INFORMATION AND COMPUTATION **114**, 192–213 (1994)

Finite-State Unification Automata and Relational Languages

YAEL SHEMESH AND NISSIM FRANCEZ*

Computer Science Department, Technion—Israel Institute of Technology, Haifa 32000, Israel

We define the new notion of a (finite-state) unification automaton, a device for finite-state recognition of relational languages by means of unification transitions. Words in such a language are formed by composing base relations, and have the general form $r_{i_1}(x_{j_1}, x_{k_1}) \cdots r_{i_n}(x_{j_n}, x_{k_n})$ for some n. Generation of such languages by regarding Horn clauses as grammars has been considered before, but to the best of our knowledge, recognizing such languages by suitably designed automata is a new approach. The main result presented is a pumping lemma, forming a necessary condition for finite-state recognizability. Some example results about such automata are given. © 1994 Academic Press, Inc.

1. INTRODUCTION

In this paper, we introduce a new kind of (finite state) automata, intended for an abstract study of *relational* (formal) *languages*. Ultimately, our aim is to study properties of *relations* by studying their representations as such languages. At this stage, some first steps toward the study of the languages themselves are attempted. In this paper, out attention is restricted to *finite-state recognizability* of such languages. Thus, the study here is conducted on a purely syntactic level, where no semantic interpretation is assigned to the relational languages considered.

Clearly, in order to reach the ultimate goal of studying *relations* and their computation via query languages, such a semantics must be provided. In doing so, one can connect the results obtained here about formal relational languages to actual query processing. However, doing so is beyond the scope of the current paper (as well as beyond a reasonable size). In the conclusions, some technical difficulties encountered by some work in progress regarding the semantic inerpretation of relational languages are pointed out.

It was conceived some time ago that Datalog programs (e.g., as presented in [UG86] or [MW88]) can be viewed as an abstract (CF)

* E-mail addresses: francez@techsel (BITNET), francez@cs.technion.ac.il (INTERNET).

0890-5401/94 $6.00

grammar-like structure, *generating* (the analogues of) languages over an alphabet derived from the base relations. For example, see [Sh87, La88] can be viewed as an abstract (CF) grammar-like structure, *generating* (the analogues of) languages over an alphabet derived from the base relations. For example, see [Sh87, La88] for such a view. Our aim here is to study the approach dual to *generability*, namely that of *recognizability*. The automata we consider have *unification* as part of their basic transitions, compatible with the same in the grammar counterpart. To the best of our knowledge *recognizing* relational languages by suitably designed automata is a new approach.

Due to the nature of relational languages, the alphabet is infinite. Thus, in addition to a finite number of states, our automata are equipped also with a finite number of *registers*, capable of holding as value a variable name (out of an *infinite* set of variable names). However, the operations on registers are restricted in a very strong way, so that the automaton is justly considered finite-state. We believe this approach to be the natural extension of finite-state machines to words over an infinite alphabet. In particular, the closure properties and decidability results characteristic of finite automata are basically preserved. Technically, the extension is obtained by replacing the *equality*-based transitions of classical automata by *unification*-based transitions. The latter employ both equality tests and copying. The use of unification in such a context is innovative on its own.

In the paper, we present the relevant defintions and several results, serving as examples to the power of the approach. We do not present here a systematic and full development of the theory. We believe that this new approach will turn beneficial and contribute to the understanding of query languages in the same way as the classic theories of automata and formal languages [HU79] contributed to the understanding of imperative programming.

Another treatment of a related family of automata, not confined to an alphabet related to relational languages, can be found in [KF90].

2. Notations and Basic Definitions

We mostly follow the notation in [Lo87].

A basic alphabet Σ is a finite, nonempty set of *relation names*, the elements of which are denoted by r, s, etc., possibly indexed.

$X = \{x_i \mid i \geqslant 1\}$ is the (infinite) set of *variables*. In examples, we also use x, y, z, etc.

An *explicit relational symbol* is of the form $r(x_i, x_j)$, where $r \in \Sigma$, $x_i, x_j \in X$.

The (infinite) alphabet is $\Sigma_X = \{r(x_i, x_j) \mid r \in \Sigma,\ x_i, x_j \in X\}$, the collection of all explicit relational symbols.

An *explicit relational chain* c is a finite word over Σ_X. As usual, Σ_X^* denotes the set of all explicit relational symbols (over Σ_X).

For example, let $\Sigma = \{r, p\}$; then $r(x_2, x_5)\, r(x_1, x_8)\, p(x_2, x_1)$ is an explicit relational chain.

An *explicit relational language* is a set $C \subseteq \Sigma_X^*$ of explicit relational chains.

A *variable-pure substitution* θ is a finite set of the form $\{v_1/u_1, ..., v_n/u_n\}$, where $v_i, u_i \in X$, $u_i \neq v_i$, $1 \leqslant i \leqslant n$, and the variables $v_1, ..., v_n$ are pairwise distinct. We refer to v_i, $1 \leqslant i \leqslant n$, as the *left-variables* (of θ), and to u_i, $1 \leqslant i \leqslant n$, as the respective *right-variables*. In the rest of this paper, whenever a substitution is mentioned, it means a variable-pure substitution. A substitution is 1–1 if $i \neq j \Rightarrow u_i \neq u_j$.

