldots, b_N\bar{c}_N)$ and the following two conditions hold:

1. For every $\bar{a} \in R_j^{\mathcal{S}_i}$, there exists an $l \in \{1, \ldots, N\}$ such that $b_l = 0$ and $\bar{c}_l = \bar{a}$.
2. For every $l \in \{1, \ldots, N\}$, $b_l = 0$ implies $\bar{c}_l \in R_j^{\mathcal{S}_i}$.

Let $L(N, \Upsilon_{\rm in}) := \sum_{j=1}^{n}(N(1+k_j))$. We call $\bar{e} \in D^{L(N,\Upsilon_{\rm in})}$ an *encoding* of $\mathcal{S}_i|_{\rm in}$ if $\bar{e} = (\bar{d}_1, \ldots, \bar{d}_n)$ where each $\bar{d}_j \in D^{N(1+k_j)}$ is an encoding of $R_j^{\mathcal{S}_i}$.

PROPOSITION 3. *Let N and Υ be as above, T an ASM^I transducer over Υ, and $\bar{e}$ an $L(N, \Upsilon_{\rm in})$-tuple of pairwise distinct variables. Recall the definitions of Υ^+ and $\mathcal{A}_\sigma$ (prior to Proposition 2). There is a formula $\chi_T^*(\bar{e}, i, i') \in {\rm FO}^{\rm W}(\Upsilon^+ - \Upsilon_{\rm in})$ with free$(\chi_T^*) = \{\bar{e}, i, i'\}$ such that, whenever*

- *σ and D are as above,*
- *$\mathcal{A}_\sigma^*$ is the reduct of $\mathcal{A}_\sigma$ to $\Upsilon^+ - \Upsilon_{\rm in}$,*
- *X is the set of constants in $\mathcal{S}_0$,*
- *$i, i' \in \{0, \ldots, q\}$, and*
- *$\bar{e} \in D^{L(N,\Upsilon_{\rm in})}$ is an encoding of $\mathcal{S}_i|_{\rm in}$,*

then

$$\mathcal{A}_\sigma^* \models \chi_T^*[\bar{e}, i, i'] \quad \Leftrightarrow \quad T(\mathcal{S}_i)^X = (\mathcal{S}_{i'}|_{\rm mem,\, out})^X.$$

If N is a priori fixed, then $\chi_T^*(\bar{e}, i, i')$ can be obtained from T in polynomial time.

PROOF. (Sketch.) For every $R_j \in \Upsilon_{\rm in}$, define a quantifier-free formula $decode_j(\bar{e}, \bar{x})$ as follows. Suppose that the variable tuple $\bar{e}$ has the form $(\bar{d}_1, \ldots, \bar{d}_n)$ where each $\bar{d}_{j'}$ is a $N(1+k_{j'})$-tuple. Furthermore, suppose that $\bar{d}_j$ has the form $(b_1\bar{c}_1, \ldots, b_N\bar{c}_N)$ where each $b_l\bar{c}_l$ is a $(1+k_j)$-tuple. Let $\bar{x}$ be a k_j-tuple of pairwise distinct new variables.

$$decode_j(\bar{e}, \bar{x}) \quad := \quad \bigvee\nolimits_{l=1}^{N} (b_l = 0 \wedge \bar{c}_l = \bar{x}).$$

Let C denote the set of constant symbols in Υ, and W the set of variables occurring in the variable tuple $\bar{e}$. C and W are witness sets. Let i, i' be new variables and $\Upsilon^* := \Upsilon^+ - \Upsilon_{\rm in}$. For every $\theta(\bar{x}) \in {\rm FO}^{\rm I}(\Upsilon)$, define $\theta^*(\bar{e}, \bar{x}, i) \in {\rm FO}^{\rm W}(\Upsilon^*)$ by induction on the construction of θ:

- If $\theta = R(\bar{t})$ and $R(\bar{t})$ is an input atom, say, $R = R_j$, then $\theta^* := decode_j[\bar{x}/\bar{t}]$. If $R(\bar{t})$ is a memory or output atom, let $\theta^* := R(\bar{t}, i)$. Otherwise, $\theta^* := \theta$.
- If $\theta = (t = t')$, $\theta = \neg\varphi$, or $\theta = \varphi \vee \psi$, define θ^* in the obvious way.
- If $\theta = \exists\bar{x}(\alpha \wedge \varphi)$ and α^* (resp. φ^*) is obtained from α (resp. φ) by induction hypothesis, then $\theta^* := (\exists\bar{x} \in W)(\alpha^* \wedge \varphi^*)$.

