Projection Views of Register Automata

Luc Segoufin
INRIA, Laboratoire Cogitamus
luc.segoufin@inria.fr

Victor Vianu
UC San Diego, INRIA, ENS
vianu@cs.ucsd.edu

ABSTRACT

Register automata have been used as a convenient model for specifying and verifying database driven systems. An important problem in such systems is to provide views that hide or restructure certain information about the data or process, extending classical notions of database views. In this paper we carry out a formal investigation of views of register automata by considering simple views that project away some of the registers. We show that classical register automata are not able to describe such projections and introduce more powerful register automata that are able to do so. We also show useful properties of these automata such as closure under projection and decidability of verifying temporal properties of their runs.

ACM Reference Format:
Luc Segoufin and Victor Vianu. 2020. Projection Views of Register Automata. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3375395.3387651

1 INTRODUCTION

Software systems centered around a database are pervasive in numerous applications. They are encountered in areas as diverse as electronic commerce, e-government, scientific applications, enterprise information systems, and business process management. Such systems are generally governed by highly complex workflows that involve stakeholders with very different needs and permissions. It is therefore critical to provide users with *views* of the underlying workflow, customized according to their role. Such views present the same workflow at various levels of abstraction that expose only the information relevant (or authorized) for a class of users.

While views are an integral part of practical workflow design, there have been few rigorous studies of specification mechanisms and semantics of data-driven workflow views. In this paper we carry out a formal investigation of basic views that project away some of the data as it evolves in the course of the workflow. We do so using *register automata*, which have been used as a convenient model for specifying and verifying database driven systems. Specifically, we consider *projection views* that hide some of the registers of the automaton, and possibly a portion of the database. Our main objective is to develop mechanisms for specifying the images of such projection views. This would yield customized workflows that explain to users their local views of the global workflow.

Register automata capture workflows in which a record of data values evolves as specified by transition rules. This is essentially the same as the popular *artifact* model (see related work). Each transition rule specifies, given a current record, the set of possible successor records. The automaton is equipped with an underlying relational database that can be queried by the transition rules. In addition, the automaton has a finite-state control including a Büchi acceptance condition. A run of the automaton on a given database is an infinite sequence of consecutive records satisfying the transition rules and the Büchi acceptance condition.

As an informal example, consider the (simplified) workflow of a manuscript reviewing system. The treatment of each paper might be modeled by a set of values that evolve throughout the workflow, identified by attributes such as *paper-id, author, topic, paper-state, reviewer, review-state.* There might also be an underlying database, with one relation holding the topic of each paper and another the topics that each reviewer prefers to review. The transitions follow the standard workflow for a reviewing system: a paper is submitted, a reviewer is assigned nondeterministically based on topic, the state of the paper transitions to under-review, the reviewers carry out their own workflow, possibly invoving sub-reviewers, and this proceeds until a decision is reached, possibly with a loop due to revisions. In the register automaton model, the data values of the attributes would be held in corresponding registers. The transitions would specify the above stages in the processing of the paper, using the registers and the database. While in this example one would expect the workflow to complete in finitely many steps, its runs can be easily made infinite, as in the formal model, by looping forever in the final state. Note that in this scenario, some of the users might see only a subset of the attributes. For instance, authors do not see their reviewers or the reviewer-states. And if reviewing is double blind, reviewers do not see the authors. These are projection views that hide some of the registers of the automaton.

Unfortunately, register automata are not closed under projection even in the absence of a database: it is easy to construct a register automaton whose projection view cannot be described by another register automaton. Indeed, the problem of describing the projections of register automata turns out to be challenging, and a general solution remains elusive. We mainly focus in this paper on the simpler case of register automata without a database, and make partial progress in the case when a database is present.

Since register automata are not closed under projection, we define more powerful *extended* register automata. Extended automata augment register automata with global constraints, requiring equalities or inequalities among data values held in registers that may be far apart in the run, but related using regular expressions. In the absence of a database, we show that extended register automata can not only specify projections of register automata, but are themselves closed under projection. In terms of expressive power, we show that extended register automata can express more than the projections of register automata. We provide a precise characterization of the subclass of extended automata that are projections of register automata. Note that being the projection of a register automaton is a desirable property, because it means, intuitively, that the global constraints can be enforced entirely by local transitions, in a streaming fashion, at the cost of additional registers.

Although motivated by projection views, extended automata are interesting in their own right, because global constraints occur naturally in workflows. We show that testing emptiness of extended automata is decidable, even when a database is present. As a consequence, verification for a class of temporal properties of runs (LTL-FO) is decidable as well.

In the presence of a database, we make partial progress in tackling projections of register automata. First, we show that extended automata are no longer able to describe projections of register automata. We identify two additional types of global constraints that are needed in order to describe projections. The first generalizes the inequality constraints of extended automata to inequalities of *tuples* of register values (effectively introducing disjunctions of inequalities). The second requires that a specified set of register values occurring in the run be finite. We show that with these additional features, extended register automata are able to specify projection views of register automata where some of the registers and the entire database are hidden.

Our results make use of a variety of techniques in logic and automata theory, including the finite-model theory of guarded first-order logic, and Monadic Second Order logic (MSO) on graphs and infinite strings. The decidability of verification for extended automata relies on showing that the set of state traces (the infinite sequences of states of their runs) is quasi-regular (an extension of ω-regularity), and emptiness of quasi-regular languages is decidable [5]. This stands in contrast with register automata, for which the set of state traces is ω-regular [19]. As a side benefit, our proof of quasi-regularity for extended automata also provides an alternative, simpler proof of the ω-regularity result of [19]. Regarding MSO, besides routine connections to (ω)-regular languages, we use non-trivial results on properties of graphs defined by MSO on infinite strings, as well as decidability of satisfiability on strings of certain extensions of MSO with bounding quantifiers [5, 10].

Related work. Formal work on database-driven systems has largely focused on automatic verification of temporal properties. For surveys of this considerable body of work, see [8, 13]. One of the most well-studied models of data-driven workflows are *artifact systems*, that are formal counterparts of IBM's business artifacts introduced in [20, 23]. In its simplest incarnation, an artifact system is essentially a register automaton equipped with a relational database.

Research on workflow views has traditionally focused on process-centric specifications (e.g. [9, 16, 21, 27]). These abstract away the data and their manipulations already in the original workflow specification. [26] considers view generation for artifact systems, but in a limited setting in which there is no database queried by the workflow. Abstraction operations are restricted to the special case in which some of the boolean artifact variables, seen as states of a finite-state automaton, are replaced with their ancestors in a hierarchy of states. Views of artifacts are discussed in [18] in the context of service interoperation hubs, a framework supporting business collaborations. The views include both the data and the process. The data portion of a view is essentially a select-project view, while the process view specifies a *condensation* operation in which multiple states are mapped to a single state. There is no attempt to synthesize specifications for the images of such views.

An abstract model of workflow views, used to compare the expressiveness of different workflow models, is introduced in [1]. The approach relies on the notion of *abstract tree of runs*, obtained by applying a view to the data and transitions of the workflow. Different workflows are compared by defining views that map them to a common abstraction and then using a notion of simulation on the resulting trees of runs. As an example, this mechanism is used to show that a particular workflow model based on XML is more powerful than the basic artifact model relative to appropriate abstractions. While the views considered are very general, the problem of describing their images is not addressed.

In [3], collaborative workflows involving multiple users are modeled using a local-as-view approach, whereby the data seen by each user is defined by projections of database relations. The goal is to enable users to reason about the global workflow using their own local observations. This is further pursued in [2], using selection-projection views, with the goal of providing runtime and static explanations of a user's view of the workflow. The static variant aims to produce a workflow specification describing the user's view, which is somewhat similar in flavor to the goal of this paper. However, the differences in the models and the restrictions imposed in [2] in order to obtain the specifications, render the two approaches incomparable.

The previous work most closely related to our paper is that of [19], which studies views of artifact systems that strip away the data and retain just the transitions occurring in runs. The goal is to determine under what conditions the linear-time and branching-time views are regular. It is shown that the linear-time views of tuple artifacts are always ω-regular, but branching-time views are only regular under additional restrictions. The impact of data dependencies (tuple and equality-generating dependencies) on regularity of the views is also studied. The linear-time views of [19] are essentially the state traces of our register automata. The proof of their ω-regularity in [19] relies on showing that every sequence of states satisfying a local consistency condition is in fact the state trace of a real run. In contrast, the alternative proof in the present paper relies on the finite-model property of a class of guarded first-order (FO) sentences.

Organization. The paper is organized as follows. Section 2 introduces basic concepts and terminology, as well as register automata. Extended register automata are defined in Section 3, and the quasi-regularity of their state traces is shown, leading to decidability of verification. Closure of extended register automata under projection is shown in Section 4. In particular, this shows that extended automata are powerful enough to specify projections of register automata. The precise fragment required to specify such projections is characterized in Section 5. The results on projections of register automata with a database are presented in Section 6. The paper ends with a few concluding remarks.

2 PRELIMINARIES

After introducing some basic notation, we present the model of register automata.

We view a database as a finite relational structure. A relational signature, or database schema, is a finite set of relation symbols with associated arities (non-negative integers). We also allow finitely many constant symbols in the schema. We usually denote our database schema by σ. We fix an infinite data domain $\mathbb{D}$. A database over σ is a mapping D that associates to each relation $R \in \sigma$ of arity κ a finite κ-ary relation over $\mathbb{D}$, and to each constant symbol an element of $\mathbb{D}$. The active domain of a database D, denoted $adom(D)$, consists of all values occurring in the relations of D, together with the constants.

We assume familiarity with first-order logic (FO) and its usual semantics over relational structures. When querying a database D, we will only use quantifier-free FO formulas. As usual, given a database D, a quantifier-free FO formula $\varphi(\bar{x})$ with variables $\bar{x}$, and a tuple $\bar{a}$ of elements of $\mathbb{D}$ of the same arity as $\bar{x}$, we denote by $D \models \varphi(\bar{a})$ the fact that the formula φ holds true in D with the valuation associating $\bar{a}$ to $\bar{x}$. We also use Monadic Second-Order (MSO) formulas with standard semantics to express properties of tuples of positions in an infinite string. Given an infinite string w, an MSO formula $\phi(\bar{x})$ with free first-order variables $\bar{x}$, and a tuple $\bar{a}$ of positions of w of the same arity as $\bar{x}$, we denote by $w \models \phi(\bar{x})$, the fact that w satisfies ϕ with the valuation associating $\bar{a}$ to $\bar{x}$. Recall that the set of strings satisfying an MSO sentence (no free variables) forms a regular language [7].

We distinguish a subset of first order formula that we call *type*, consisting of quantifier-free conjunctive formulas. As they play a central role in this paper, we define them in detail. An atom over σ is either an equality expression $x = z$ or an expression of the form $R(\bar{z})$, where $\bar{z}$ is a tuple of variables or constants of appropriate arity. A literal over σ is either an atom or a negated atom, i.e $\neg R(\bar{z})$ or $x \neq y$. A σ-type over $\bar{z}$ is a satisfiable conjunction of literals over σ using variables in $\bar{z}$ (we often omit σ when it is clear from the context). A type is complete if for each m-tuple $\bar{y}$ contained in $\bar{z} \cup \bar{c}$, where $\bar{c}$ are the constants of σ, and each relation R of arity m in σ, the type includes a conjunct specifying whether or not $R(\bar{y})$ holds. Moreover, for every pair (x, y) where x is a variable from $\bar{z}$ and y is either a variable from $\bar{z}$ or a constant from $\bar{c}$, a complete type specifies whether or not x and y are equal. Notice that every type can be extended into a complete type and there may be exponentially many completions of a given type. An equality type is a special case of σ-type when σ is empty or contains only constant symbols $\bar{c}$. An equality type over $\bar{z}$ then only specifies the (in)equality constraints among the elements in $\bar{z} \cup \bar{c}$.

The following standard notation will be used in this paper. If $\bar{d}$ is a k-ary tuple and $i \leq k$ then $\bar{d}[i]$ denotes its i^{th} component. For a positive integer n, we denote $[\![n]\!] = \{1, \ldots, n\}$.

We use the usual definition for database driven register automata, see for instance [6, 25]. A register automaton $\mathbf{A}$ is a tuple $(k, \sigma, Q, I, F, \Delta)$ where k is the number of registers (possibly zero), σ is a relational signature, Q a finite set of states with initial states I and final states F, and Δ a finite set of transitions. A transition is a triple (p, δ, q) where p and q are states and δ is a σ-type over $\bar{x} \cup \bar{y}$, where $\bar{x}$ and $\bar{y}$ are two k-tuples of distinct variables. The variables $x_1 \cdots x_k$ denote the value of the k registers before the transition is fired while $y_1 \cdots y_k$ denote the value of those registers after the transition was fired. The type δ then specifies how the registers can change.

Let D be a database over σ. A *run* ρ of $\mathbf{A}$ over D is an infinite sequence of triples $\{(\bar{d}_n, q_n, \delta_n)\}_{n\geq 0}$ where $q_0 \in I$, some state in F occurs infinitely often, and for each $n \geq 0$:

- $\bar{d}_n$ is a k-ary tuple of elements in $\mathbb{D}$,
- (q_n, δ_n, q_{n+1}) is a transition in Δ,
- $\delta_n(\bar{d}_n, \bar{d}_{n+1})$ holds in D,

For technical reasons, we also assume that for every run ρ there are infinitely many values in $\mathbb{D}$ that do not occur in ρ.

For a run $\rho = \{(\bar{d}_n, q_n, \delta_n)\}_{n\geq 0}$ of $\mathbf{A}$ we consider the following:

- the *register trace* is $\{\bar{d}_n\}_{n\geq 0}$
- the *control trace* is $\{(q_n, \delta_n)\}_{n\geq 0}$
- the *state trace* is $\{q_n\}_{n\geq 0}$

For a given database D, we denote by $\mathcal{R}eg(D, \mathbf{A})$ the set of register traces of the runs of $\mathbf{A}$ over D. The set of register traces of all runs of $\mathbf{A}$ over all possible databases is denoted by $\mathcal{R}eg(\mathbf{A})$. The corresponding control traces are denoted by $\mathcal{C}ontrol(\mathbf{A})$ while the state traces are denoted by $\mathcal{S}tate(\mathbf{A})$.

Recall that our main motivation in this paper is to study automata as models of workflows. In a workflow, the purpose of the specification is to describe the evolution of data (aka registers) in the course of the workflow, for every given database. Therefore, it is natural to use $\mathcal{R}eg(D, \mathbf{A})$ and $\mathcal{R}eg(\mathbf{A})$ as measures of expressiveness of our automata.