Let θ be a substitution and c be an explicit relational chain. Then $c\theta$, the *instance* of c by θ, is the explicit relational chain obtained from c by simultaneously replacing each occurrence of the variable v_i in c by the variable u_i, $1 \leqslant i \leqslant n$. Clearly, $(c_1 c_2)\theta = (c_1\theta)(c_2\theta)$.

Let c_1 and c_2 be explicit relational chains. We then say c_1 is *not* an instance of c_2 iff there exists *no* substitution θ such that $c_2\theta = c_1$. We say that c_1 and c_2 are *variants* (of each other) iff there exist variable-pure substitutions θ and σ such that $c_1 = c_2\theta$ and $c_2 = c_1\sigma$. Note that both θ and σ are 1–1.

EXAMPLE. Let $c_1 = r(x_1, x_5)$ and $c_2 = r(x_2, x_2)$. Then c_2 is an instance of c_1 obtained by the variable-pure substitution $\theta = \{x_1/x_2, x_5/x_2\}$. Yet c_1 is not an instance of c_2, since *all* instances of c_2 are of the form $c(v, v)$ for some $v \in X$. Such explicit relational symbols are called *diagonal*. Thus, c_1 and c_2 are not variants.

In our discussion, we *identify* all the explicit relational symbols and chains with all their respective variants. We use the terms *relational symbol* and *relational chain* (omitting the explicitness) to refer to these objects.

We naturally extend the notion of variants to explicit relational languages C_1 and C_2. Here again, we intend to identify explicit languages which are variants of each other. We use the term *relational language*, again denoted by C, to refer to these languages.

EXAMPLE. For the relational languages

$$C_1 = \{r_1(x_1, x_2)\, r_2(x_2, x_3), ..., r_1(x_{2n-1}, x_{2n})\, r_2(x_{2n}, x_{2n+1}) \mid n \geqslant 1\}$$

and

$$C_2 = \{r_1(x_{2n+1}, x_{2n})\, r_2(x_{2n}, x_{2n-1}), \ldots, r_1(x_3, x_2)\, r_2(x_2, x_1) \mid n \geqslant 1\},$$

we get $C_1 = C_2$; i.e., they both denote the *same* relational language.

EXAMPLE. The relational chain $r(x_1, x_1)$ is *not* a member of the relational language $\{r(x_1, x_2)\}$ since $r(x_1, x_1)$ is not a variant of $r(x_1, x_2)$.

From this point onward, we do not refer at all to explicit entities, only to elements unique up to a variant.

The generative view of Horn clauses as grammars (Horn-grammars) is obtained by applying the clauses top-down, in a "free" way, i.e., without any reference to the tables constituting the usual logical semantics of the base relations. Thereby, we obtain a "free" outcome, an uninterpreted relational language, instead of the table of the defined relations as usual.

EXAMPLE. Consider the language

$$C = \{r(x_1, x_2) \cdots r(x_{m-1}, x_m)\, r(x_m, x_{m-1}) \cdots r(x_2, x_1) \mid m \geqslant 2\}.$$

It is easy to see that this language is generated by the Horn-grammar G_C below:

$$G_C{::}\ R(x, y) \to r(x, z)\, R(z, y)\, r(z, x) \mid r(x, y)\, r(y, x).$$

In this grammar, $R(x, y)$ is the (only) nonterminal, and $r(x, y)$ is the only terminal. An example of a generation of a chain $c \in L(G_C)$ is

$$R(x_1, x_3) \to r(x_1, x_2)\, R(x_2, x_3)\, r(x_2, x_1)$$
$$\to r(x_1, x_2)\, r(x_2, x_3)\, r(x_3, x_2)\, r(x_2, x_1).$$

However, note that, given a relational chain in general, it is not explicit which are its free variables (carriers of the corresponding relation) for the sake of its logical interpretation. Thus, one could view the language C above as generable also by the following Horn-grammar G'_C, in which the initial variable is *unary*:

$$G'_C{::}\ R(x) \to r(x, y)\, R(y)\, r(y, x) \mid r(x, y)\, r(y, x).$$

The generation of the above relational chain now has the form

$$R(x_1) \to r(x_1, x_2)\, R(x_2)\, r(x_2, x_1)$$
$$\to r(x_1, x_2)\, r(x_2, x_3)\, r(x_3, x_2)\, r(x_2, x_1).$$

Or even a 0-ary initial variable could be used, by having G''_C contain, in addition to the rules in G'_C, also the rule $R \rightarrow R(x)$ and declaring R to be the initial variable.

We return to this issue in the Conclusion. We now define our new automata.

A finite-state datalog automaton (FSDA) is a 6-tuple $A = (\Sigma, Q, q_0, F, W, T)$, where:

Σ, Q, q_0, and F are the usual (finite) alphabet, states, initial state, and accepting states, respectively.