For every $R \in \Upsilon_{\rm mem} \cup \Upsilon_{\rm out}$, let $\varphi_R(\bar{x})$ and $\chi_R(\bar{x})$ be defined as in Section 2, where Π now denotes the program of T. Let $\theta_R := \chi_R$ if $R \in \Upsilon_{\rm mem}$; otherwise $\theta_R := \varphi_R$. W.l.o.g., we may assume that no variable in W occurs in any θ_R. For every $\theta_R(\bar{x})$, let $\theta_R^*(\bar{e}, \bar{x}, i) \in {\rm FO}^{\rm W}(\Upsilon^*)$ be defined inductively as above. Finally,

$$\chi_T^*(\bar{e}, i, i') \quad := \quad \bigwedge\nolimits_R \big[(\forall\bar{x} \in C)\big(\theta_R^*(\bar{e}, \bar{x}, i) \leftrightarrow R(\bar{x}, i')\big)\big],$$

where R ranges in $\Upsilon_{\rm mem} \cup \Upsilon_{\rm out}$. $\Box$

Let (FO$^{\rm W}$+TC) denote FO$^{\rm W}$ augmented with the transitive closure operator TC [7]. An occurrence of a TC operator in a (FO$^{\rm W}$+TC) formula is called *positive* if the occurrence is in the scope of an even number of negations. By (FO$^{\rm W}$+posTC) we denote the set of those (FO$^{\rm W}$+TC) formulas in which every occurrence of a TC operator is positive.

THEOREM 3. RUN-SAT$^{{\rm in}\,\leq\, N}$(ASM$^{\rm I}$-T, T$^{\rm I}$) *is polynomial-time reducible to* FIN-SAT(FO$^{\rm W}$+posTC).

PROOF. (Sketch.) We follow the reduction as in the first step, but now take locality of ASM$^{\rm I}$ transducer runs and boundedness of the maximal input flow into account.
Consider an instance (T, φ) of RUN-SAT$^{{\rm in}\,\leq\, N}$(ASM$^{\rm I}$-T, T$^{\rm I}$) with T and φ over Υ. Let $R_1, \ldots, R_n$ be the enumeration of $\Upsilon_{\rm in}$ according to the order on $\Upsilon_{\rm in}$, k_j the arity of R_j, and $\Upsilon^* := \Upsilon^+ - \Upsilon_{\rm in}$. We provide a sentence $\chi_{T,\varphi}^* \in$ (FO$^{\rm W}$+posTC)(Υ^*) which has a finite model iff there exists a run of T with maximal input flow $\leq N$ that satisfies φ.
Assume that φ is in negation normal form and let ${\rm cl}(\varphi)$, b_θ, m_θ, $\bar{b}$, and $\bar{m}$ be as in the reduction of RUN-SAT(ASM-T, T). Let $\bar{e}$ be an $L(N, \Upsilon_{\rm in})$-tuple of pairwise distinct new variables. Every FO formula $\theta \in {\rm cl}(\varphi)$ is now a FO$^{\rm I}$ sentence over Υ. Obtain $\theta^*(\bar{e}, i) \in {\rm FO}^{\rm W}(\Upsilon^*)$ from θ inductively as in the proof of Proposition 3.
To the definition of $\chi_{T,\varphi}^*$. Redefine *next* to $next^* \in {\rm FO}(\Upsilon^*)$ with free$(next^*) \subseteq \{\bar{e}, i, i', \bar{b}, \bar{b}', \bar{m}, \bar{m}'\}$:

$$next^* := (\bar{e} \in D') \wedge (i \in I') \wedge \chi_T^*(\bar{e}, i, i') \wedge \bigwedge\nolimits_{\theta \in {\rm cl}(\varphi)} next_\theta^*,$$

where χ_T^* is obtained from T according to Proposition 3, and $next_\theta^* := next_\theta$ if $\theta \notin {\rm FO}^{\rm I}$; otherwise $next_\theta^* := b_\theta \to \theta^*(\bar{e}, i)$. Let C denote the set of constant symbols in Υ.