Example 1. *Consider the 2-register automaton $\mathbf{A}$ with states q_1 and q_2, initial and final state q_1, no database (σ is empty) and transitions: $\{(q_1, \delta_1, q_2), (q_2, \delta_2, q_2), (q_2, \delta_3, q_1)\}$, where δ_1 is the equality type $x_1 = x_2 \wedge x_2 = y_2$, δ_2 the equality type $x_2 = y_2$, and δ_3 is the equality type $x_2 = y_2 \wedge y_2 = y_1$. The first type, δ_1, tests that the current two registers have the same data value using $x_1 = x_2$, and copies this value to the second register of the next position, $x_2 = y_2$. The second type, δ_2, simply propagates the value of the second register. The third type, δ_3, propagates the value of the second register and also copies its content to the first one, $y_1 = y_2$. A typical run is of the form:*

$$(d_1 d_1, q_1, \delta_1)(d_2 d_1, q_2, \delta_2)(d_3 d_1, q_2, \delta_2)(d_4 d_1, q_2, \delta_3)(d_1 d_1, q_1, \delta_1) \cdots$$

The control and state traces are:
$\mathcal{C}ontrol(\mathbf{A}) = ((q_1, \delta_1)(q_2, \delta_2)^*(q_2, \delta_3))^\omega$ *and* $\mathcal{S}tate(\mathbf{A}) = (q_1 q_2^+)^\omega$.

It will be useful in several of our proofs to assume that the register automaton $\mathbf{A}$ is *complete* in the sense that in each of its transitions the σ-type is complete. As our model is non-deterministic this can be assumed without loss of generality in terms of expressive power. However this may be at the cost of an exponential blow-up in the size of the automaton.

Example 2. *The register automaton of Example 1 is not complete as, for instance, δ_1 does not say anything about how the registers compare at the next step, i.e. does not enforce any relationship between y_1 and y_2. To complete it we would need to replace it with two types δ_1' and δ_1'' containing the literals of δ_1 together with $y_1 = y_2$ in the case of δ_1' and $y_1 \neq y_2$ in the case of δ_1'' and replace any transition using δ_1 with two transitions, one with δ_1', one with δ_1''. Observe that, because $x_1 = x_2 = y_2$ hold in δ_1, settling the relationship between y_2*

and y_1 also settles all other relationships. The types δ_2 and δ_3 can be completed in a similar way.

It will also be useful to assume that $\mathbf{A}$ is *state-driven*, meaning that for each state p there is at most one δ such that $(p, \delta, q) \in \Delta$ for some q. Every automaton can be converted to a state-driven one at the cost of a quadratic blowup: if $\mathbf{A} = (k, \sigma, Q, I, F, \Delta)$ is a register automaton then its state-driven variant is the register automaton $\mathbf{A}' = (k, \sigma, Q \times X, I', F', \Delta')$ where X is the set of σ-types occuring in Δ, I' is the set of pairs (p, δ) such that $p \in I$, similarly for F', the transitions are the tuples $((p, \delta), \delta, (q, \delta'))$ where $(p, \delta, q) \in \Delta$. Note that a state-driven automaton is not necessarily deterministic. Observe that in a state-driven automaton, the state trace of a run uniquely determines its control trace.

Example 3. *Consider again the register automaton $\mathbf{A}$ of Example 1. It is not state driven as q_2 occurs in two transitions with δ_2 and δ_3. Consider the register automaton $\mathbf{A}'$ with three states, q_1, q_2', q_2'' and transitions (q_1, δ_1, q_2'), (q_1, δ_1, q_2''), (q_2', δ_2, q_2'), (q_2', δ_2, q_2'') and (q_2'', δ_3, q_1). It is easy to verify that $\mathbf{A}'$ and $\mathbf{A}$ have the same register traces.*

When looking at a sequence of the form $\{(q_n, \delta_n)\}_{n \geq 0}$ can we say that it is actually the control trace of some register automaton? The notion of symbolic control trace from [19], slightly adapted to our context, provides a positive answer. Roughly speaking, a sequence as above is a symbolic control trace of a register automaton if any two consecutive symbols could be generated by a transition of the automaton. More formally, let $\mathbf{A} = (k, \sigma, Q, I, F, \Delta)$, and $\bar{c}$ the constants symbols of σ.

An ω-word $\{(q_n, \delta_n)\}_{n \geq 0}$ is called a *symbolic control trace* of $\mathbf{A}$ if (i) $q_0 \in I$ and there is a state in F that occur infinitely often, (ii) for every $n \geq 0$ $(q_n, \delta_n, q_{n+1}) \in \Delta$ and (iii) δ_n and δ_{n+1} agree on the common registers, i.e. $\delta_n|\bar{y}$ is isomorphic to $\delta_{n+1}|\bar{x}$ (by the isomorphism that maps y_i to x_i for $i \in [\![k]\!]$, where for a tuple $\bar{z}$ of variables, $\delta|\bar{z}$ is the conjunction of literals of δ using only variables from $\bar{z}$ or constants). We denote by $\mathcal{SC}ontrol(\mathbf{A})$ the set of symbolic control traces of $\mathbf{A}$. Clearly, $\mathcal{SC}ontrol(\mathbf{A})$ is ω-regular. It is shown in [19] that in fact $\mathcal{C}ontrol(\mathbf{A}) = \mathcal{SC}ontrol(\mathbf{A})$. In particular, for every complete register automaton $\mathbf{A}$, the languages $\mathcal{C}ontrol(\mathbf{A})$ and $\mathcal{S}tate(\mathbf{A})$ are ω-regular languages.

3 EXTENDED REGISTER AUTOMATA

We wish to study projections of the register traces of an automaton. Given a k-register automaton $\mathbf{A}$, a database D, and an integer $m \leq k$, we denote by $\Pi_m(\mathcal{R}eg(D, \mathbf{A}))$ the projection of $\mathcal{R}eg(D, \mathbf{A})$ retaining only the values for registers 1 to m (the case where $m = 0$ corresponds to projecting out all registers). Register automata are not closed under projections in the sense that there may be no m-register automaton $\mathbf{A}'$ such that for all database D, $\mathcal{R}eg(D, \mathbf{A}') = \Pi_m(\mathcal{R}eg(D, \mathbf{A}))$. Indeed, this is the case even with no database, as illustrated by the following.

Example 4. *Consider again the register automaton $\mathbf{A}$ of Example 1. Now consider the projection of the runs of $\mathbf{A}$ on the first register. The register trace of these runs contain all sequence of data values such that the initial data value of the register occurs infinitely often. This cannot be enforced by a register automaton. Indeed, suppose towards a contradiction that $\mathbf{A}'$ is such a register automaton (which we may assume complete). Consider the accepting run of $\mathbf{A}'$ whose register trace is $d_1 d_2 d_1 d_2 \cdots$. By definition, all transitions of this run contain the equality type $x_1 \neq y_1$. Hence, replacing d_1 by d_3 everywhere except the first position also yields an accepting run of $\mathbf{A}'$. However, $d_1 d_2 d_3 d_2 d_3 \cdots$ is not a register trace in the projection.*

As suggested by the above, describing projections of runs requires a mechanism for specifying equalities and inequalities among register values that are no longer local to individual transitions. The example suggests that the paths between related values in a run could be described using regular expressions of states. To this end, we introduce the more powerful *extended* register automata model.

An *extended register automaton* is a pair $\mathcal{A} = (\mathbf{A}, \Sigma)$ where $\mathbf{A} = (k, \sigma, Q, I, F, \Delta)$ is a register automaton and Σ is a finite set of regular expressions over Q, each denoted by $e^{=}_{ij}$ or $e^{\neq}_{ij}$, where $i, j \in [\![k]\!]$. We say that a run $\rho = \{(\bar{d}_n, q_n, \delta_n)\}_{n \geq 0}$ of $\mathbf{A}$ satisfies Σ, denoted $\rho \models \Sigma$, if for every $e^{=}_{ij} \in \Sigma$ and every $e^{\neq}_{ij} \in \Sigma$ and $0 \leq n_1 \leq n_2$, if $q_{n_1} \ldots q_{n_2} \in e^{=}_{ij}$ then $\bar{d}_{n_1}[i] = \bar{d}_{n_2}[j]$ and if $q_{n_1} \ldots q_{n_2} \in e^{\neq}_{ij}$ then $\bar{d}_{n_1}[i] \neq \bar{d}_{n_2}[j]$. The set of runs of $\mathcal{A}$ consists of the runs of $\mathbf{A}$ that satisfy Σ. The set of register, control, and state traces are defined as for register automata.

Observe that in an extended automaton $\mathcal{A} = (\mathbf{A}, \Sigma)$ there are two independent sources of constraints over the register values: the local constraints enforced by the σ-type δ in a transition of $\mathbf{A}$ and the global constraints Σ. Note that the global constraints can simulate the local (in)equality constraints, so the types used in transitions may have no (in)equalities if so desired, without loss of expressiveness.

Example 5. *Consider again Example 4. The projection of the runs of $\mathbf{A}$ on its first register can be described using the extended automaton $\mathcal{B} = (\mathbf{B}, \Sigma)$, where $\mathbf{B}$ is an automaton with one register, states $\{p_1, p_2\}$ where p_1 is both initial and accepting, and transitions $\{(p_1, \gamma, p_2), (p_2, \gamma, p_2), ((p_2, \gamma, p_1)\}$ where γ is the empty type, and Σ consists of $e^{=}_{11} = p_1 p_2^+ p_1$. The global constraint Σ enforces that there is a data value d such that each time $\mathbf{B}$ switches to state p_1 the data value of the register is d.*

We will see that, in the absence of a database, our extended model of register automata is powerful enough to describe the projection of the register trace of any register automaton. This is no longer the case when a database is present. Indeed, we will see in Section 6 that additional features are needed in this case.

We first notice that the additional expressive power of extended register automata is only due to the global inequality constraints. Indeed, as shown below, the global equality constraints can be simulated using extra registers.

Proposition 6. *For each extended automaton $\mathcal{A}$ with k registers, there exists an extended register automaton $\mathcal{B}$ with no global equality constraints such that for all database D, $\mathcal{R}eg(D, \mathcal{A}) = \Pi_k(\mathcal{R}eg(D, \mathcal{B}))$.*

Proof. Let $\mathcal{A} = (\mathbf{A}, \Sigma)$. The general idea for constructing $\mathcal{B} = (\mathbf{B}, \Gamma)$ is as follows. On the first k registers, $\mathbf{B}$ just simulates the transitions of $\mathbf{A}$. To enforce the global equality constraints in Σ, $\mathbf{B}$ uses additional registers. Consider an equality constraint $e^{=}_{ij} \in \Sigma$ and let $Q^{=}$ be the set of states of the minimal automaton for

$e^{=}_{ij}$. In order to enforce the corresponding constraint, at any time during the run, **B** nondeterministically guesses whether the current position is involved in a global equality test using $e^{=}_{ij}$ and then verifies that the guess is correct.

In the case where the guess is "no", **B** starts a simulation of the automaton for $e^{=}_{ij}$ and rejects if it ever reaches an accepting state.

In the case where the guess is "yes", **B** stores the current value of register i in a new register and also starts a simulation of the automaton for $e^{=}_{ij}$. Whenever the automaton reaches an accepting state, **B** checks that the data value stored in register j equals the one stored in the new register. We only need finitely many registers because it is enough to have one register per state in $Q^{=}$. Indeed, if two simulations are in the same state, the run can only proceed if the associated registers hold the same data value. The global constraints Γ of $\mathcal{B}$ are the inequality constraints of Σ, lifted to the states of **B**. The Büchi acceptance condition of **A** is also lifted easily to **B**. □

Global inequality constraints however cannot always be simulated using extra registers. This is illustrated with the following example.

Example 7. *Consider an extended automaton $\mathcal{A}$ with only one register and no database, whose global constraints ensure that all register values occurring in a run are distinct. This automaton cannot be simulated even using using extra registers. The proof of this fact is postponed to Section 5, Example 17. We only show here that there cannot be a 1-register automaton equivalent to $\mathcal{A}$. Suppose towards a contradiction that there is a register automaton* **A** *such that $\mathcal{R}eg(\mathbf{A}) = \mathcal{R}eg(\mathcal{A})$. Consider a run $\rho = \{(d_n, q_n, \delta_n)\}_{n\geq 0}$ of* **A** *in which all register values are distinct. Clearly, each δ_i must either be empty or $x_1 \neq y_1$. Suppose δ_i is empty for some i. Then ρ' obtained by replacing d_{i+1} with d_i is still a run of* **A** *but not one of $\mathcal{A}$. Now suppose that all δ_i are $x_1 \neq y_1$. Let d be a fresh data value and ρ' be obtained from ρ by replacing each d_i by d for all even i. Again, ρ' is a run of* **A** *but is not a run of $\mathcal{A}$. Thus,* **A** *does not have the same register traces as $\mathcal{A}$.*

Adding global regular constraints seems innocuous, but it raises several technical challenges. In particular, the result of Koutsos and Vianu mentioned above, establishing the ω-regularity of the state traces, no longer holds. This is shown by the following example.

Example 8. *Consider an extended automaton $\mathcal{A}$ with only one register and two states p and q. The signature σ of $\mathcal{A}$ contains only one unary symbol P, hence the databases are finite sets. The transitions of $\mathcal{A}$ ensure that the data value of the register is always in the domain of the database, i.e. contains $P(x_1)$. The global constraints of $\mathcal{A}$ ensure that between any two occurrences of state p with no occurrence of state q in between, the register values are pairwise disjoint. Hence the state traces of $\mathcal{A}$ are such that there is a finite bound (the size of the domain of the database) on the size of the longest consecutive sequence of occurrences of p, a non ω-regular property.*

Having some well-behaved characterization of the state traces is important for deciding various properties of register automata, such as the existence of a run, or whether all its runs satisfy properties specified in LTL or other reasonable logics. In the case of extended register automata, we can show that the control and state traces can be described by a well-behaved extension of ω-regular languages introduced by Bojańczyk, called quasi-regular, for which satisfiability over ω-words remains decidable. The quasi-regular languages include all languages that can be defined by a sentence of the form $\exists N \forall S(\varphi(S) \rightarrow |S| < N)$, where N is a natural number, S a set variable and φ a formula in MSO, which is sufficient for our purposes. Decidability of satisfiability of quasi-regular languages is shown in [5]. This is significant because, as in the case of register automata, it can be used to show decidability of a wide range of static analysis questions. Quasi-regular languages are also shown in [5] to be closed under union, intersection and homomorphisms.