W is a vector of *registers.* All the registers may have a special value *nil*, indicating, during unification, being noninstantiated. This is also their initial value. The register number i is denoted $w_i \in X \cup \{nil\}$. We use $k = |W|$ for the number of registers, and $\overline{nil}$ to denote the register vector s.t. $w_i = nil$, $1 \leqslant i \leqslant k$. As usual, context determines when w_i denotes the register itself and when it denotes the contents of the register. We use N to denote the natural numbers.

T is the *transition relation* of the type $T: Q \times W \times \Sigma \times N^2 \times N^{*-k} \rightarrow Q \times W$, where $N^{*-k} = \bigcup_{0 \leqslant j \leqslant k} N^j$. We denote transitions by

$$(q, W) \xrightarrow[w_{m_1}, ..., w_{m_n} \leftarrow nil]{r(w_i, w_j)} (q', W').$$

Such a transition is possible iff the input relational symbol $r^*(x_h, x_l)$ satisfies

$$U(\langle i, j \rangle, W, \langle m_1, ..., m_n \rangle, r^*(x_h, x_l), W'),$$

holding if and only if all the following conditions hold:

$$r^* = r \tag{1}$$

$$(w_i = nil \vee w_i = x_h) \wedge ((w'_i = x_h \wedge i \notin \{m_1, ..., m_n\}) \vee (w'_i = nil \wedge i \in \{m_1, ..., m_n\})) \tag{2}$$

$$(w_j = nil \vee w_j = x_l) \wedge ((w'_j = x_l \wedge j \notin \{m_1, ..., m_n\}) \vee (w'_j = nil \wedge j \in \{m_1, ..., m_n\})) \tag{3}$$

$$(i \neq j) \vee (h = l) \tag{4}$$

$$\forall g (1 \leqslant g \leqslant k) \wedge (g \notin \{m_1, ..., m_n\}) \wedge g \notin \{i, j\} \Rightarrow w_g = w'_g \tag{5}$$

$$\forall g (1 \leqslant g \leqslant k) \wedge (g \in \{m_1, ..., m_n\}) \Rightarrow w'_g = nil. \tag{6}$$

The operational interpretation of a transition is as follows. In state q and registers W, when the input is $r^*(x_h, x_l)$, the automaton can select any transition such that the input *unifies* (in the usual logic-programming sense) with $r(w_i, w_j)$, to produce the new contents of the register vector W'. If the registers already have a value different from *nil*, the unification reduces to an equality test. If the value of a register is *nil*, the unification reduces to copying the input variable. After the unification all the registers indicated below the arc are reset to *nil* (we use the term *nilification* to denote this action). Finally, the automaton changes its state to q'.

A *configuration* γ of an FSDA automaton A is a pair $\langle q, W \rangle$, where $q \in Q$ is the *current state* and W is the *current contents* of the registers vector. A configuration captures an instantaneous description (or snapshot) during a computation of A in the usual way.

A *computation* of an FSDA automaton (on an input relational chain c) is a maximal (finite) sequence γ_i, $0 \leqslant i \leqslant l$, for some $l \geqslant 0$ (the length of the computation), of configurations, where $\gamma_0 = \langle q_0, \overline{nil} \rangle$ and γ_{i+1} is related to γ_i in the transition relation.

A *partial computation* may start from an arbitrary configuration and read only a part of the input relational chain.

A relational chain c is *accepted* by an FSDA A iff there exists a computation of A on c with a final configuration $\gamma = \langle q, W \rangle$, with $q \in F$. Note that the final values of the registers are immaterial.

We let $C(A)$ be the relational language accepted by A, in the usual way. A relational language is *relational regular* (rr) iff it is accepted by some FSDA A. We denote by **RR** the collection of all rr relational languages.

A configuration γ is *reachable* if there exists a relational chain $c \in C(A)$ such that the partial accepting computation of A on c reaches γ. The *support* $\mathrm{s}(\gamma)$ of a configuration γ is defined by $\mathrm{s}(\gamma) = \{i \mid 1 \leqslant i \leqslant k,\ w_i \neq nil\}$, the collection of all the non-*nil* registers. Its cardinality is denoted by $|\mathrm{s}(\gamma)|$.

For an FSDA automaton A we let $\mu_A = \max\{|\mathrm{s}(\gamma)| \mid \gamma \text{ reachable}\}$, the maximal support of reachable configurations. A configuration γ is *saturated* iff $|\mathrm{s}(\gamma)| = \mu_A$.

A consequence of the FSDA definition is that if a relational chain c is accepted by an FSDA A then A also accepts every instance of c.

We define an *expanded language* as a relational language which is closed under instances. Clearly, an rr relational language is an expanded one.

In the examples we use a self-explanatory graphical representation of automata.