$$\begin{aligned} \chi_{T,\varphi}^* \quad &:= \quad \exists\bar{e}i\bar{b}, \bar{e}'i'\bar{b}', \bar{m}'\big(initial^* \wedge run^*\big) \\ initial^* \quad &:= \quad \bigwedge\nolimits_{R \in \Upsilon_{\rm mem} \cup \Upsilon_{\rm out}} \big((\forall\bar{x} \in C)\neg R(\bar{x}, i)\big) \\ run^* \quad &:= \quad [{\rm TC}_{\bar{e}i\bar{b}\bar{m}, \bar{e}'i'\bar{b}'\bar{m}'}\; next^*](\bar{e}i\bar{b}\bar{0}, \bar{e}'i'\bar{b}'\bar{0}) \;\wedge \\ &\qquad\; [{\rm TC}^S_{\bar{e}i\bar{b}\bar{m}, \bar{e}'i'\bar{b}'\bar{m}'}\; next^*](\bar{e}'i'\bar{b}'\bar{0}, \bar{e}'i'\bar{b}'\bar{m}') \;\wedge \\ &\qquad\; b_\varphi \wedge \bigwedge\nolimits_{\psi{\bf U}\chi \in {\rm cl}(\varphi)} (b'_{\psi{\bf U}\chi} \to m'_{\psi{\bf U}\chi}) \end{aligned}$$

(Notice that $\chi_{T,\varphi}^*$ as defined above is not a (FO$^{\rm W}$+posTC) formula. To obtain the desired (FO$^{\rm W}$+posTC) sentence remove in the above definition of $\chi_{T,\varphi}^*$ the prefix of existential quantifiers and regard all free variables of the resulting formula as new constant symbols.)
Correctness: One can show that $(T, \varphi) \in$ RUN-SAT$^{{\rm in}\,\leq\, N}$ (ASM$^{\rm I}$-T, T$^{\rm I}$) iff $\chi_{T,\varphi}^* \in$ FIN-SAT(FO$^{\rm W}$+TC). (For the "if" direction unwind the essential part of a model of $\chi_{T,\varphi}^*$ to obtain a local run of T satisfying φ. Lemma 4 then yields a genuine run of T with maximal input flow $\leq N$ that satisfies φ. For the "only-if" direction again use Lemma 3.)
Polynomial-time computability: $\chi_T^*(\bar{e}, i, i')$ can be obtained from T in polynomial time by Proposition 3. For polynomial-time computability of $next^*$ it remains to show that for every

$\theta \in \mathrm{cl}(\varphi)$, θ^* is polynomial-time computable. This can be proved by induction on the construction of θ (and in fact was already used in the proof of Proposition 3). Polynomial-time computability of *next** immediately entails polynomial-time computability of *run**. Polynomial-time computability of *initial** follows as in the reduction of RUN-SAT(ASM-T, T). Altogether, $\chi^*_{T,\varphi}$ is polynomial-time computable. □

As a corollary we obtain containment of problems (4) and (4').

COROLLARY 3. *For any $m \geq 0$ and $N \geq 1$:*

- $\text{VERIFY}^{\text{in} \leq N}_m(\text{ASM}^{\text{I}}\text{-T}, \text{UT}^{\text{I}}) \in \text{PSPACE}$.
- $\text{VERIFY}^{\text{in} \leq N}(\text{ASM}^{\text{I}}\text{-T}, \text{UT}^{\text{I}}) \in \text{EXPSPACE}$.

PROOF. (Sketch.) Let (E+TC) denote the existential fragment of (FO+TC), and FIN-SAT_m the restrictions of FIN-SAT to instances where only symbols of arity $\leq m$ occur. One can show that:

- $\text{FIN-SAT}_m(\text{FO}^{\text{W}}\text{+posTC})$ is polynomial-time reducible to $\text{FIN-SAT}_m(\text{E+TC})$,
- $\text{FIN-SAT}_m(\text{E+TC}) \in \text{PSPACE}$, and
- $\text{FIN-SAT}(\text{E+TC}) \in \text{EXPSPACE}$.