We use here (and later) the following notation. Consider $w = \{(q_n, \delta_n)\}_{n\geq 0} \in \mathcal{SC}ontrol(\mathcal{A})$. Let us represent by (x, i) the register i in position x. Define the equivalence relation $\sim^{\mathcal{A}}_{w}$ on the set $\{(x, i) \mid x \geq 0, i \in [\![k]\!]\}$ as the reflexive, symmetric, transitive closure of the equalities induced by Σ and all δ_n's. In particular if $q_n \cdots q_m \in e^{=}_{ij}$ for some $n < m$ then $(n, i) \sim^{\mathcal{A}}_{w} (m, j)$ and if $x_i = y_j$ is part of δ_n then $(n, i) \sim^{\mathcal{A}}_{w} (n + 1, j)$. We denote the equivalence class of (x, i) by $[(x, i)]^{\mathcal{A}}_{w}$ and by $\mathcal{E}^{w}_{\mathcal{A}}$ the set of equivalence classes of $\sim^{\mathcal{A}}_{w}$. For $\epsilon_1, \epsilon_2 \in \mathcal{E}^{w}_{\mathcal{A}}$, define $\epsilon_1 \not\approx^{\mathcal{A}}_{w} \epsilon_2$ if there exists δ_n or a constraint in Σ that specifies that two members of ϵ_1 and ϵ_2 are not equal. If $\mathcal{A}$ is state-driven, we sometimes use by slight abuse in the above notation a state trace w instead of the induced control trace. Whenever $\mathcal{A}$ is understood we omit it from the notation and use simply $\sim_w$, $[(x, i)]_w$ and $\mathcal{E}^w$. We show the following.

Theorem 9. *Given an extended register automaton $\mathcal{A}$, $\mathcal{C}ontrol(\mathcal{A})$ and $\mathcal{S}tate(\mathcal{A})$ are quasi-regular.*

Proof. Let $\mathcal{A} = (\mathbf{A}, \Sigma)$ be an extended register automaton with k-registers. We can assume that $\mathcal{A}$ is complete, since otherwise the original control traces can be obtained from the corresponding completed automaton via a homomorphism. Similarly, we show the proof only for $\mathcal{C}ontrol(\mathcal{A})$, since $\mathcal{S}tate(\mathcal{A})$ is a homomorphic image of $\mathcal{C}ontrol(\mathcal{A})$.

In view of Proposition 6 we assume that Σ contains only inequality global constraints, as equality global constraints can be simulated using extra registers and states. Observe that the original control traces can be recovered by a homomorphism, so quasi-regularity is not affected.

By slight abuse, we say that an equivalence class ϵ of $\sim_w$ is in the active domain of the database relative to w if for some $(n, i) \in \epsilon$, x_i occurs in a positive relational literal of δ_n or y_i occurs in a positive relational literal of δ_{n-1}. We denote the set of such equivalence classes by $adom_w(\mathcal{E}^{w}_{\mathcal{A}})$

Consider $w \in \mathcal{SC}ontrol(\mathbf{A})$. We associate to w a graph G_w as follows. The vertices are the equivalence classes in $adom_w(\mathcal{E}^{w}_{\mathcal{A}})$. There is an edge between ϵ_1 and ϵ_2 if $\epsilon_1 \not\approx_w \epsilon_2$. Each equivalence class ϵ can be represented by a pair (x, i) where x is the smallest position of w in which a member of ϵ occurs, and i it the smallest register containing it in position x. Clearly, there is an MSO formula that defines G_w on w using the representatives of each class. Recall that we also have an MSO formula stating that $w \in \mathcal{SC}ontrol(\mathbf{A})$. From this formula we can construct a formula witnessing quasi-regularity, by stating that w is in $\mathcal{SC}ontrol(\mathbf{A})$ and that there is a bound N on the largest clique of G_w. Clearly, this can be expressed by a formula ψ of the form $\exists N \forall S(\varphi(S) \rightarrow |S| < N)$. We next prove that the language defined by ψ is exactly $\mathcal{C}ontrol(\mathcal{A})$.

One direction is clear: every word w in $Control(\mathcal{A})$ is also in $\mathcal{SC}ontrol(\mathbf{A})$ and the largest clique of G_w is bounded by the size N of the active domain of the database witnessing the membership of w in $Control(\mathcal{A})$. Thus, w satisfies ψ.

To show the converse, consider $w = \{(q_n, \delta_n)\}_{n\geq 0}$ in $\mathcal{SC}ontrol(\mathbf{A})$ satisfying ψ, with N as the bound on the largest clique of G_w. We need to construct a finite database D and a run ρ of $\mathcal{A}$ over D whose control trace is w. We proceed in two stages. First, we construct from w a database D' and a run ρ' of $\mathbf{A}$ over D' whose control trace is w. Next, we modify D' and ρ' in order to obtain a database D and a run ρ over D that additionally satisfies the constraints in Σ.

We start with the case when σ contains no constants. We then explain how the proof can be modified in order to account for constants.

Assume now that σ contains no constant symbols and towards the first stage, let σ' be the signature extending the signature σ of $\mathbf{A}$ with two new relation symbols, R of arity k, and S of arity $2k$. For each σ-type δ over $\bar{x} \cup \bar{y}$, let $\pi_1(\delta)$ be the σ-type induced by δ on $\bar{x}$. From $\mathbf{A}$ we construct the following formula $\Psi_\mathbf{A}$:

$$\bigwedge_{(p,\delta,q)\in\Delta} \forall\bar{x}\Big[R(\bar{x}) \wedge \pi_1(\delta)(\bar{x})\Big] \rightarrow \Big[\exists\bar{y} S(\bar{x}\bar{y}) \wedge R(\bar{y}) \wedge \delta(\bar{x}\bar{y})\Big]$$
$$\wedge \bigwedge_{(p,\delta,q)\in\Delta} \exists\bar{x} R(\bar{x}) \wedge \pi_1(\delta)(\bar{x})$$

As our σ-types are quantifier-free formulas, $\Psi_\mathbf{A}$ is guarded in the sense of [4] hence it has the finite model property [17]. We now construct an infinite model I of $\Psi_\mathbf{A}$. For every transition (p, δ, q) of $\mathbf{A}$ we add to I a tuple $\bar{d}_\delta$ such that $I \models R(\bar{d}_\delta) \wedge \pi_1(\delta)(\bar{d}_\delta)$. Once this is done we construct the rest of I by chasing the formula $\Psi_\mathbf{A}$: whenever $I \models R(\bar{a}) \wedge \pi_1(\delta)(\bar{a})$ for some transition (p, δ, q), we add $\bar{b}$ such that $I \models S(\bar{a}\bar{b}) \wedge R(\bar{b}) \wedge \delta(\bar{a}\bar{b})$ by introducing fresh new elements as needed. By construction, $I \models \Psi_\mathbf{A}$. Because $\Psi_\mathbf{A}$ has the finite model property, there is a finite database I^* such that $I^* \models \Psi_\mathbf{A}$.

Let D' be the finite database obtained by restricting I^* to σ. We construct by induction a run $\rho' = \{(\bar{d}_n, q_n, \delta_n)\}_{n\geq 0}$ of $\mathbf{A}$ over D'. Let $\alpha_n = \pi_1(\delta_n)$ for $n \geq 0$. For the basis, set $\bar{d}_0$ to any tuple witnessing satisfaction of $\exists\bar{x}R(\bar{x}) \wedge \alpha_0(\bar{x})$ by I^*. For the induction step, let $n > 0$ and assume we have constructed $\bar{d}_0 \cdots \bar{d}_n$ such that $(\bar{d}_0, q_0, \delta_0) \cdots (\bar{d}_n, q_n, \delta_n)$ forms a valid prefix of a run of $\mathbf{A}$ over D', and $D' \models \alpha_n(\bar{d}_n)$. By construction, $I^* \models R(\bar{d}_n) \wedge \alpha_n(\bar{d}_n)$. Hence, there is a tuple $\bar{d}_{n+1}$ for which $I^* \models R(\bar{d}_{n+1}) \wedge S(\bar{d}_n\bar{d}_{n+1}) \wedge \delta_n(\bar{d}_n\bar{d}_{n+1})$. Clearly, $(\bar{d}_0, q_0, \delta_0) \cdots (\bar{d}_{n+1}, q_{n+1}, \delta_{n+1})$ forms a valid prefix of a run of $\mathbf{A}$ over D'.

Let $\rho' = \{(\bar{d}_n, q_n, \delta_n)\}_{n\geq 0}$. Clearly, ρ' is a run of $\mathbf{A}$ over D' and its control trace is w. This completes the first stage of the proof. Observe that this part of the proof involves $\mathbf{A}$ alone and is independent of the constraints of $\mathcal{A}$. As such, it provides an alternative (and simpler) proof of the result of [19] that $Control(\mathbf{A})$ equals $\mathcal{SC}ontrol(\mathbf{A})$ (and is therefore ω-regular) for every register automaton $\mathbf{A}$.

In the second stage, we modify the database D' and the run ρ' obtained above in order to enforce satisfaction of Σ. To this end, we use several properties of the graph G_w constructed earlier. Since G_w is represented by an MSO formula over an infinite string, it has bounded clique-width (see for example Theorem 7.36 in [11]). Like any graph of bounded clique-width, G_w is χ-bounded [15], i.e. its chromatic number is bounded by a function of its clique number. In our case this means that it is $h(N)$-colorable for some function h which is explicit in the proof of [15]. Let g be an $h(N)$-coloring function of G_w, associating to each vertex of G_w a positive integer less or equal to $h(N)$ such that no vertices connected by an edge have the same associated integer. We extend g to all equivalence classes of $\sim_w$ as follows. If a class ϵ is in the active domain but not occurring in an edge of G_w, let $g(\epsilon) = 0$. If ϵ is not in the active domain assign to it an arbitrary unique integer larger than $h(N)$.

We now construct a run $\rho = \{(\bar{e}_n, q_n, \delta_n)\}_{n\geq 0}$ and a database D such that ρ is a run of $\mathbf{A}$ over D with the same control trace as ρ' but additionally satisfying Σ. Intuitively, ρ and D are obtained by appropriately coloring data values in order to enforce inequalities.

To any position x and register i, set $\bar{e}_x[i]$ as $(\bar{d}_x[i], g([(x, i)]_w)$. This defines ρ. We construct D from D' as follows: whenever $R(d_1, \cdots, d_l)$ is a fact of D', then $R((d_1, \alpha_1), \cdots, (d_l, \alpha_l))$ are facts of D for all possible values of $\alpha_i \in [\![h(N)]\!]$.

We first claim that ρ is a run of $\mathbf{A}$ over D. To see this it is enough to verify that for each position x, $D \models \delta_x(\bar{e}_x, \bar{e}_{x+1})$. This is immediate from the construction and the fact that $D' \models \delta_x(\bar{d}_x, \bar{d}_{x+1})$.

We now claim that ρ verifies all constraints of Σ. Assume $w_n \cdots w_m \in e^{\neq}_{ij}$. Then either $([(n, i)]_w, [(m, j)]_w)$ is an edge of G_w, or at least one of ($[(n, i)]_w$ and $[(m, j)]_w$) is not in the active domain. In either case, $g([(n, i)]_w) \neq g([(m, j)]_w)$ and $\bar{e}_n[i] \neq \bar{e}_m[j]$ as desired.

This completes the proof for the case when σ contains no constants. We now explain how to extend it when constants are present. Recall that from any symbolic control trace w such that the associated graph has a bound N on its cliques, we need to construct a finite database D and a run ρ of $\mathcal{A}$ over D whose control trace is w.

We proceed as above to construct D with minor modifications in order to cope with the constants. We modify the formula $\Psi_\mathbf{A}$ as follows. First we restrict the conjuncts to those transitions containing only types δ occurring in w. Next, we add at the beginning of the formula an existential quantification $\exists\bar{z}$ where $\bar{z}$ has arity l, the number of constant symbols, and replace any occurrence of the constant c_i by z_i. Finally the schema is modified by eliminating the constants and adding l to the arity of all relational symbols. Moreover, each atom $R(\bar{u})$ is replaced by $R(\bar{u}, \bar{z})$. Observe that by definition of symbolic control trace with constants, the isomorphism type of $\bar{z}$ is the same throughout the trace. It then follows that $\Psi_\mathbf{A}$ has an infinite model, constructed as above, where in addition the substructure induced on the constant symbols is induced by the type of $\bar{z}$.

Using the finite model property we therefore have a finite model I^* for $\Psi_\mathbf{A}$. The database D' is constructed from I^* in the obvious way: fix arbitrarily a tuple $\bar{z}$ making the formula true and associate the constant symbols with $\bar{z}$. For each symbol R of σ, the relation R in D' consists of the tuples $\bar{u}$ such that $R(\bar{u}, \bar{z})$ is a fact of I^* for the chosen $\bar{z}$.

As above, it is straightforward to verify that the database D' yields a run of $\mathbf{A}$ whose control trace is w. The remaining part of the proof proceeds as in the case without constants, by combining D' with N in order to obtain the desired database D and run ρ of

$\mathcal{A}$ that has control trace w and additionally satisfies the inequality constraints. □

As noted above, the proof of Theorem 9 includes an alternative, simpler proof of the result of [19] that for register automata, with no global constraints, the control trace is ω-regular.

It follows from Theorem 9 that the emptiness problem is decidable for extended register automata. Indeed the proof of Theorem 9 is constructive and satisfiability of the corresponding formula is decidable [5].

Corollary 10. *Given an extended register automaton $\mathcal{A}$, it is decidable whether there exists a finite database D and an infinite run ρ of $\mathcal{A}$ over D.*

Verification of extended automata

We next briefly discuss how the quasi-regularity of the control traces of extended register automata can be used to show decidability of LTL-FO properties of such automata [1], generalizing previous results on verification of register automata (aka artifact systems) [12, 14]. We begin by reviewing the temporal language LTL-FO. First, LTL (linear-time temporal logic) is propositional logic augmented with temporal operators **G** (always), **F** (eventually), **X** (next) and **U** (until) (e.g., see [24]). Informally, **G**p says that p holds at all times in the run, **F**p says that p will eventually hold, **X**p says that p holds at the next configuration, and p **U** q says that q will hold at some point in the future and p holds up to that point. For example, $\mathbf{G}(p \rightarrow \mathbf{F}q)$ says that whenever p holds, q must hold sometime in the future.

LTL-FO is an extension of LTL obtained by interpreting propositions with quantifier-free FO statements over the database schema σ. The statements use the variables $\bar{x}$ and $\bar{y}$ referring to consecutive registers, and in addition may use *global* variables, shared by different statements and allowing to refer to other values across the run. The global variables are universally quantified over the entire formula.