EXAMPLE. The following FSDA (Fig. 1) defines the expanded language

$$C = \{r_1(x_1, x_2)\, r_2(x_2, x_3)\, r_1(x_3, x_4) \cdots r_2(x_{2n}, x_{2n+1}) \mid n \geqslant 1\}$$

generated by the Horn-grammar

$$R(x, y) \to r_1(x, z)\, r_2(z, w)\, R(w, y) \mid r_1(x, z)\, r_2(z, y).$$

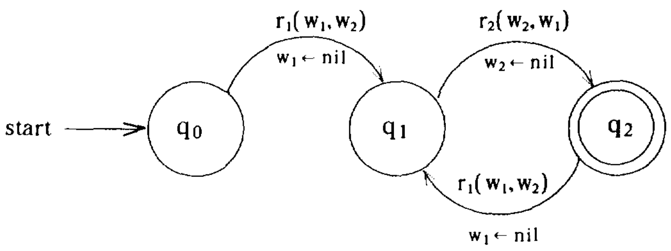

FIG. 1. An example automaton.

Proof. $\Leftarrow$ We prove by induction on the length l that every chain of the above form is accepted by A with a final configuration $\langle q_2, [x_a, nil] \rangle$, where x_a is the rightmost variable in the chain. Note that all the chains of the above form are of even length.

Basis: $l = 2$. In this case, the chain $c = r_1(x_1, x_2)\, r_2(x_2, x_3)$. Initially, the configuration is $\langle q_0, [nil, nil] \rangle$, and the transition to q_1 can take place as all six conditions are easily seen to hold. This transition ends in the configuration $\langle q_1, [nil, x_2] \rangle$, from which the transition to q_2 can take place, ending in the configuration $\langle q_2, [x_3, nil] \rangle$, which accepts the chain and has the required form.

Assumption: Assume the claim for $|c| \leqslant l = 2n$. Let c be of the above form with $|c| = l + 2 = 2(n+1)$, i.e., $c = r_1(x_1, x_2)$ $r_2(x_2, x_3)$ $r_1(x_3, x_4) \cdots r_2(x_{2n}, x_{2n+1})\, r_1(x_{2n+1}, x_{2n+2})\, r_2(x_{2n+2}, x_{2n+3})$. Consider the prefix c' of c, $c' = r_1(x_1, x_2)\, r_2(x_2, x_3)\, r_1(x_3, x_4) \cdots r_2(x_{2n}, x_{2n+1})$. Clearly, $|c'| = l$, and also $c' \in C$. By the induction assumption, the computation of A on c' ends in the configuration $\langle q_2, [x_{2n+1}, nil] \rangle$, and a similar reasoning as in the basis step establishes the claim. This ends the proof for this direction.

$\Rightarrow$ Let $\pi = \gamma_i$, $0 \leqslant i \leqslant l$, with $\gamma_0 = \langle q_0, nil \rangle$, $\gamma_l = \langle q, W \rangle$, be a computation of A on some c. Note that $q \neq q_0$. By induction on the length $|\pi| = l$, we show that π satisfies the following two properties:

1. If $\gamma_l = \langle q_2, W \rangle$, then $W = [x_{2a+1}, nil]$, and $c = r_1(x_1, x_2)$ $r_2(x_2, x_3) \cdots r_2(x_{2a}, x_{2a+1})$, with $a \geqslant 1$.

2. If $\gamma_l = \langle q_1, W \rangle$, then $W = [nil, x_{2a}]$, and $c = r_1(x_1, x_2)$ $r_2(x_2, x_3) \cdots r_1(x_{2a-1}, x_{2a})$, with $a \geqslant 1$.

Basis: $l = 1$. Such a computation π starts in q_0 and ends in q_1. It accepts a chain of the form $c = r(x_1, x_2)$ which has the claimed form, and since initially $w_1 = w_2 = nil$, then after unification and "nilification" of w_1, the final configuration is as required.

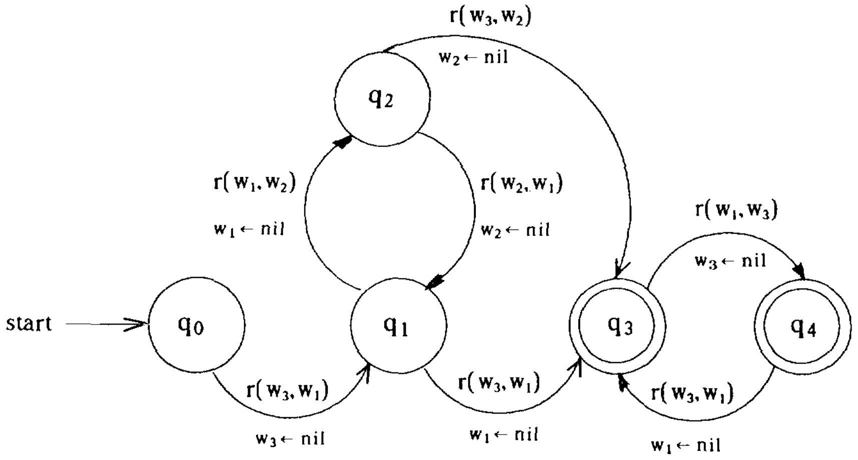

FIG. 2. Another example.

Basis: $l=2$. Similar.