(See, e.g., [17].) By Theorem 3, we have:

- $\text{RUN-SAT}^{\text{in} \leq N}_m(\text{ASM}^{\text{I}}\text{-T}, \text{T}^{\text{I}}) \in \text{PSPACE}$, and
- $\text{RUN-SAT}^{\text{in} \leq N}(\text{ASM}^{\text{I}}\text{-T}, \text{T}^{\text{I}}) \in \text{EXPSPACE}$.

Since both PSPACE and EXPSPACE are closed under complementation and polynomial-time reductions, the observation following the definition of RUN-SAT implies the corollary. □

One can arrange the definition of $\chi^*_{T,\varphi}$ in the proof of Theorem 3 so that it becomes a sentence over $\Upsilon^+ - (\Upsilon_{\text{in}} \cup \Upsilon_{\text{out}})$. Hence, imposing an upper bound m on the arities of database and memory relations suffices to avoid an exponential blow-up of the space complexity.

Step 3: Reduction of $\text{RUN-SAT}^{\text{db}}(\text{ASM-T}, \text{T})$. We now turn to ASM transducers. A slight upgrade of the reduction in Theorem 3 yields the promised containment results for ASM transducers which are supposed to run on a given database only.

THEOREM 4. $\text{RUN-SAT}^{\text{db,in} \leq N}(\text{ASM-T}, \text{T})$ *is polynomial-time reducible to* $\text{FIN-SAT}(\text{FO}^{\text{W}}\text{+posTC})$.

PROOF. (Sketch.) Obtain the fragment FO^{IC} of FO by adding to FO^{I} (see Definition 5) the new formula-formation rule **(WBQ)** for witness-bounded quantification (see below Lemma 4), with the restriction that witness sets now only contain constant symbols. Define ASM^{IC}-T and T^{IC} as ASM^{I}-T and T^{I} in Definition 5, respectively, except that now every occurrence of FO^{I} in the definition is replaced with FO^{IC}.

Claim 1. $\text{RUN-SAT}^{\text{in} \leq N}(\text{ASM}^{\text{IC}}\text{-T}, \text{T}^{\text{IC}})$ can be reduced to $\text{FIN-SAT}(\text{FO}^{\text{W}}\text{+posTC})$ in polynomial time.

The proof of Claim 1 is similar to the proof of Theorem 3. Claim 1, the next reduction, and the observation that this reduction also holds if the maximal input flow of runs is a priori bounded, imply the theorem.

Claim 2. $\text{RUN-SAT}^{\text{db}}(\text{ASM-T}, \text{T})$ can be reduced to $\text{RUN-SAT}(\text{ASM}^{\text{IC}}\text{-T}, \text{T}^{\text{IC}})$ in polynomial time.

To the proof of Claim 2. Consider an instance $(T, \varphi, \mathcal{D})$ of $\text{RUN-SAT}^{\text{db}}(\text{ASM-T}, \text{T})$ with T and φ over Υ, and $\mathcal{D}$ a database over Υ_{db}. W.l.o.g., we may assume that every element in $\mathcal{D}$ is denoted by a constant symbol in Υ_{db}. We construct an instance (T', φ') of $\text{RUN-SAT}(\text{ASM}^{\text{IC}}\text{-T}, \text{T}^{\text{IC}})$ such that $(T', \varphi') \in \text{RUN-SAT}(\text{ASM}^{\text{IC}}\text{-T}, \text{T}^{\text{IC}})$ iff $(T, \varphi, \mathcal{D}) \in \text{RUN-SAT}^{\text{db}}(\text{ASM-T}, \text{T})$.
Let C be the set of constant symbols in Υ_{db}, and X the set of constants in $\mathcal{D}$. X is the domain of $\mathcal{D}$ by assumption on C. For every $\theta \in \text{T}(\Upsilon)$, obtain $\theta^C \in \text{T}^{\text{IC}}(\Upsilon)$ from θ by replacing every FO quantifier with the corresponding C-bounded quantifier.
Obtain T' from T by replacing the guard θ of every rule in the program of T with θ^C. Obviously, T' is an ASM^{IC} transducer over Υ that can be constructed from T and $\mathcal{D}$ in polynomial time. For every infinite sequence σ of states over Υ, σ is a run of T' on $\mathcal{D}$ iff σ is a run of T on $\mathcal{D}$. In other words, T' and T are equivalent on the database $\mathcal{D}$. Let $\varphi' \in \text{T}^{\text{IC}}(\Upsilon)$ be defined by:

$$\begin{aligned}
\varphi' &:= \varphi^C \wedge \psi \wedge \chi \\
\psi &:= \mathbf{G} \bigwedge\nolimits_{R \in \Upsilon_{\text{in}}} \forall \bar{x}\, (R(\bar{x}) \rightarrow \bar{x} \in C) \\
\chi &:= \bigwedge\nolimits_{\mathcal{D} \models \gamma} \gamma,
\end{aligned}$$

where γ ranges in the set of atomic and negated atomic sentences over Υ_{db}. φ' can be obtained from T, φ, and $\mathcal{D}$ in polynomial time.
Correctness: For the "only-if" direction suppose that there exists a run ρ of T on $\mathcal{D}$ satisfying φ. Since T' and T are equivalent on $\mathcal{D}$, ρ is also a run of T'. It is easy to check that $\rho \models \varphi'$. This shows $(T', \varphi') \in \text{RUN-SAT}(\text{ASM}^{\text{IC}}\text{-T}, \text{T}^{\text{IC}})$.
For the "if" direction assume that $\rho' = (\mathcal{S}'_i)_i$ is a run of T' satisfying φ'. Let $\mathcal{D}'$ be the database $\mathcal{S}'_0|_{\text{db}}$, and X' the set of constants in $\mathcal{D}'$. Because $\mathcal{D}' \models \chi$, we may assume that $X' = X$ and $\mathcal{D}$ is a substructure of $\mathcal{D}'$. Let $\rho := (\mathcal{S}_i)_i$, where $\mathcal{S}_i := \mathcal{S}'^X_i$. One can verify that ρ is a run of T' satisfying φ. Since ρ is a run on $\mathcal{D}$, and T' and T are equivalent on $\mathcal{D}$, ρ is also a run of T on $\mathcal{D}$. Hence, $(T, \varphi, \mathcal{D}) \in \text{RUN-SAT}^{\text{db}}(\text{ASM-T}, \text{T})$. □

COROLLARY 4. $\text{RUN-SAT}^{\text{db}}_m(\text{ASM-T}, \text{T})$ *is polynomial-time reducible to* $\text{FIN-SAT}_{m+1}(\text{FO}^{\text{W}}\text{+posTC})$.

PROOF. (Sketch.) Consider an instance $(T, \varphi, \mathcal{D})$ of $\text{RUN-SAT}^{\text{db}}_m(\text{ASM-T}, \text{T})$ as in the proof of Claim 2. Let d denote the cardinality of $\mathcal{D}$. First, reduce $(T, \varphi, \mathcal{D})$ to an instance (T', φ') of $\text{RUN-SAT}(\text{ASM}^{\text{IC}}\text{-T}, \text{T}^{\text{IC}})$ according to the proof of Claim 2. Then, with $N := d^m$, reduce (T', φ') to an instance $\chi^*_{T',\varphi'}$ of $\text{FIN-SAT}(\text{FO}^{\text{W}}\text{+posTC})$ according to the proof of Claim 1. $\chi^*_{T,\varphi}$ is polynomial-time computable because $L(N, \Upsilon_{\text{in}})$ is polynomially bounded in the size of $\mathcal{D}$ and Υ_{in}, if m is fixed. □

Containment of problems (1) and (1') now follows by Corollary 4 and Theorem 4 as in the proof of Corollary 3.

COROLLARY 5. *For any $m \geq 0$ and $N \geq 1$:*

- $\text{VERIFY}^{\text{db}}_m(\text{ASM-T}, \text{UT}) \in \text{PSPACE}$.
- $\text{VERIFY}^{\text{db,in} \leq N}(\text{ASM-T}, \text{UT}) \in \text{EXPSPACE}$.