Definition 11. *Let σ be a relational signature and $\bar{x}$, $\bar{y}$, $\bar{z}$ be tuples of distinct variables, where $\bar{x}$ and $\bar{y}$ have arity k. An LTL-FO sentence is an expression $\forall\bar{z}\varphi_f$, where (i) φ is an LTL formula with propositions P, and (ii) f is a mapping from P to quantifier-free FO formulas over σ using variables in $\bar{x}\bar{y}\bar{z}$.*

The semantics of an LTL-FO sentence $\xi = \forall\bar{z}\varphi_f$ is defined as follows. Let $\mathcal{A}$ be a extended automaton with k registers. Let $\rho = \{(\bar{d}_n, q_n, \delta_n)\}_{n\geq 0}$ be a run of $\mathcal{A}$ on database D. Let μ be a valuation of $\bar{z}$ into $\mathbb{D}$. An FO formula $\psi(\bar{x}, \bar{y}, \bar{z})$ is satisfied at position i with valuation μ if $D \models \psi(\bar{d}_i, \bar{d}_{i+1}, \mu(\bar{z}))$. The run ρ satisfies φ_f with valuation μ if $\{\sigma_i\}_{i\geq 0} \models \varphi$, where σ_i is the truth assignment for P in which p is true iff $f(p)$ is satisfied at position i with valuation μ. Finally, $\rho \models \forall\bar{z}\varphi_f$ if $\rho \models \varphi_f$ with every valuation μ of $\bar{z}$ into $\mathbb{D}$. We say $\mathcal{A}$ satisfies an LTL-FO sentence ξ, denoted $\mathcal{A} \models \xi$, if $\rho \models \xi$ for every run ρ of $\mathcal{A}$.

Note that, for the purpose of verification, the global variables can be easily eliminated from an LTL formula $\forall\bar{z}\varphi_f$. Indeed, the global variables $\bar{z}$ can be simulated by adding $|\bar{z}|$ registers that are propagated at each transition (so the value of each such register remains constant throughout a run). Thus, each run provides a valuation for $\bar{z}$ and the new automaton satisfies φ_f iff the initial one satisfies $\forall\bar{z}\varphi_f$. We assume from here on that LTL-FO formulas have no global variables.

Observe that in a complete automaton, the control trace of a run $\rho = \{(\bar{d}_n, q_n, \delta_n)\}_{n\geq 0}$ provides sufficient information to determine if the run satisfies φ_f. This is because at each position i, δ_i provides the complete type of $\bar{x} \cup \bar{y}$. This also allows extending the semantics of φ_f from runs to control traces in the obvious way. Clearly, satisfaction of φ_f by a control trace can be defined by an MSO formula $\alpha(\varphi_f)$. Let β be the quasi-regular formula defining $Control(\mathcal{A})$. Since MSO is closed under complement and quasi-regular languages are closed under intersection, $\beta \wedge \neg\alpha(\varphi_f)$ defines a quasi-regular language, which is empty iff $\mathcal{A} \models \varphi_f$. Thus, we have:

Theorem 12. *It is decidable, given an extended register automaton $\mathcal{A}$ and an LTL-FO formula φ_f for $\mathcal{A}$, whether $\mathcal{A} \models \varphi_f$.*

The precise complexity of the decision problem mentioned in Theorem 12 is open. The current proof of Theorem 9 uses MSO logic in order to express the fact that there is a bound on the size of all cliques in a given graph. This clearly does not yield optimal complexity. An obvious way to get lower complexity would be to use an automaton of small size for expressing the same fact. This is left for future work.

4 PROJECTIONS OF EXTENDED REGISTER AUTOMATA WITH NO DATABASE

Characterizing projections of general register automata turns out to be challenging. In this paper we focus, as a first step, on register automata without an underlying database. We will show that in this case, extended register automata are sufficient to specify projections. Moreover, they are themselves closed under projection. We discuss in Section 6 some of the challenges raised when a database is present.

In the remainder of the section, all (extended) automata are assumed to be without an underlying database and the relational signature is dropped from their definition.

We next show that extended register automata are closed under projection.

Theorem 13. *Let $\mathcal{A}$ be an extended register automaton with k registers and $m < k$. There is an extended register automaton $\mathcal{A}'$ with m registers such that $\mathcal{R}eg(\mathcal{A}') = \Pi_m(\mathcal{R}eg(\mathcal{A}))$.*

Proof. It is enough to prove the theorem for $m = k - 1$ as we can then project out the desired registers one by one.

Fix an extended register automaton $\mathcal{A} = (\mathbf{A}, \Sigma)$, where $\mathbf{A} = (k, Q, I, F, \Delta)$. We construct a new extended automaton $\mathcal{A}' = (\mathbf{A}', \Sigma')$ with $k - 1$ registers recognizing the projection of the traces of $\mathcal{A}$ on its first $k - 1$ registers. The general idea is that $\mathbf{A}'$ mimics the behavior of $\mathbf{A}$ on the remaining registers and Σ' collects all global constraints induced by the register k on the first $k - 1$ registers.

We can assume that $\mathcal{A}$ is complete and state-driven. We can also assume that its (in)equality constraints are consistent on all its control traces (otherwise, since this is clearly a regular property, we can intersect $\mathcal{A}$ with a Büchi automaton that accepts the control traces

[1]Here and in some previous work, LTL-FO uses only quantifier-free FO

on which $\mathcal{A}$ is consistent). For each equality type δ, let $\delta|_{(k-1)}$ be its restriction to the first $k-1$ registers. The construction of $\mathbf{A}'$ will require performing refinements of the states and transitions of $\mathbf{A}$ and modifications of its acceptance condition. Intuitively, this corresponds to taking the intersection of $\mathbf{A}$ with several automata. More precisely, we will construct $\mathbf{A}' = (k-1, Q', I', F', \Delta')$ for which there is a surjective mapping $\alpha : Q' \mapsto Q$ such that $(p', \delta', q') \in \Delta'$ iff $(\alpha(p), \delta, \alpha(q)) \in \Delta$ and $\delta' = \delta|_{k-1}$, $\alpha(I') = I$, and $w \in (Q')^\omega$ is accepted by $\mathbf{A}'$ iff $\alpha(w)$ is accepted by $\mathbf{A}$. In particular, $\mathcal{S}tate(\mathbf{A})$ is a homomorphic image of $\mathcal{S}tate(\mathbf{A}')$. Observe that $\mathbf{A}'$ is still complete and state-driven (so its control and state traces are interchangeable). We will define Σ' so that:

(†) for each state trace w of $\mathcal{A}'$, $\sim_w^{\mathcal{A}'}$ is the restriction of $\sim_{\alpha(w)}^{\mathcal{A}}$ to the first $k-1$ registers, and similarly for $\not\approx_w^{\mathcal{A}'}$ and $\not\approx_{\alpha(w)}^{\mathcal{A}}$.

where $\mathcal{A}' = (\mathbf{A}', \Sigma')$. Suppose we have defined such Σ'. We claim that $\Pi_{k-1}(\mathcal{R}eg(\mathcal{A})) = \mathcal{R}eg(\mathcal{A}')$. The inclusion $\Pi_{k-1}(\mathcal{R}eg(\mathcal{A})) \subseteq \mathcal{R}eg(\mathcal{A}')$ is immediate. Consider the converse. Let $\rho' = \{(\bar{d}'_n, q_n, \delta_n|_{(k-1)})\}_{n\geq 0}$ be a run of $\mathcal{A}'$. Let w be the state trace of ρ'. For each equivalence class ϵ' of $\sim_w^{\mathcal{A}'}$, let ϵ be the equivalence class of $\sim_{\alpha(w)}^{\mathcal{A}}$ containing it. By (†), for all $\epsilon'_1, \epsilon'_2 \in \mathcal{E}^w_{\mathcal{A}'}$, $\epsilon'_1 \not\approx_w^{\mathcal{A}'} \epsilon'_2$ iff $\epsilon_1 \not\approx_{\alpha(w)}^{\mathcal{A}} \epsilon_2$. We define a mapping $f : \mathcal{E}^{\alpha(w)}_{\mathcal{A}} \mapsto \mathbb{D}$ by $f(\epsilon) = \bar{d}_n[i]$ if $(n, i) \in \epsilon$ for some position n and some $i < k$, and $f(\epsilon)$ is a new fresh value if ϵ contains no class of $\mathcal{E}^w_{\mathcal{A}'}$. By (†), f is well defined. For every position x, we define $e_x[k]$ to equal $f([(x,k)]^{\mathcal{A}}_{\alpha(w)})$. Let $\rho = \{(\bar{d}_n, \alpha(q_n), \delta_n)\}_{n\geq 0}$ where $\bar{d}_n = (\bar{d}'_n, e_n)$. It is immediate to verify that ρ is a run of $\mathcal{A}$. By construction, the register trace of ρ' and ρ agree on the first $k-1$ registers.

We now show how to construct $\mathbf{A}'$ and Σ' satisfying (†). Let $w \in Q^\omega$ and recall that a state uniquely determine the type. Consider the reflexive, symmetric, transitive closure of the equality constraints induced by Σ and Δ on w. More precisely, denote $w \models e^{=}_{ij}(x,y)$ if $x \leq y$ and the sequence of states from x to y is in $e^{=}_{ij}$, and $w \models \delta^{=}_{ij}(x)$ if δ_x states that $x_i = x_j$, and $w \models \delta^{=}_{ij}(x, x+1)$ if δ_x states that $x_i = y_j$. The inequality counterparts $w \models e^{\neq}_{ij}(x,y)$, $w \models \delta^{\neq}_{ij}(x)$, and $w \models \delta^{\neq}_{ij}(x, x+1)$ are defined similarly. Let

$$\begin{aligned}\psi^{=}((x,i),(y,j)) =& (x \leq y \wedge e^{=}_{ij}(x,y)) \vee (y \leq x \wedge e^{=}_{ji}(y,x)) \\ &\vee (x = y \wedge (\delta^{=}_{ij}(x) \vee i = j)) \\ &\vee (y = x+1 \wedge \delta^{=}_{ij}(x, x+1))\end{aligned}$$

We can interpret $\psi^{=}((x,i),(y,j))$ over the string whose positions are pairs (x,i) ($x \geq 0, i \in [\![k]\!]$) in lexicographic order. Clearly, $\psi^{=}$ is definable in MSO over such strings, and so is its transitive closure $\psi^{=}_{*}$. For fixed i and j, let $\varphi^{=}_{ij}(x,y) = \psi^{=}_{*}((x,i),(y,j)) \wedge x \leq y$ interpreted on Q^ω. We will show shortly how to extract from this formula the desired regular expression $e^{=}_{ij}$ for all i, j. We first develop an analogous formula for inequalities. Consider the formula

$$\begin{aligned}\varphi^{\neq}_{ij}(x,y) \quad =& \bigvee\nolimits_{l,l' \in [\![k]\!]} \exists u \exists v (\varphi^{=}_{il}(x,u) \wedge e^{\neq}_{ll'}(u,v) \wedge \varphi^{=}_{l'j}(v,y)) \\ &\vee \exists u (\varphi^{=}_{il}(x,u) \wedge \delta^{\neq}_{ll'}(u) \wedge \varphi^{=}_{l'j}(u,y)) \\ &\vee \exists u (\varphi^{=}_{il}(x,u) \wedge \delta^{\neq}_{ll'}(u,u+1) \wedge \varphi^{=}_{l'j}(u+1,y))\end{aligned}$$

Intuitively, $\varphi^{\neq}_{ij}(x,y)$ specifies all inequalities implied by the initial inequalities together with the equalities.

We next show how to extract the regular expressions $e^{=}_{ij}$ and $e^{\neq}_{ij}$ from the formulas $\varphi^{=}_{ij}(x,y)$ and $\varphi^{\neq}_{ij}(x,y)$. To this end, we use the following "folklore" lemma, that is implicit in Büchi's initial proof that MSO characterizes ω-regular languages [7].

Lemma 14. *Given an MSO formula $\varphi(x,y)$ over Q^ω there exists a Büchi automaton B reading symbols from Q and having Q' as set of states, together with a finite-state automaton C reading symbols from Q', such that for every word $w \in Q^\omega$ and positions $x \leq y$ of w we have: $w \models \varphi(x,y)$ iff there is an accepting run of B on w such that if w' is the corresponding sequence of states of B then the segment of w' between positions x and y is a word accepted by C.*

By applying Lemma 14 concomittantly to all $\varphi^{=}_{ij}(x,y)$ and $\varphi^{\neq}_{ij}(x,y)$ and combining the results, we can refine the states and transitions of $\mathbf{A}$ using the automata B given by the lemma and use the automata C for specifying each of the regular expressions $e^{=}_{ij}$ and $e^{\neq}_{ij}$.

Let Σ' consist of the $e^{=}_{ij}$ and $e^{\neq}_{ij}$ obtained above. It is straightforward to verify that the resulting extended automaton satisfies (†). □

Since register automata are special cases of extended automata, Theorem 13 shows that extended automata are powerful enough to specify projections of register automata, as desired. But are general extended register automata unnecessarily powerful for describing projections of register automata? If so, what is the subclass of extended automata that captures precisely projections of register automata? We answer these questions next.

5 LR-BOUNDEDNESS

As in the previous section, the automata we consider here do not access a database, i.e. their database schema is empty. As will be seen in Example 17, even in this limited setting, extended register automata can produce register traces that are *not* the projections of the register traces of a register automaton. We now present a subclass of extended register automata that characterizes precisely projections of register automata. As mentioned earlier, being the projection of a register automaton is desirable because it means, intuitively, that the global constraints can be enforced entirely by local transitions, in a streaming fashion, at the cost of additional registers.

Before stating our characterization we need some further notation. We say that two extended automata $\mathcal{A}$ and $\mathcal{A}'$ with no database are *register-trace equivalent* if $\mathcal{R}eg(\mathcal{A}) = \mathcal{R}eg(\mathcal{A}')$. While not important for this paper, we note that register-trace equivalence is undecidable already for 1-register extended automata without a database. Indeed, it is shown in [22] that given a $(k+1)$-register automaton $\mathbf{A}$ without a database (additionally equipped with accepting states) it is undecidable whether the set of finite prefixes of $\Pi_1(\mathcal{R}eg(\mathbf{A}))$ contains all words in $\mathbb{D}^*$. This can be easily adapted to show that it is undecidable in our model whether $\Pi_1(\mathcal{R}eg(\mathbf{A}))$ contains all infinite words in $\mathbb{D}^\omega$. But as shown in Theorem 13, $\Pi_1(\mathcal{R}eg(\mathbf{A}))$ can be specified using an extended register automaton without a database, using one register. The undecidablity of register-trace equivalence then follows.