Assumption: Assume the claim for $|\pi| \leqslant 2n$. Consider a computation π with $|\pi| = 2n+1$. By the inductive assumption, after the first $2n$ steps, the input had the form $c = r_1(x_1, x_2)\ r_2(x_2, x_3) \cdots r_1(x_{2n-1}, x_{2n})$, and the resulting configuration is $\langle q_1, [nil, x_{2n}]\rangle$. Thus, in the last step, the transition from q_1 to q_2 takes place. Since unification succeeds, w_1, which

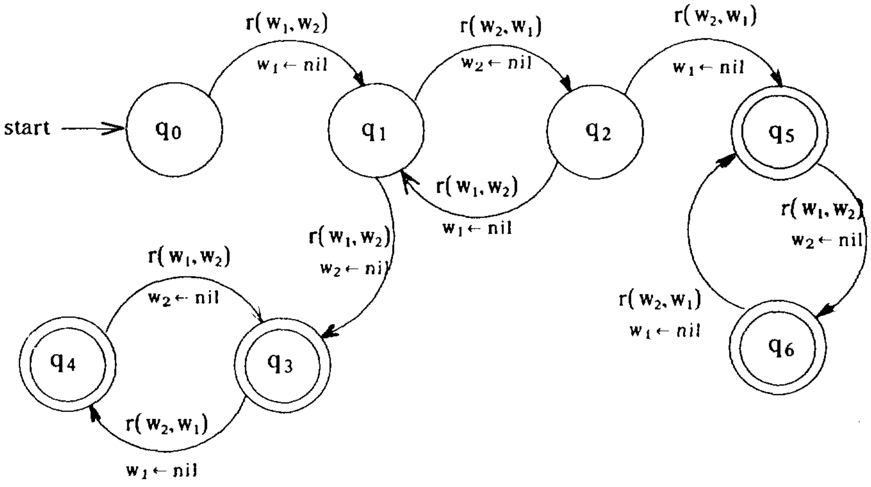

FIG. 3. An equivalent automaton.

had the value *nil*, copies the last variable x_{2n+1}, while w_2 verifies that the first variable in the input symbol is indeed x_{2n} and is "nilified" thereafter. Thus, property 1 is reestablished. Next, for a computation π with $|\pi| = 2n+2$, the reasoning is similar and omitted. This concludes the proof for the second direction. ▮

EXAMPLE. The following FSDA (Fig. 2) defines the expanded language:

$$\{r(x_1, x_2)\, r(x_2, x_3) \cdots r(x_{n-1}, x_n)\, r(x_{n+1}, x_n)$$
$$r(x_{n+2}, x_{n+1}) \cdots r(x_{n+m}, x_{n+m-1}) \mid n \geqslant 2, m \geqslant 1\}.$$

The proof is similar to the previous one but with somewhat more complicated inductive assumptions and is omitted. For this relational language there is a tradeoff between the number of states and the number of registers. The above FSDA includes three registers and five states. An equivalent FSDA described below (Fig. 3) has only two registers but has seven states. The proof that this automaton accepts the same language is tedious and is omitted.

3. RESULTS

In this section, several example theorems regarding FSDAs are stated and proved.

The first theorem shows that the FSDA power is limited; i.e., not all the expanded relational languages are *rr*.

THEOREM. $C =^{\mathrm{df}} \{r(x_1, x_{2n})\ r(x_2, x_{2n-1}) \cdots r(x_n, x_{n+1})\ r(x_{n+1}, x_n) \cdots r(x_{2n}, x_1) \mid n \geqslant 1\}$ is not *rr*.

Proof. The basic intuition behind the proof is as follows. Each $c \in C$ is naturally divided into two parts c_1 and c_2, with $|c_1| = |c_2|$. In terms of the above specific representation of C, c_1 consists of those $r(x_i, x_j)$ in c with $i < j$ and c_2 consists of those $r(x_i, x_j)$ in c with $i > j$. For a more general representation (using arbitrary variants), the description would use the *reverse* of the sequence of first arguments as the sequence of second arguments. Each variable which appears in c_1 appears also in c_2. Therefore, for $|c| = 2n$, after reading c_1, $2n$ variables have to be "remembered" and tested in c_2. The proof shows that this cannot be done because the size of the FSDA register vector is bounded. It is not hard to see that the language C can be generated by a Horn grammar. We now present the details of the proof.

By way of contradiction, suppose that there exists an FSDA A s.t. $C = C(A)$. From the definition it follows that there exists a $c^* \in C$ s.t. the computation of A on c^* reaches a saturated configuration. Suppose that

$$c^* = r(x_1, x_{2n^*})\, r(x_2, x_{2n^*-1}) \cdots r(x_{2n^*}, x_1).$$

Clearly, $c^* = c_1^* c_2^*$ with $|c_1^*| = |c_2^*|$, where for c_1^* any element $r(x_i, x_j)$ satisfies $i < j$, and for c_2^* every element satisfies $i > j$ (and in none is $i = j$ possible!). Consider $\hat{c}$, the *shortest* prefix of c^* leading A to a saturated configuration $\hat{\gamma} = \langle \hat{q}, \hat{W} \rangle$.