Let $\mathcal{A}$ be an extended automaton with k registers. Let $w \in \mathcal{C}ontrol(\mathcal{A})$. For a position h in w, let $L(h) = \{h' \mid h' \leq h\}$ and $R(h) = \{h' \mid h' > h\}$ be the set of positions of w to the left or to the right of h. For a class $\epsilon \in \mathcal{E}^w_{\mathcal{A}}$ and $h \geq 0$ we denote $\epsilon \sqsubseteq L(h)$ if $\{n \mid (n,i) \in \epsilon, i \in [\![k]\!]\} \subseteq L(h)$ and $\epsilon \sqsubseteq R(h)$ if $\{n \mid (n,i) \in \epsilon, i \in [\![k]\!]\} \subseteq R(h)$. If $\epsilon \not\sqsubseteq L(h)$ and $\epsilon \not\sqsubseteq R(h)$, we say that ϵ *straddles* h. By a simple pigeonhole argument, the number of such ϵ is at most $\|\Sigma\|$ where $\|\Sigma\|$ is the total number of states of all the automata describing all the regular expressions in Σ.

Let G^w_h be the following graph. Its set of nodes is $\mathcal{E}^w_{\mathcal{A}}$. The edges are as follows. For $\epsilon_1, \epsilon_2 \in \mathcal{E}^w_{\mathcal{A}}$ such that $\epsilon_1 \sqsubseteq L(h)$ and $\epsilon_2 \sqsubseteq R(h)$, there is an edge between ϵ_1 and ϵ_2 iff $\epsilon_1 \not\approx_w \epsilon_2$.

The graph G^w_h describes all inequality constraints of Σ that must be enforced at position h in w. Intuitively, without constraints and finitely many registers, we have only finite memory and therefore can enforce only finitely many such edges. However, if several edges share the same endpoint, the same register can be used for all of them. Hence a relevant parameter is the size of the vertex cover of G^w_h. This justifies the next definition. Note that we do not need to consider equality constraints as those can always be simulated with extra registers.

Definition 15. *An extended register automaton $\mathcal{A}$ is LR-bounded if there exists $N > 0$ such that, for every $w \in Control(\mathcal{A})$ and $h \geq 0$, the graph G^w_h has a vertex cover of size at most N.*

Observe that LR-boundedness is a syntactic notion, not preserved under register-trace equivalence. That is, an LR-bounded extended register automaton $\mathcal{A}$ may be register-trace equivalent to an extended automaton $\mathcal{A}'$ that is not LR-bounded, as shown next.

Example 16. *Let $\mathcal{A} = (\mathbf{A}, \Sigma)$ be a automaton with one register, no constraints (i.e $\Sigma = \emptyset$), no database, one state q and one transition (q, δ, q), where δ is the equality type $x_1 \neq y_1$ enforcing that the data value changes at each step. Clearly, $\mathcal{A}$ is LR-bounded since for all $w \in Control(\mathcal{A})$ and all positions h of w, G^w_h has a single edge conecting the class of the current positions to the one of the next position. Now consider the automaton $\mathcal{A}' = (\mathbf{A}', \Sigma')$ where $\mathbf{A}'$ has two states p and q and two transitions (q, δ, q), (p, δ, p) with the same equality type δ as above, together with a constraint $e^{\neq}_{11} \in \Sigma'$ defined by the regular expression p^+. A run of $\mathcal{A}'$ starting with state p will generate a register trace where all values are pairwise distinct. But this register trace is also a trace of a run of $\mathcal{A}'$ starting with state q, which in turn is also a register trace of $\mathcal{A}$. Hence $\mathcal{R}eg(\mathcal{A}) = \mathcal{R}eg(\mathcal{A}')$. However $\mathcal{A}'$ is not LR-bounded as in a position h of a control trace of $\mathcal{A}'$ starting with state p, all positions to the left of h are connected to all positions to the right of h.*

We will see that LR-boundedness characterizes the projections of register automata up to register-trace equivalence. We now show an example of an extended register automaton that is *not* register-equivalent to any LR-bounded register automaton. In view of Theorem 19, this extended automaton cannot be simulated by any register automaton, even with additional registers.

Example 17. *Consider again the extended automaton in Example 7. The automaton has one register, one state with a trivial looping transition and a global inequality constraint ensuring that all data values stores in the register are pairwise distinct. Suppose that there is an LR-bounded automaton $\mathcal{B}$ with the same register traces. Let N be the bound witnessing LR-boundedness. Consider a run ρ of $\mathcal{B}$ with all values of its register pairwise distinct. Consider the situation at position $N + 2$. Since $\mathcal{B}$ is LR-bounded there are positions x, y such $x < N + 2 < y$ such that there is no edge between x and y in G_{N+2}. It is then straightforward to verify that the run ρ' formed from ρ by identifying the values at position x and y is still a run of $\mathcal{B}$, a contradiction.*

Before proceeding, we show that LR-boundedness is a decidable property.

Theorem 18. *It is decidable whether an extended register automaton $\mathcal{A}$ with no access to the database is LR-bounded.*

Proof. Let $\mathcal{A} = (\mathbf{A}, \Sigma)$ be an extended register automaton with k registers. We need to test whether there exists a bound N on the size of vertex covers of all the G^w_h for all traces $w \in Control(\mathcal{A})$ and positions h of w.

Note first that in the absence of a database, the set of control traces of $\mathcal{A}$ forms an ω-regular language. This follows from the construction in Theorem 13, that yields a finite-state automaton after projecting away all registers.

Also observe that for each $w \in Control(\mathcal{A})$, each equivalence class $\epsilon \in \mathcal{E}^w_{\mathcal{A}}$ can be uniquely represented by the lexicographically minimum pair (n, i) such that register i contains an element of the class in position n. Moreover, for each $i \in [\![k]\!]$ there is an MSO formula $\alpha_i(x)$ such that $w \models \alpha_i(n)$ iff (n, i) is the representative of some equivalence class. Using these MSO formulas one can obtain MSO formulas $\phi_{ij}(z, x, y)$ such that for each $w \in Control(\mathcal{A})$ and each position h of w, $w \models \phi_{ij}(h, n, m)$ iff (n, i) and (j, m) are representatives of classes forming an edge in G^w_h. As we can also express in MSO the fact that a set S is a vertex cover of some graph, we are left to decide statements of the form:

$\exists N$ *such that for all words w from a given ω-regular language,*

$$\forall x \exists S \varphi(x, S) \wedge |S| \leq N$$

where φ is in MSO. It turns out that such statements belong to a class of MSO formulas known to be decidable [10]. □

We are now ready to state the main result of this section.

Theorem 19. *Let $\mathcal{B}$ be an extended automaton with m registers. $\mathcal{R}eg(\mathcal{B})$ equals $\Pi_m(\mathcal{R}eg(\mathbf{A}))$ for some register automaton $\mathbf{A}$ iff $\mathcal{B}$ is register-trace equivalent to some LR-bounded extended automaton.*

Since the proof is rather involved, we present separately the "only-if" part and the "if" part. We start with the "only if" part of Theorem 19.

Proposition 20. *Let $\mathcal{B}$ be an extended automaton with m registers. If $\mathcal{R}eg(\mathcal{B})$ equals $\Pi_m(\mathcal{R}eg(\mathbf{A}))$ for some register automaton $\mathbf{A}$ then $\mathcal{B}$ is register-trace equivalent to some LR-bounded extended automaton.*

Proof. We exhibit an LR-bounded extended register automaton $\mathcal{A}$ such that $\Pi_m(\mathcal{R}eg(\mathbf{A})) = \mathcal{R}eg(\mathcal{A})$. Without loss of generality we assume that $\mathbf{A}$ is complete and state-driven. Let k be the number of registers of $\mathbf{A}$ and let $m < k$. We start with a few preliminaries. Let $\mathcal{A}$ be an extended register automaton. We say that a mapping f from $\mathcal{E}^w_{\mathcal{A}}$ to $\mathbb{D}$ is consistent with w in $\mathcal{A}$ if for all classes $\epsilon_1, \epsilon_2 \in \mathcal{E}^w_{\mathcal{A}}$, if $\epsilon_1 \not\approx_w \epsilon_2$ then $f(\epsilon_1) \neq f(\epsilon_2)$. The following lemma will provide the global constraints used in the construction of $\mathcal{A}$.

Lemma 21. *Let $\mathbf{A} = (k, Q, I, F, \Delta)$ be a complete and state-driven register automaton. For all $i, j \leq k$ there exist regular expressions*

$e^{=}_{ij}$ and $e^{\neq}_{ij}$ such that for all state traces w of $\mathbf{A}$, positions $a \leq b$ and registers $i, j \in [\![k]\!]$, $(a, i) \sim^{w}_{\mathbf{A}} (b, j)$ iff the factor of w between position a and b is in $e^{=}_{ij}$ and $[(a, i)]_w \not\approx_w [(b, j)]_w$ iff the same factor is in $e^{\neq}_{ij}$.

Proof. Let $i, j \in [\![k]\!]$. We construct a finite state automaton A for $e^{=}_{ij}$. At any time during its run, A simulates $\mathbf{A}$ and remembers in its state the set of registers whose value in the current position must be equal to that of register i at the start of the run. The states of A are the subsets of $[\![k]\!]$. The initial state of A contains only i. For a state q of $\mathbf{A}$ with δ the equality type induced by q, A has a transition $(S, q, \delta(S))$, where $\delta(S) = \{m \mid \exists l \in S\ x_l = y_m \in \delta\}$. The accepting states of A are all those containing j. It is immediate to check that A has the desired properties.

We proceed similarly for $e^{\neq}_{ij}$. It is immediate to see that $[(a, i)] \not\approx [(b, j)]$ iff there is a position c between a and b such that:

- $[(a, i)] \sim [(c, l)]$, $[(c + 1, m)] \sim [(b, j)]$, and the state q at position c induces an equality type δ containing $x_l \neq y_m$, or
- $[(a, i)] \sim [(c, l)]$, $[(c, m)] \sim [(b, j)]$, and the state q at position c induces an equality type δ containing $x_l \neq x_m$.

This property is easily checkable by a finite state automaton. □

Given an equality type δ, let $\delta|m$ be δ restricted to registers $[1..m]$. Let Σ consist of $e^{=}_{ij}$ and $e^{\neq}_{ij}$ for $i, j \in [\![m]\!]$ as given by Lemma 21. Let $\mathcal{A}$ be the extended register automaton $(\mathbf{A}', \Sigma)$ where $\mathbf{A}' = (m, Q, I, F, \Delta')$ and $\Delta' = \{(q, \delta|m, q') \mid (q, \delta, q') \in \Delta\}$. Notice that as $\mathbf{A}$ is state-driven, so is $\mathcal{A}$. We first show that $\mathcal{R}eg(\mathcal{A}) = \Pi_m(\mathcal{R}eg(\mathbf{A}))$.

The inclusion $\Pi_m(\mathcal{R}eg(\mathbf{A})) \subseteq \mathcal{R}eg(\mathcal{A})$ is immediate. Consider the converse, $\mathcal{R}eg(\mathcal{A}) \subseteq \Pi_m(\mathcal{R}eg(\mathbf{A}))$. Let $\rho = \{(\bar{d}'_n, q_n, \delta_n|m)\}_{n\geq 0}$ be a run of $\mathcal{A}$ and w be its state trace. By construction w is also a state trace of $\mathbf{A}$. In view of Lemma 21, $\sim^{w}_{\mathcal{A}}$ is the restriction of $\sim^{w}_{\mathbf{A}}$ to $[1..m]$. For each equivalence class ϵ' of $\sim^{w}_{\mathcal{A}}$, let ϵ be the equivalence class of $\sim^{w}_{\mathbf{A}}$ containing it. From the same lemma it also follows that for all $\epsilon'_1, \epsilon'_2 \in \mathcal{E}^{w}_{\mathcal{A}}$, $\epsilon'_1 \not\approx_w \epsilon'_2$ in $\mathcal{A}$ iff $\epsilon_1 \not\approx_w \epsilon_2$ in $\mathbf{A}$. Consider the function f' from $\mathcal{E}^{w}_{\mathcal{A}}$ to $\mathbb{D}$ associating $\bar{d}'_n[i]$ to the class $[(n, i)]^{\mathcal{A}}_w$ for every $n \geq 0$ and $i \in [\![m]\!]$. This function is clearly consistent with w in $\mathcal{A}$. Let f be a function from $\mathcal{E}^{w}_{\mathbf{A}}$ to $\mathbb{D}$ that maps every ϵ that contains an equivalence class ϵ' of $\sim^{w}_{\mathcal{A}}$ to $f'(\epsilon')$ and every other ϵ to a fresh new data value. Clearly, f is consistent with w in $\mathbf{A}$. Let $\rho' = \{(\bar{d}_n, q_n, \delta_n)\}_{n\geq 0}$ where $\bar{d}_n[i] = f([(n, i)]^{\mathbf{A}}_w)$. It is easy to verify that ρ' is a run of $\mathbf{A}$ and by construction the register trace of ρ is the projection of the register trace of ρ' on $[1..m]$.

We next show that $\mathcal{A}$ is LR-bounded. Let $w = \{(q_n, \delta_n)\}_{n\geq 0}$ be a control trace of $\mathcal{A}$ and denote $w_{ab} = (q_a, \delta_a) \ldots (q_b, \delta_b)$ for $0 \leq a \leq b$. Let $h \geq 0$. Let (ϵ_1, ϵ_2) be an edge in G^{w}_{h}. By Lemma 21, there exist a, b, $a \leq h < b$ such that $(a, i) \in \epsilon_1$, $(b, j) \in \epsilon_2$ for some $i, j \in [\![k]\!]$ and $w_{ab} \in e^{\neq}_{ij}$. From the definition of $\not\approx^{\mathbf{A}}_{w}$ in the case of register automata, it follows that there exists $l \in [\![k]\!]$ such that either (i) $[(a, i)]^{\mathbf{A}}_w \sim^{\mathbf{A}}_w [(h, l)]^{\mathbf{A}}_w$ and $[(h, l)]^{\mathbf{A}}_w \not\approx^{\mathbf{A}}_w [(b, j)]^{\mathbf{A}}_w$ or (ii) $[(a, i)]^{\mathbf{A}}_w \not\approx^{\mathbf{A}}_w [(h, l)]^{\mathbf{A}}_w$ and $[(h, l)]^{\mathbf{A}}_w \sim^{\mathbf{A}}_w [(b, j)]^{\mathbf{A}}_w$.

Let $\mathbf{C}$ consist of the classes $\epsilon_1 \sqsubseteq L(h)$ for which (i) holds, together with the classes $\epsilon_2 \sqsubseteq R(h)$ for which (ii) holds. Clearly, $\mathbf{C}$ is a vertex cover of G^{w}_{h} and its size is bounded by k. Thus, $\mathcal{A}$ is LR-bounded. □

We next prove the "if" part of Theorem 19.

Proposition 22. *Let $\mathcal{B}$ be an LR-bounded extended automaton with m registers. Then $\mathcal{R}eg(\mathcal{B})$ equals $\Pi_m(\mathcal{R}eg(\mathbf{A}))$ for some register automaton $\mathbf{A}$.*

Proof. Consider an LR-bounded register automaton $\mathcal{B} = (\mathbf{B}, \Sigma)$. Proposition 6 shows how global equality constraints can be simulated with additional registers, so we assume that Σ does not have global equality constraints of the form $e^{=}_{ij}$.

It remains to take care of the inequality constraints. For readability, we provide the proof for the case when $m = 1$ (so $\mathcal{B}$ has a single register). The extension to multiple registers is straightforward. Let N be the vertex cover bound witnessing the fact that $\mathcal{B}$ is LR-bounded.

We denote $e^{\neq}_{11}$ by $e^{\neq}$. We also denote the equivalence class $[(n, 1)]_w$ by $[n]_w$ for $n \geq 0$ (or simply $[n]$ when w is understood). We construct a register automaton $\mathbf{A}$ for which $\mathcal{R}eg(\mathcal{B}) = \Pi_1(\mathcal{R}eg(\mathbf{A}))$. The intuition is the following. Let $\{d_n\}_{n\geq 0}$ be a register trace of $\mathcal{B}$. A computation of $\mathbf{A}$ on $\{d_n\}_{n\geq 0}$ guesses an accompanying control trace $w = \{(q_n, \delta_n)\}_{n\geq 0}$ of $\mathcal{B}$ and first checks that $\rho = \{(d_n, q_n, \delta_n)\}_{n\geq 0}$ is a run of $\mathcal{B}$. Register 1 of $\mathbf{A}$ is treated in each transition identically to register 1 of $\mathcal{B}$. In addition, $\mathbf{A}$ checks satisfaction of the inequality constraints of Σ. To do this, extra registers are used to carry information, for every position $h \geq 0$, about the prefix $\{(d_n, q_n, \delta_n)\}_{n\leq h}$ and to guess information about the suffix $\{(d_n, q_n, \delta_n)\}_{n>h}$. At a given position that participates in inequality constraints, $\mathbf{A}$ is faced with two choices: either store the current data value in some register and then later check that it is indeed unequal to all the data values paired with the current position by $e^{\neq}$, or guess, and store in the registers, a set of data values unequal to the current one, and later check that all positions paired with the current one have a data value in the guessed set. The choice $\mathbf{A}$ will make at position h will depend on the out-degree of h in G^{w}_{h}: if it is larger than N then it will choose the first option, otherwise the second. LR-boundedness will guarantee that this strategy can be enforced using a number of registers depending only on N.

We next present more details. Recall that the construction is shown for the case when $\mathcal{B}$ has a single register and no global equality constraints (because these can be simulated with extra registers by Proposition 6). We assume without loss of generality a normal form on $e^{\neq}$: if $e^{\neq}$ connects a member n of an equivalence class ϵ_1 to a member m, $m > n$, of another equivalence class ϵ_2, then m is the first member of ϵ_2 that is larger than n and n is the last member of ϵ_1 that is smaller than m. Let $E^{\neq} = (Q^{\neq}, q^{\neq}_0, \alpha^{\neq}, F^{\neq})$ be a deterministic automaton accepting $e^{\neq}$, where $Q^{=(\neq)}$ is the set of states, $q^{=(\neq)}_0$ the initial state, $\alpha^{=(\neq)}$ the transition function, and $F^{=(\neq)}$ the set of accepting states.

For each $h \geq 0$, consider the graph $\bar{G}^{w}_{h}$ whose edges are all pairs (n, m) for which $w_{nm} \in e^{\neq}$ and $n \leq h, m > h$. Observe that such an edge exists iff there are ϵ_1, ϵ_2 such that $n \in \epsilon_1$, $m \in \epsilon_2$, and (i) (ϵ_1, ϵ_2) is an edge in G^{w}_{h}, or (ii) $\epsilon_1 \not\approx_w \epsilon_2$ and ϵ_1 or ϵ_2 straddles h. Also note that, due to the normal form for $e^{\neq}$, for each ϵ_1, ϵ_2 there is at most one corresponding edge in $\bar{G}^{w}_{h}$. Since there is at most one equivalence classes that can straddle h, it follows that $\bar{G}^{w}_{h}$ has a vertex cover of size at most $M = N + 1$.

Clearly, it is sufficient for **A** to enforce, for all $h \geq 0$ all inequalities represented by $\bar{G}^w_h$. To do so we use registers, with additional bookkeeping information in the states. We distinguish between two kinds of nodes in $\bar{G}^w_h$: (a) nodes that have a large out-degree and (b) nodes with a small out-degree. In the first case **A** will store the data value of the node of G^w_h in its registers while in the second case it will store the non-deterministically guessed data values of the target nodes. Fix $k > 0$ (a parameter of the construction). We use $2k$ registers, denoted $R_a = \{a_1, \ldots a_k\}$ and $R_b = \{b_1, \ldots b_k\}$. Registers in R_a hold values of nodes of type (a), and those in R_b hold values of nodes of type (b). For register r, we denote the value it holds by $val(r)$. We also use, as components of the state of **A**, mappings $on : R_a \cup R_b \mapsto \{0, 1\}$ (indicating, as earlier, whether r is "occupied" or "free"), and $state : R_a \cup R_b \mapsto Q^{\neq}$ providing, for each register, an associated state. For each $q \in Q^{\neq}$, we denote $reg_a(q) = \{r \in R_a \mid on(r) = 1, state(r) = q\}$ and similarly for $reg_b(q)$. For $q \in Q^{\neq}$, let $val\text{-}state_a(q)$ be the bag of values held by the registers in $reg_a(q)$, and similarly for $val\text{-}state_b(q)$. At position h, $val\text{-}state_a(q)$ holds the values of current nodes of type (a), such that for all $d \in val\text{-}state_a(q)$, if d was guessed at transition n, then $\alpha^{\neq}(q_0^{\neq}, w_{nh}) = q$ (the multiplicities are irrelevant). And $val\text{-}state_b(q)$ represents the bag of values such that, if $d \in val\text{-}state_b(q)$ is introduced at transition n, $\alpha^{\neq}(q_0^{\neq}, w_{nh}) = q$ and the multiplicity of d is the number of target nodes reachable from n that occur beyond h and have value d. We additionally maintain, as a component of the state of **A**, a set $Q^{\neg}_{\neq} \subseteq Q^{\neq}$ of states that are updated at each transition and are prohibited from ever reaching an accepting state.

In a computation of **A**, all registers are initially available (i.e., $on(r) = 0$ for all r). This is easily enforced by having **A** reject otherwise. In a transition from position $h - 1$ to h, **A** does the following:

(1) for each accepting state q of $Q^{\neq}$:
   (a) if $q \in Q^{\neg}_{\neq}$, reject;
   (b) if $d_h \in val\text{-}state_a(q)$, reject;
   (c) if $reg_b(q) \neq \emptyset$ and $d_h \notin val\text{-}state_b(q)$, reject; if $reg_b(q) \neq \emptyset$ and $d_h \in val\text{-}state_b(q)$, pick one $r \in reg_b(q)$ for which $val(r) = d_h$, and set $on(r) = 0$ (this decreases the multiplicity of d_h in $val\text{-}state_b(q)$).
   (d) guess whether there is another $\neq$-match from q in the future. For a negative guess, insert q in $Q^{\neg}_{\neq}$, and set $on(r) = 0$ for all $r \in reg_a(q)$. For a positive guess, either continue or do the following ("switch" from R_a to R_b):
   - if $val\text{-}state_b(q) \neq \emptyset$, do nothing;
   - if $val\text{-}state_b(q) = \emptyset$ pick a subset of registers r of R_b for which $on(r) = 0$ (if none exists, reject), set each $on(r) = 1$, $state(r) = q$, and set each $val(r)$ to an arbitrary value not in $val\text{-}state_a(q)$. Moreover, set $on(r) = 0$ for all $r \in reg_a(q)$;

(2) guess whether h yields at least one edge in $\bar{G}^w_h$. For a negative guess, add $q_0^{\neq}$ to $Q^{\neg}_{\neq}$. For a positive guess, nondeterministically choose one of the following:
   (i) pick a register $r \in R_a$ for which $on(r) = 0$ (if such exists) and set $state(r) = q_0^{\neq}$, $on(r) = 1$, and $val(r) = d_h$;
   (ii) if $val\text{-}state_b(q_0^{\neq}) \neq \emptyset$ and $d_h \notin val\text{-}state_b(q_0^{\neq})$, continue; if $val\text{-}state_b(q_0^{\neq}) = \emptyset$ pick a subset of registers r of R_b for which $on(r) = 0$ (if such exist), set each $on(r) = 1$, $state(r)$ to $q_0^{\neq}$, and set each $val(r)$ to an arbitrary value different from d_h.

(3) for $p \in Q^{\neq}$, let $pre(p, q_h) = \{q \in Q^{\neq} \mid \alpha^{\neq}(q, q_h) = p\}$. Reject if for some p and $q_1, q_2 \in pre(p, q_h)$, $val\text{-}state_b(q_1) \neq val\text{-}state_b(q_2) \neq \emptyset$. Otherwise, for each p choose an arbitrary $q \in pre(p, q_h)$ for which $val\text{-}state_b(q) \neq \emptyset$ (if such exists), and set $on(r) = 0$ for all $r \in reg_b(q')$ for $q' \in pre(p, q_h)$, $q' \neq q$ (this merges the bags corresponding to the states in $pre(p, q_h)$).

(4) for each register $r \in R_a \cup R_b$, advance its state according to $\alpha^{\neq}$ on input q_h; similarly, advance each state in $Q^{\neg}_{\neq}$.

Note that the construction of **A** is parameterized by k, which determines its register "budget". We claim that $\Pi_1(\mathcal{R}eg(\mathbf{A})) \subseteq \mathcal{R}eg(\mathcal{B})$ for *every* k, and $\mathcal{R}eg(\mathcal{B}) \subseteq \Pi_1(\mathcal{R}eg(\mathbf{A}))$ for some sufficiently large k.

Consider $\Pi_1(\mathcal{R}eg(\mathbf{A})) \subseteq \mathcal{R}eg(\mathcal{B})$. Let $\rho = \{(\bar{d}_n, \bar{q}_n, \bar{\delta}_n)\}_{n \geq 0}$ be a run of **A**. By construction, $\{(d_n, q_n, \delta_n)\}_{n \geq 0}$ is a run of **B**, where $\{d_n\}_{n \geq 0} = \Pi_1(\{\bar{d}_n\}_{n \geq 0})$ and $w = \{(q_n, \delta_n)\}_{n \geq 0}$ is the corresponding control trace of **B** generated by **A**. We need to show that $\{(d_n, q_n, \delta_n)\}_{n \geq 0}$ satisfies Σ. Note that at each transition from h to $h + 1$, **A** non-deterministically guesses whether h is the source of an edge of $\bar{G}^w_h$. In the case of an incorrect negative guess, the run is guaranteed to reject by reaching $Q^{\neg}_{\neq}$ containing an accepting state. If the guess is correct, the run rejects in these cases: (i) the constraint corresponding to the edge is violated, or (ii) the register budget is exceeded, or (iii) other incorrect guesses about the computation. Thus, an accepting run guarantees satisfaction of all constraints.

Now consider $\mathcal{R}eg(\mathcal{B}) \subseteq \Pi_1(\mathcal{R}eg(\mathbf{A}))$. Let $\{d_n\}_{n \geq 0}$ and $w = \{(q_n, \delta_n)\}_{n \geq 0}$ the control trace of $\mathcal{B}$ generated by **A**. We need to show that, given a sufficiently large k, there exists a computation of **A** that enforces all constraints of Σ on $\{(d_n, q_n, \delta_n)\}_{n \geq 0}$. We claim that such a computation is guaranteed to exist for $k > M^2$. Consider the computation in which the guesses at each transition from h to $h + 1$ are made according to the following strategy. Consider first the treatment of h ((2) above). If h yields no edge in $\bar{G}^w_h$, make the negative guess by inserting $q_0^{\neq}$ in $Q^{\neg}_{\neq}$. Otherwise, the positive guesses are made as follows ($deg(h)$ denotes the degree of h in $\bar{G}^w_h$):
(a) if $deg(h) > M$, take choice 2(i) (propagate d_h in a free register of R_a).
(b) if $deg(h) \leq M$, take choice 2(ii), by generating a bag of size $deg(h)$ containing the values of the targets nodes from h.

Now consider the choices in 1(d). Consider the values propagated in registers of R_a that reach an accepting state q. Let $deg_h(q)$ be the number edges from q at position h, i.e. the number of distinct words w_{hn} for which $\alpha^{\neq}(q, w_{hn}) \in F^{\neq}$ $(n > h)$. If $deg_h(q) > M$, continue. Otherwise, "switch" from R_a to R_b. If a new bag is generated, it contains the values of the target nodes from q at h (so its size is $deg_h(q)$).

With the above strategy, we claim that at each position h, (i) the number of occupied registers of R_a is at most M and (ii) the number of occupied registers in R_b is at most M^2. Let V be a vertex cover of $\bar{G}^w_h$ of size M. To see (i), note that for every $q \in Q^{\neq}$ for which $reg_a(q) \neq \emptyset$, $deg_h(q) > M$. This means that every value in R_a corresponds to a node in V. For (ii), note that every value in R_b

corresponds to a target node in $\bar{G}_h^w$ connected to a source node of degree $\leq M$. Clearly, the number of such target nodes is at most M^2 (if α is the number of such source nodes in V and β is the number of such target nodes in V, then we have $\alpha + \beta = M$ and the number of target nodes is less than $\beta + \alpha M$, hence at most M^2).

Altogether, the total number of registers used by $\mathbf{A}$ to enforce the constraints is $2 \cdot M^2 + 1$. □

This completes the proof of Theorem 19.