Claim. $\hat{c}$ is prefix of c_1^*.

Suppose that the first time A reaches a saturated configuration $\hat{\gamma}$ occurs when reading $r(x_i, x_j)$ with $i > j$ (i.e., in c_2^*, the second half of c^*). Thus, at least one register, say w_m, participating in this transition has a *nil* value before the transition, and w.l.o.g. has x_i as its value after the transition (otherwise, the previous configuration was already saturated, contrary to the assumption of $\hat{c}$ being shortest).

From the definition of C it follows that the relational symbol $r(x_j, x_i)$ was already read beforehand. Thus, consider the partial computation of A on $c^i = r(x_1, x_{2n^*}) \cdots r(x_j, x_i) \cdots r(x_l, x_i)$, obtained from the first $2n^* - i$ components of c^*, to which $r(x_l, x_i)$ is appended at the end (with $l \neq j$). During this computation, A reaches a saturated configuration $\gamma' = \langle q', W' \rangle$, s.t. W' differs from $\hat{W}$ only in w_m, where $\hat{w}_m = x_j$, while $w_{m'} = x_l$. Since $C(A) = C$, it follows that after reading $r(x_j, x_i)$ during the computation on c^*, and hence also during the computation on c^i, no further use is made of $\hat{w}_m$ until its "nilification," and similarly for w'_m. Thus, the relational chain $c^+ = c^i \cdots r(x_{2n^*}, x_1)$ is also accepted by A, through $c^+ \notin C$. Thus, the claim is established.

Returning to the main proof, we now define the relational chain $\bar{c} = r(x_1, x_{2\bar{n}}) \cdots r(x_{\bar{n}}, x_{\bar{n}+1}) \cdots r(x_{2\bar{n}}, x_1)$, for some $\bar{n} > n^*$. Suppose that $r(x_s, x_t)$ is the last symbol in the prefix $\bar{\bar{c}}$ of $\bar{c}$ leading for the first time to a saturated configuration, which is a variant of $\hat{c}$. The computation of A on $\bar{c}$ will first reach a saturated configuration $\langle q, \bar{\bar{W}} \rangle$, say after the prefix $\bar{\bar{c}}$ of c.

The next relational symbol after $\bar{\bar{c}}$ is $r(x_{s+1}, x_{t-1})$, for some t s.t. (by the claim above) $s + 1 < t - 1$, causing a transition via an edge $(q, W, r(w_a, w_b))$ to $(q', \bar{\bar{W}}')$. Since x_{s+1} and x_{t-1} are "new" variables, which have not appeared in $\bar{\bar{c}}$, we get that $\bar{\bar{w}} = \bar{\bar{w}}_b = nil$. Hence, after this relational symbol is read, at least one of the following is the case:

(a) $\bar{\bar{w}}'_a = nil$

(b) $\bar{\bar{w}}'_b = nil$

(c) There exists an h, $1 \leqslant h \leqslant k$, s.t. $\bar{\bar{w}}_h \neq nil$ and $\bar{\bar{w}}'_h = nil$.

Otherwise, the support increases after this symbol is read, contrary to the assumption of the saturation of $\bar{\bar{W}}$.

In case (a): The relational chain $r(x_1, x_{2\bar{n}}) \cdots r(x_{s+1}, x_{t-1}) \cdots r(x_{t-1}, x_j) \cdots r(x_{2\bar{n}}, x_1)$, for some j, $j \neq s + 1$, is accepted by A, even though it is not in C.

In case (b): Similar.

In case (c): Either there exist d, e s.t. $r(x_1, x_{2\bar{n}} \cdots r(x_d, x_e) \cdots r(x_{e'}, x_d) \cdots r(x_{2\bar{n}}, x_1)$, with $e' \neq e$, is accepted though not in C, or $r(x_1, x_{2\bar{n}}) \cdots r(x_d, x_e) \cdots r(x_e, x_{d'}) \cdots r(x_{2\bar{n}}, x_1)$, with $d' \neq d$, is accepted, though not in C.

Thus, in all cases a contradiction is reached, concluding the proof. Note that under the natural logical semantics, this language degenerates when taking x_1 and x_{n+1} as free variables, since

$$\exists x_2 x_3 : [r(x_1, x_2)\, r(x_3, x_4)\, r(x_4, x_3)\, r(x_2, x_1)]$$

is logically equivalent to

$$\exists x_2 x_3 x_5 x_6 : [r(x_1, x_2)\, r(x_5, x_6)\, r(x_3, x_4)\, r(x_4, x_3)\, r(x_6, x_5)\, r(x_2, x_1)]. \quad \blacksquare$$

This proof is a special case of the more general *pumping lemma* presented below, which constitutes a general necessary condition for relational regularity. However, both its statement and its proof transcend their counterparts in ordinary regular formal languages, due to the role of the variables. This reflects the fact that the recognizing automaton has more information than just the state, namely the contents of the W registers, up to a variant.

As a preliminary effort, we first extend the notion of variance also to gegister vectors and prove the main result about this notion to be used in the proof of the pumping lemma. However, this result may have other applications, and is therefore stated and proved separately.

DEFINITION. Two register vectors (of the same length k) W, V are *variants* of each other if

(1) $\{i \mid w_i = nil\} = \{i \mid v_i = nil\}$, and

(2) For every $1 \leqslant i, j \leqslant k : w_i = w_j \Leftrightarrow v_i = v_j$.

We denote this relation by $W \operatorname{sim} V$. From this definition it follows, that if $W \operatorname{sim} V$, then there exists a (1–1) substitution θ, s.t. the left-variables of θ are values of W-registers, the right-variables of θ are values of V-registers, and $W\theta = V$ (and similarly in the other direction, by symmetry). The notation $W\theta$ is self-explanatory, assuming that $w_i = nil \Rightarrow w_i\theta = nil$ by convention.

DEFINITION. Two configurations $\gamma = \langle q, W \rangle$, $\gamma' = \langle q', W' \rangle$ are *similar* iff $q = q'$ and $W \operatorname{sim} W'$.

Similarity of configurations is the natural generalization of state-equality (cycle in the graph) of usual finite-state automata.

Note that for each FSDA A, the number of non-similar configurations is bounded by a function of $|Q|$ and k; hence "long" computations have to repeat similar configurations.

As we show, similar configurations contain the same information about the continuation of computations leaving them.

LEMMA (Similarity Invariance). *For any FSDA A, if there is a partial copputation π (over a relational chain c) from configuration $\langle q_1, W^1\rangle$ to configuration $\langle q_2, W^2\rangle$, then for every (1–1) substitution θ, there is a corresponding partial computation π' (over $c\theta$) from configuration $\langle q_1, W^1\theta\rangle$ to configuration $\langle q_2, W^2\theta\rangle$.*

Proof. Note that θ has no effect on the Σ symbols in c; thus the ability to follow a path in A on $c\theta$ depends only on the variables arrangement in $c\theta$. We proceed by induction on $l = |c|$.

Basis: $l = 1$. Let $c = r(x_i, x_j)$, and suppose the computation of A on c leads (in one step) from the configuration $\langle q_1, W^1\rangle$ to $\langle q_2, W^2\rangle$. Thus, there is a transition from q_1 involving $r(w_a, w_b)$ for some $1 \leqslant a, b \leqslant k$, and $w_{i_1}, ..., w_{i_m} \leftarrow nil$. One of the four following cases holds.

(a) $w_a^1 = nil \wedge a \in \{i_1, ..., i_m\} \wedge w_a^2 = nil$

(b) $w_a^1 = nil \wedge a \notin \{i_1, ..., i_m\} \wedge w_a^2 = x_i$

(c) $w_a^1 = x_i \wedge a \in \{i_1, ..., i_m\} \wedge w_a^2 = nil$

(d) $w_a^1 = x_i \wedge a \notin \{i_1, ..., i_m\} \wedge w_a^2 = x_i$.

For each case, we have to show that the same transition is possible from $(q_1, W^1\theta)$ on $r(x_i, x_j)\theta$ to (q_2, W'), s.t. $W' = W_2\theta$.

Case (a). Since $w_a^1 = nil$, we also have $w_a^1\theta = nil$, and hence the unification of w_a^1 with the first input variable is possible, upon which $w'_a = x_i\theta$, and after the nilification of w'_a (since $a \in \{i_1, ..., i_m\}$, we end with $w'_a = nil = w_a^2\theta$.

Case (b). Similar, but with no nilification, $w'_a = x_i\theta = w_a^2\theta$.

Case (c). Since $w_a^1 = x_i$, we have $w_a^1\theta = x_i\theta$, which indeed is the first input variable by assumption, hence the unification is possible. The remaining argument about nilification is as in case (a).

Case (d). Similar.

Next, the unification of the second register w_b has to be accounted for.

For the case $a \neq b$, the argument is similar, the same four possibilities arising independently also for b. The only remaining case is the diagonal case $a = b$. In this case, $w_a^2 = w_a^2\theta$ is already known for the intermediate stage. Since the original double unification succeeded, the original input was $r(x_i, x_i)$. Hence, the new input is $r(x_i\theta, x_i\theta)$, and the test of equality succeeds here too. Nilification is treated as before.

Induction hypothesis: Assume the claim for $l \leqslant n$.