6 PROJECTION WITH A DATABASE

We have seen in the previous section that extended automata can specify the projections of register automata with no database. In this section we discuss some of the issues that arise when the register automaton is equipped with a database. As we shall see, describing projections becomes a much more a challenging problem. In particular, extended automata are no longer able to specify the projections.

Consider a register automaton with a database. One can consider various notions of projection. In addition to hiding some of the registers, one could project some of the database relations, or hide them altogether in the projected runs. Perhaps surprisingly, performing no projection at all on the database appears to be the most challenging option. We illustrate this next.

Example 23. *Consider a register automaton $\mathbf{A}$ with 2 registers and states p and q, with initial and final state p. Its database is a binary relation E representing edges in a graph together with a unary relation U. There are two transitions in $\mathbf{A}$: (p, δ, q) and (q, δ', p). The types δ and δ' enforce that the value of register 2 remains unchanged and that the value of register 1 is always in U, i.e. both types contain $x_2 = y_2 \wedge U(x_1)$. Moreover δ contains $E(x_2, x_1)$ while δ' contains $\neg E(x_2, x_1)$. Hence the projections of runs of $\mathbf{A}$ on register 1 consist of infinite sequences of nodes of the database for which there is a node in the graph that points exactly to every even position in the sequence. This is a property over the database that an extended register automaton cannot express. Indeed, assume towards a contradiction that there is an extended automaton $\mathcal{B}$ such that for all databases D, $\mathcal{R}eg(D, \mathcal{B}) = \Pi_1(\mathcal{R}eg(D, \mathbf{A}))$. Let D be the database consisting of the edge $E(c, d_0)$ and the facts $U(d_0), U(d_1)$, where c, d_0, and d_1 are distinct values. Then the sequence $d_0 d_1 d_0 d_1 d_0 d_1 \cdots$ is in $\Pi_1(\mathcal{R}eg(D, \mathbf{A}))$. Hence there is an accepting run ρ of $\mathcal{B}$ over D whose register trace is the above sequence. Observe that no type in a transition of the run includes a positive atom $E(-,-)$ since such an atom would be false (as the only edge is $E(c, d_0)$ and $c \neq d_0, d_1$). Now consider the database D' which is the same as D with the edge $E(c, d_0)$ removed. Then ρ remains an accepting run of $\mathcal{B}$ over D', since by the earlier observation, all types at each step remain true. But the sequence is not in $\Pi_1(\mathcal{R}eg(D', \mathbf{A}))$, contradiction.*

The above example is relatively simple but already suggests that describing projections of runs may require specifying non-trivial combinatorial properties involving both the register trace and the database. As seen above, extended automata are unable to describe such projections.

In this section we make partial progress by considering the special case of projections in which the entire database is hidden, together with some of the registers. Consider a register automaton $\mathbf{A} = (k, \sigma, Q, I, F, \Delta)$, and $m \leq k$. We denote by $\Pi_m(\mathcal{R}eg(D, \mathbf{A}))$ the projection of $\mathcal{R}eg(D, \mathbf{A})$ retaining only the values of registers $[1..m]$. We would like to describe these projections as the runs of some register automaton without a database. More precisely, we are looking for an automaton $\mathcal{A}$ with no database such that: (i) for each database D over σ and run of $\mathbf{A}$ over D whose register trace is r, there is some run of $\mathcal{A}$ whose register trace is $\Pi_m(r)$, and (ii) conversely, for each run of $\mathcal{A}$ whose register trace is r', there is a database D over σ and a run of $\mathbf{A}$ over D whose register trace is r and $r' = \Pi_m(r)$. In other words, we wish to have that $\mathcal{R}eg(\mathcal{A}) = \bigcup_D \Pi_m(\mathcal{R}eg(D, \mathbf{A}))$. To this end, we augment extended automata with two new kinds of global constraints: finiteness contraints and tuple inequality constraints.

Finiteness constraints. Consider again the automaton $\mathbf{A}$ of Example 23. We wish to describe the sequences of values in register 1 when the database is hidden. First, the data values occurring at even and odd positions must be disjoint. This can be enforeced by an inequality constraint $e^{\neq}_{11}$ on values at odd distance in the run. However, this is not sufficient, as it allows for infinitely many data values in the run. In order to deal with this problem, we introduce finiteness constraints defined by MSO formulas $\varphi_i^{\text{fin}}(x)$, where x is a free variable. A run $\{(\bar{d}_n, q_n, \delta_n)\}_{n\geq 0}$ of a register automaton satisfies $\varphi_i^{\text{fin}}(x)$ if the set $\{\bar{d}_m[i] \mid \{q_n\}_{n\geq 0} \models \varphi_i^{\text{fin}}(m)\}$ is finite. Allowing only finitely many values in the run, together with the previous constraint, is clearly sufficient for describing the projections of the register traces of $\mathbf{A}$ on the first register.

Tuple inequality constraints. Consider again the register automaton $\mathbf{A}$ of Example 23, but now E is ternary and δ contains $E(x_1, x_2, y_1)$ while δ' contains is negation, $\neg E(x_1, x_2, y_1)$. We again wish to project away register 2. It is now allowed for a data value in register 1 to appear in both odd and even positions within the same run. However, if we consider an odd position α and an even position β within a run, the tuple formed with the data values in register 1 at position α and $\alpha + 1$ cannot be equal to the tuple formed with the data values in register 1 at a position β and $\beta + 1$. To deal with this, we introduce a mechanism to specify tuple inequality constraints as follows. A tuple inequality constraint for a register automaton $\mathbf{A}$ with states Q is an MSO formula $\varphi^{\neq}_{\bar{i},\bar{j}}(\bar{\alpha}, \bar{\beta})$ over Q^ω, where $\bar{\alpha}$ and $\bar{\beta}$ are tuples of distinct free variables of the same arity l and $\bar{i}, \bar{j}$ are tuples of registers of the same arity. The constraint is satisfied in a run $\{(\bar{d}_n, q_n, \delta_n)\}_{n\geq 0}$ if for every pair of tuples $(\bar{\alpha}, \bar{\beta})$ such that $\{q_n\}_{n\geq 0} \models \varphi^{\neq}_{\bar{i},\bar{j}}(\bar{\alpha}, \bar{\beta})$, $(d_{\alpha_1}[i_1], \ldots d_{\alpha_l}[i_l]) \neq (d_{\beta_1}[j_1], \ldots d_{\beta_l}[j_l])$.

In the extension of Example 23 with a ternary symbol E mentioned above, in order to describe the projections to register 1, we would add the binary constraints $\varphi^{\neq}_{11,11}(xx', yy')$ with a formula expressing the fact that position x is odd, position y is even, $x' = x + 1$ and $y' = y + 1$. Notice that the inequality constraints $e^{\neq}_{11}$, that can be used when E is binary in order to enforce that a data value at an even position can not be equal to a data value at a odd position, can equivalently be enforced with a tuple inequality constraint $\varphi^{\neq}_{1,1}(x, y)$ whose formula express the fact that x is even and y is odd. Similarly, every inequality constraint can be expressed as a tuple inequality constraint of arity 1. Tuple inequality constraints are therefore generalizations of inequality constraints.

We next define an automaton model that augments extended automata with finiteness and tuple inequality constraints. An *enhanced automaton* is a pair $(\mathbf{A}, \Sigma)$ where $\mathbf{A}$ is a register automaton and Σ is a set of equality, tuple inequality, and finiteness constraints for $\mathbf{A}$. We show the following.

Theorem 24. *Let $\mathbf{A}$ be an automaton with k registers and schema σ. Let $m \le k$. There is an enhanced automaton $\mathcal{B}$ with m registers and no database, such that $\mathcal{R}eg(\mathcal{B}) = \bigcup_D \Pi_m(\mathcal{R}eg(D, \mathbf{A}))$, where the databases D are over schema σ.*

Proof. Let $\mathbf{A} = (k, \sigma, Q, I, F, \Delta)$ be a register automaton. Without loss of generality we can assume that $\mathbf{A}$ is state-driven and complete. Let $m \le k$. We construct the enhanced automaton $\mathcal{B} = (\mathbf{B}, \Sigma)$ as follows. First, $\mathbf{B} = (m, Q, I, F, \Delta')$ where $\Delta' = \{(p, \delta|m, q) \mid (p, \delta, q) \in \Delta\}$ and $\delta|m$ denotes the restriction of δ to the first m registers. Note that $\mathbf{B}$ remains complete and state-driven. We now describe Σ. The equality constraints are constructed as done in the absence of a database (see the proof of Theorem 13). The finiteness constraints are defined as follows. Let $w = \{q_n\}_{n\ge 0}$ be a state trace of $\mathbf{B}$ (which is also a state trace of $\mathbf{A}$). As $\mathbf{A}$ is state-driven, q_n determines the type δ_n such that $\{(q_n, \delta_n)\}_{n\ge 0}$ is the control trace associated to w. For each $i \in [\![m]\!]$, $\varphi_i^{\text{fin}}(x)$ defines the set of positions h such that $[(h, i)]_w^{\mathbf{A}}$ is in $adom_w(\mathcal{E}_{\mathbf{A}}^w)$ (recall the definition of $adom_w(\mathcal{E}_{\mathbf{A}}^w)$ in the proof of Theorem 9). Clearly, this can be specified in MSO. Observe that in an actual run on database D, the values of registers in the active domain are generally a subset of $adom(D)$. Indeed, D may contain values not occurring in any register in the run.

We next explain the construction of the tuple inequality constraints. We use the following notation. For $r \in \{x_i, y_i\}, r' \in \{x_j, y_j\}$ and positions n and n', we denote $(n, r) \sim_w^{\mathbf{A}} (n', r')$ if $(u_n, i) \sim_w^{\mathbf{A}} (u_{n'}, j)$, where $u_n = n$ if $r = x_i$, $u_n = n + 1$ if $r = y_i$, and similarly for $u_{n'}$.

Consider a control trace $w = \{(q_n, \delta_n)\}_{n\ge 0}$ of $\mathbf{A}$. First, the inequality constraints $e_{ij}^{\neq}$ constructed as in Theorem 13 are expressed straightforwardly as monadic tuple inequality constraints $\varphi_{ij}^{\neq}(\alpha, \beta)$. Next, we define several tuple inequality constraints for each symbol R in σ. Let κ be the arity of R. Each tuple inequality constraint $\varphi_{\bar{i},\bar{j}}^{\neq}(\bar{\alpha}, \bar{\beta})$ for R states that there exist positions n and n' and a partition $\{E, F\}$ of $[\![\kappa]\!]$ such that:

- $|\bar{\alpha}| = |\bar{\beta}| = |F|$
- δ_n contains a negative literal $\neg R(s_1, \dots s_\kappa)$
- $\delta_{n'}$ contains a positive literal $R(r_1, \dots, r_\kappa)$
- $(n, s_l) \sim_w^{\mathbf{A}} (n', r_l)$ for $l \in E$,
- $(\alpha_l, i_l) \sim_w^{\mathbf{A}} (n, s_l)$ for $l \in F$,
- $(\beta_l, j_l) \sim_w^{\mathbf{A}} (n', r_l)$ for $l \in F$.

Clearly, the above can be stated in MSO. This means that the positions involved can be specified using regular expressions over the states in w.

Let Σ consist of the equality, finiteness, and tuple inequality constraints defined above and $\mathcal{B}$ be $(\mathbf{B}, \Sigma)$. We claim that $\mathcal{R}eg(\mathcal{B}) = \bigcup_D \Pi_m(\mathcal{R}eg(D, \mathbf{A}))$, where each D is over σ.

The inclusion $\mathcal{R}eg(\mathcal{B}) \supseteq \bigcup_D \Pi_m(\mathcal{R}eg(D, \mathbf{A}))$ is simple and omitted here. We next prove the converse.

We will need the following easily proven lemma. Intuitively, it says that in a run of a register automaton on a database, the values of registers that are not forced to be in the active domain of the database by the control trace can be changed to values outside the active domain of the database in any way consistent with the (in)equalities, without affecting validity of the run on that database.

Lemma 25. *Let $\mathbf{A} = (k, \sigma, Q, I, F, \Delta)$ be a register automaton and $\rho = \{(\bar{d}_n, q_n, \delta_n)\}_{n\ge 0}$ a run of $\mathbf{A}$ over some database D. Denote by w the control trace of ρ. Let g be a mapping from $\mathcal{E}_{\mathbf{A}}^w$ to $\mathbb{D}$ that associates to each $\epsilon \in adom_w(\mathcal{E}_{\mathbf{A}}^w)$ the value induced by ρ, and an arbitrary value in $\mathbb{D} - adom(D)$ to each $\epsilon \notin adom_\rho(\mathcal{E}_{\mathbf{A}}^w)$, such that $\epsilon_1 \not\approx_w^{\mathcal{A}} \epsilon_2$ implies that $g(\epsilon_1) \neq g(\epsilon_2)$. Consider the sequence $g(\rho)$ obtained from ρ by replacing $\bar{d}_n[i]$ with $g([(n, i)]_w)$ (observe that g does not modify registers whose value is in $adom(D)$). Then $g(\rho)$ is a run of $\mathbf{A}$ on D.*

Let $\rho' = \{(\bar{d}_n', q_n, \delta_n|m)\}_{n\ge 0}$ be a run of $\mathcal{B}$. We need to construct a database D over σ and a run ρ of $\mathbf{A}$ over D such that the projection of the register trace of ρ to the first m registers is $\{\bar{d}_n'\}_{n\ge 0}$. The control trace of ρ will be $w = \{(q_n, \delta_n)\}_{n\ge 0}$. It remains to construct D and the values of the registers $\bar{d}_n$ for each position n.

This is done as follows. From ρ' we construct a symbolic trace w^* of some register automaton $\mathbf{A}^*$ over the schema σ^* which is σ augmented with the set C of constants from the active domain (this is made more precise below). Essentially, w^* and $\mathbf{A}^*$ are the extension of w and $\mathbf{A}$ to the schema σ^* enforcing equality with the constants whenever necessary. By the proof of Theorem 9 there is a finite database D^* over the schema σ^* witnessing the fact that the symbolic trace is actually a control trace via some run ρ^* over D^*. We then need to slightly modify D^* and ρ^* so that the projection of the register trace of ρ^* to the first m registers is the register trace of ρ'. This yields the desired database and run of $\mathbf{A}$ over the database. We now provide the details of the construction.

We start by defining the set of constants that are part of σ^*. Let C be the set of data values occurring in ρ' that are forced by w to be the active domain of the database. In other words, C is the set of data values d such that $d = \bar{d}_h'[i]$ for some register $i \le m$ and some position h such that $[(h, i)]_w^{\mathbf{A}} \in adom(\mathcal{E}_w^{\mathbf{A}})$. The finiteness constraints of Γ ensure that C is finite.

We now construct a symbolic trace $w^* = \{(q_n, \delta_n^*)\}_{n\ge 0}$, where δ_n^* is defined from δ_n as follows.

For each δ_n, we denote by $C(\delta_n)$ the extension of δ_n with the constants C by including the following positive literals (for all $i \in [\![k]\!]$, $c \in C$, $R \in \sigma$, and $\bar{h} \subseteq \bar{x} \cup \bar{y} \cup C$):

- $x_i = c$ if $(n, i) \sim_w^{\mathbf{A}} (n', j)$ with $j \le m$ and $\bar{d}_{n'}'[j] = c$
- $y_i = c$ if $(n+1, i) \sim_w^{\mathbf{A}} (n', j)$ with $j \le m$ and $\bar{d}_{n'}'[j] = c$
- $R(h_1, \dots h_\kappa)$ if there is a partition $\{E, F\}$ of $[\![k]\!]$ such that $h_i \in \bar{x} \cup \bar{y}$ for $i \in E$, h_i is $c_i \in C$ for $i \in F$, and for some position n' there is $R(r_1, \dots, r_\kappa) \in \delta_{n'}$ such that for all $i \in E$, $(n, h_i) \sim_w^{\mathbf{A}} (n', r_i)$, and for all $i \in F$, $r_i = c_i$ has been derived in $\delta_{n'}$ by one of the previous two items.

Let δ_n^* consist of $C(\delta_n)$ together with the negative literals induced by the complement of the above positive literals. Thus, δ_n^* is complete. To show that it is consistent, it is enough to show consistency of the positive facts with respect to equality. Consistency without constants is inherited from δ_n. Therefore, the non-trivial cases are those involving constants. We illustrate with two representative cases. For equality, we need to show, for instance, that $x_i = c, x_j = c \in \delta^*_n$ implies that $x_i = x_j \in \delta^*_n$ Suppose that $x_i \neq x_j \in \delta^*_n$. By definition, we have positions n'_1, n'_2 such that $(n,i) \sim^{\mathbf{A}}_w (n'_1,u)$ with $u \leq m$ and $\bar{d}'_{n'_1}[u] = c$ and $(n,j) \sim^{\mathbf{A}}_w (n'_2,v)$ with $v \leq m$ and $\bar{d}'_{n'_2}[v] = c$. By definition we also have $(n,i) \not\sim^{\mathbf{A}}_w (n,j)$ and therefore $(n'_1,u) \not\sim^{\mathbf{A}}_w (n'_2,v)$. But as $\rho' \models \Sigma$, this implies (assuming that $n'_1 \leq n'_2$) that $q_{n'_1} \cdots q_{n'_2} \in e^{\neq}_{uv}$ contradicting the fact that $\bar{d}'_{n'_1}[u] = \bar{d}'_{n'_2}[v] = c$. The other cases are treated similarly.

Consider now consistency of relational atoms with respect to equality. Again, the interesting case involves the constants. As a typical example, suppose $R(x_1,c_2,c_3), x_2 = c_2, x_3 = c_3 \in \delta^*_n$. We need to show that $R(x_1,x_2,x_3) \in \delta^*_n$. Suppose towards a contradiction that $\neg R(x_1,x_2,x_3) \in \delta^*_n$. By definition of the equalities $x_2 = c_2, x_3 = c_3$, there exist positions α_2, α_3 such that $(n,2) \sim^{\mathbf{A}}_w (\alpha_2, i_2)$ with $i_2 \leq m$ and $\bar{d}_{\alpha_2}[i_2] = c_2$ and $(n,3) \sim^{\mathbf{A}}_w (\alpha_3, i_3)$ with $i_3 \leq m$ and $\bar{d}_{\alpha_3}[i_3] = c_3$. By definition of $R(x_1,c_2,c_3)$, we have a position n', and registers r_1,r_2,r_3 such that $(n,x_1) \sim^{\mathbf{A}}_w (n',r_1)$ and $r_2 = c_2, r_3 = c_3 \in \delta^*_{n'}$. By definition of the equalities $r_2 = c_2, r_3 = c_3$, there are positions β_2,β_3 such that $(n',r_2) \sim^{\mathbf{A}}_w (\beta_2,j_2)$, $(n',r_3) \sim^{\mathbf{A}}_w (\beta_3,j_3)$, for $j_2,j_3 \leq m$, and $\bar{d}_{\beta_2}[j_2] = c_2$, $\bar{d}_{\beta_3}[j_3] = c_3$. However, the inequality constraint for $\neg R(x_1,x_2,x_3)$ at position n, $R(r_1,r_2,r_3)$ at position n', $E = \{1\}$, $F = \{2,3\}$, $\bar{\alpha} = \alpha_2\alpha_3$, $\bar{\beta} = \beta_2\beta_3$, $\bar{i} = i_2 i_3$ and $\bar{j} = j_2 j_3$, says that $(\bar{d}_{\alpha_2}[i_2], \bar{d}_{\alpha_3}[i_3]) \neq (\bar{d}_{\beta_2}[j_2], \bar{d}_{\beta_3}[j_3])$, a contradiction.

Consider the sequence $w^* = \{(q_n, \delta^*_n)\}_{n \geq 0}$. We show that w^* satisfies the conditions of a symbolic control trace with constants C. As we have constructed the δ^*_n so that they are complete and consistent, it remains to show that δ^*_{n+1} is consistent with δ^*_n in the sense that $\delta^*_n|(\bar{y} \cup C)$ is isomorphic to $\delta^*_{n+1}|(\bar{x} \cup C)$. The case of literals without constants is inherited from w. Suppose that $y_i = c \in \delta^*_n$. From the definition of equality with constants, and the fact that $(n,y_i) \sim^{\mathbf{A}}_w (n+1,x_i)$, it follows that $x_i = c \in \delta^*_{n+1}$. Similarly, suppose $R(\bar{h},\bar{c})$ is part of δ^*_n for $\bar{h} \subseteq \bar{y}$. From the fact that $(n,y_i) \sim^{\mathbf{A}}_w (n+1,x_i)$ for every $i \in [\![k]\!]$ it easily follows that $R(\bar{h}(\bar{y} \leftarrow \bar{x}),\bar{c})$ is part of δ^*_{n+1}, where $\bar{h}(\bar{y} \leftarrow \bar{x})$ is obtained from $\bar{h}$ by replacing each y_i with x_i, as desired.

One can easily construct a register automaton $\mathbf{A}^*$ over σ^* for which w^* is a symbolic control trace (the automaton simply allows all transitions occurring in the sequence). By the proof of Theorem 9, this symbolic run is actually the control trace of a real run. Hence there is a finite database D^* over σ^* and a run $\rho^* = \{(\bar{d}_n, q_n, \delta^*_n)\}_{n \geq 0}$ of $\mathbf{A}^*$ over D^*. Observe that by construction of the δ^*_n, $\bar{d}_n$ agrees with $\bar{d}'_n$ on all registers with values in C, but may disagree on others. To fix this, we modify D^* and ρ^* as follows. We first ensure that $adom(D^*)$ contains no value that occurs in ρ' but is not in C by applying to D^* and ρ^* an appropriate isomorphism (note that this uses the assumption that each run leaves out infinitely many values of $\mathbb{D}$). that is the identity on C. Let $\bar{D}^*$ and $\bar{\rho}^* = \{(\bar{e}_n, q_n, \delta_n)\}_{n \geq 0}$ be the resulting database and run. Finally, we can use Lemma 25 to modify $\{\bar{e}_n\}_{n \geq 0}$ so that it agrees with $\{\bar{d}'_n\}_{n \geq 0}$ on the first m registers. More precisely, let g be the mapping on $\mathcal{E}^w_{\mathbf{A}}$ such that $g([n,i]^{\mathbf{A}}_w) = d'_{n'}[j]$ if $(n,i) \sim^{\mathbf{A}}_w (n',j)$ for some n' and $j \leq m$, and g is the identity everywhere else. Note that g is the identity on $adom_w(\bar{\rho}^*)$ (including C). Indeed, if $(n,i) \sim^{\mathbf{A}}_w (n',j)$ then either $\bar{e}_n[i] = \bar{d}'_{n'}[j] \in C$ or $[(n,i)]^{\mathbf{A}}_w \notin adom_w(\bar{\rho}^*)$. The fact that g does not modify register values in C, together with the (in)equality constraints of Σ, ensure that g is consistent with (in)equalities. Also, $g(\epsilon) \in \mathbb{D} - adom(\bar{D}^*)$ whenever $g(\epsilon) \neq \epsilon$. By Lemma 25, $g(\bar{\rho}^*)$ is a run of $\mathbf{A}$ on $\bar{D}^*$ and by construction, the projection of its register trace on the first m registers is the register trace of ρ'. □

7 CONCLUSIONS

The main contribution of this paper is to gain insight into the means needed to specify views of database-driven systems. We use as a vehicle register automata, essentially identical to artifact systems. We have seen that describing even very simple projection views of register automata without a database requires the addition of global constraints. This gave rise to extended register automata. We showed that these have desirable properties, including decidability of verification of LTL-FO properties. We also characterized the precise fragment of extended automata needed to specify projections of register automata. In the absence of a database, extended register automata are themselves closed under projection, but this fails when a database is present. We therefore considered the additional features needed to specify projections of register automata with a database. While a solution to the general case remains elusive, we addressed projection views in which the entire database is hidden in addition to some of the registers. We showed that these views can be specified by extended automata further augmented with finiteness constraints and tuple inequality constraints. This gave rise to enhanced automata.

Recent results, which we will present in the full paper, show that the enhanced register automata model enjoys most of the nice properties of the extended register automata model. In particular, its state and control traces remain quasi-regular, generalizing Theorem 9. As a consequence, enhanced automata have all the nice decidablity properties mentioned in Theorem 12. On the other hand, Theorem 24 does not fully extend to enhanced automata. Specifically, enhanced automata are closed under the projection that removes the database entirely (yielding the variant of Theorem 24 where $k = m$, but with an enhanced register automaton as input instead of a simple register automaton). However, they are not closed under projections that eliminate one or more registers.

The results obtained highlight the technical challenges in specifying views of database driven systems. Remaining open questions closely related to this paper include the following: (i) precise complexity of our decision procedures and automata constructions, and (ii) characterization of more general projection views that include arbitrary projections of database relations. More broadly, richer classes of views need to be investigated for richer models of database driven systems. The difficulties encountered even in the simple framework studied here suggest that obtaining precise specifications of such views may turn out to be a challenging goal. As a second best, it would be useful to provide approximate descriptions that satisfy certain tightness conditions.

ACKNOWLEDGMENTS

Victor Vianu was supported in part by the National Science Foundation under award III-1815247.

REFERENCES

[1] Serge Abiteboul, Pierre Bourhis, and Victor Vianu. Comparing workflow specification languages: A matter of views. *ACM Transactions on Database Systems*, 37(2), 2012.

[2] Serge Abiteboul, Pierre Bourhis, and Victor Vianu. Explanations and transparency in collaborative workflows. In *PODS*, 2018.

[3] Serge Abiteboul and Victor Vianu. Collaborative data-driven workflows: Think global, act local. In *PODS*, pages 91–102, 2013.

[4] Hajnal Andréka, Johan van Benthem, and István Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274, 1998.

[5] Mikolaj Bojańczyk. A bounding quantifier. In *Computer Science Logic, CSL*, pages 41–55, 2004.

[6] Mikolaj Bojańczyk, Luc Segoufin, and Szymon Toruńczyk. Verification of database-driven systems via amalgamation. In *PODS*, pages 63–74, 2013.

[7] J.R. Büchi. On a decision method in restricted second-order arithmetic. In *Int. Congr. for Logic, Methodology and Philosophy of Science*, pages 1–11. Standford Univ. Press.

[8] Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. Foundations of data-aware process analysis: a database theory perspective. In *PODS*, 2013.

[9] Issam Chebbi, Schahram Dustdar, and Samir Tata. The view-based approach to dynamic inter-organizational workflow cooperation. *Data Knowl. Eng.*, 56(2):139–173, February 2006.

[10] Thomas Colcombet. Regular cost functions, part I: logic and algebra over words. *Logical Methods in Computer Science*, 9(3), 2013.

[11] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

[12] Elio Damaggio, Alin Deutsch, and Victor Vianu. Artifact systems with data dependencies and arithmetic. *ACM Trans. Database Syst.*, 37(3):22, 2012.

[13] Alin Deutsch, Richard Hull, Yuliang Li, and Victor Vianu. Automatic verification of database-centric systems. *SIGLOG News*, 5(2):37–56, 2018.

[14] Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. Automatic verification of data-centric business processes. In *ICDT*, pages 252–267, 2009.

[15] Zdenek Dvorak and Daniel Král. Classes of graphs with small rank decompositions are x-bounded. *Eur. J. Comb.*, 33(4):679–683, 2012.

[16] Rik Eshuis and Paul Grefen. Constructing customized process views. *Data and Knowledge Engineering*, 64(2):419 – 438, 2008.

[17] Erich Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64(4):1719–1742, 1999.

[18] R. Hull, N.C. Narendra, and A. Nigam. Facilitating workflow interoperation using artifact-centric hubs. In *Intl. Conf. Service Oriented Computing (ICSOC)*, 2009.

[19] Adrien Koutsos and Victor Vianu. Process-centric views of data-driven business artifacts. *J. Comput. Syst. Sci.*, 86:82–107, 2017.

[20] Santhosh Kumaran, Prabir Nandi, Terry Heath, Kumar Bhaskaran, and Raja Das. ADoc-oriented programming. In *Symposium on Applications and the Internet*, pages 334–341. IEEE, 2003.

[21] Duen-Ren Liu and Minxin Shen. Workflow modeling for virtual processes: An order-preserving process-view approach. *Inf. Syst.*, 28(6):505–532, 2003.

[22] Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

[23] Anil Nigam and Nathan S Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.

[24] Amir Pnueli. The temporal logic of programs. In *FOCS*, 1977.

[25] Luc Segoufin and Szymon Toruńczyk. Automata based verification over linearly ordered data domains. In *STACS*, volume 9, pages 81–92, 2011.

[26] Sira Yongchareon and Chengfei Liu. A process view framework for artifact-centric business processes. In *OTM Conferences (1)*, pages 26–43, 2010.

[27] Xiaohui Zhao, Chengfei Liu, Wasim Sadiq, and Marek Kowalkiewicz. Process view derivation and composition in a dynamic collaboration environment. In *OTM Conferences (1)*, pages 82–99, 2008.