Induction step: Let c be s.t. $|c| = n+1$. Let c be split into a concatenation $c = c_1 c_2$ with $1 \leqslant |c_i| \leqslant n$, $i = 1, 2$. Thus, there is a partial computation of A, leading from $\langle q_1, W^1 \rangle$ to $\langle \bar{q}, \bar{W} \rangle$ on c_1, and another partial computation leading from $\langle \bar{q}, \bar{W} \rangle$ to $\langle q_2, W^2 \rangle$ on c_2. Since $|c_1| \leqslant n$, by the induction hypothesis, there is a partial computation leading from $\langle q_1, W^1\theta \rangle$ to $\langle \bar{q}, \bar{W}\theta \rangle$ on $c_1\theta$. Similarly, there is another partial computation leading from $\langle \bar{q}, \bar{W}\theta \rangle$ to $\langle q_2, W^2\theta \rangle$. Combining these two partial computations yields the required partial computation leading from $\langle q_1, W^1\theta \rangle$ to $\langle q_2, W^2\theta \rangle$ on $(c_1\theta)(c_2\theta) = c\theta$. ∎

In order to state the pumping lemma, we introduce the following notation. Let c be a chain, m a natural number, and $S = \{\theta_i \mid 0 \leqslant i\}$ a set of substitutions with $\theta_0 =^{\text{df}} \varnothing$ (the identity substitution). We denote by $c^{(S,m)}$ the chain $c\theta_0 \cdots \theta_m$, then

$$p(c', c'', c''', S, m) = c'c''^{(S,0)}c''^{(S,1)} \cdots c''^{(S,m)}c'''^{(S,m)}$$

(called the mth pump w.r.t. S).

THEOREM (Pumping Lemma). *Let $C \in \mathbf{RR}$. Then there exists $n \geqslant 1$ such that for each $c \in C$ with $|c| \geqslant n$ there exists a decomposition $c = c'c''c'''$ such that*:

1. $|c'c''| \leqslant n$
2. $|c''| \geqslant 1$

3. *There exists a* 1–1 *substitution θ, every left-variable v_i of which satisfies the following condition*:

(a) *The left-variable v_i occurs in c'' and its corresponding right-variable u_i occurs in c' and $c'(c'''\theta)) \in C$.*

4. *There exists a set S of* 1–1 *substitutions s.t. each $\theta_m \in S$, $m \geqslant 1$, satisfies: for each of its left-variable v_i one of the following two conditions holds*:

(a) *The left-variable v_i occurs in $c'^{(S,m-1)}$ and its corresponding right-variable u_i occurs in $c''^{(S,m-1)}$, or*

(b) *The left-variable v_i occurs in $c''^{(S,m-1)}$ but not in $c'^{(S,m-1)}$, and its corresponding right-variable u_i is a* new *variable (not occurring at all in $\{\theta_j \mid j < m\}$). The new variables are selected with indices as low as possible, and $p(c', c'', c''', S, m) \in C$.*

Before presenting the proof of the pumping lemma, we give an example of its use as a necessary condition for being rr.

EXAMPLE. Let

$$C^* = \{r(x_1, x_{2n})\, r(x_2, x_{2n-1}) \cdots r(x_n, x_{n+1})\, r(x_{n+1}, x_n) \cdots r(x_{2n}, x_1) \mid n \geqslant 1\}.$$

We show that $C^* \notin \mathbf{RR}$. Assume that $C^* \in \mathbf{RR}$ and let n be the integer in the pumping lemma. Let $l > n$ and let

$$c = r(x_1, x_{2l})\, r(x_2, x_{2l-1}) \cdots r(x_l, x_{l+1})\, r(x_{l+1}, x_l) \cdots r(x_{2l}, x_1).$$

Each chain c in C^*, and in particular the c above, is naturally divided into two parts of the same length, subscripted by 1 and 2 respectively ($c = c_1 c_2$). The variables in c_2 appear in the opposite order of the variables in c_1. By the pumping lemma, c may be decomposed into $c'c''c'''$, where $|c'c''| \leqslant n$; therefore also $|c'c''| < l$, $|c'| \geqslant 1$, and there exists a set of 1–1 substitutions S as implied by clause 4 in the lemma. In particular, consider $m = 1$, i.e., a single upwards pumping. We show that the chain obtained by this pumping can not be decomposed into two parts as described above.

The chain c'' is a part of c_1 since $|c'c''| < l$ and $|c_1| = l$. Hence the variables which appear in c'' do not appear in c'. Therefore the variables $c'\theta_1$ are new variables, and distinct pairwise. By the definition of C^* it follows that $c'c''(c''\theta_1)$ is a prefix of pc_1', the first component of the decomposition of the new chain $pc =^{\mathrm{df}} p(c'\, c'', c''', S, 1)$. Let c_1''' be the longest prefix of c''' which is part of c_1. The substitution θ_1 does not affect any of the variables in c_1''', since they are all distinct pairwise from those that appear in c to their left, and thus $c'c''(c''\theta_1)(c_1'''\theta_1)$ is a prefix of pc_1. Let c_2''' be the suffix of c''' taken from c_2. Then

$$pc = p(c', c'', c''', S, 1) = c'c''(c''\theta_1)(c_1'''\theta_1)(c_2'''\theta_1),$$

in which

$$|c'| + |c''| + |c_1'''\theta_1| = l$$

$$|c''\theta_1| \geqslant 1$$

$$|c_2'''| = l.$$

Thus, the prefix $c'c''(c''\theta_1)(c_1'\theta_1)$ of pc_1 is of length greater than l. Still, $(c_2'''\theta_1)$, which is the candidate being the second part pc_2 of the chain, is of length l. Thus $pc = p(c', c'', c''', S, 1)$ is not in C^*, a contradiction. We conclude that $C^* \notin \mathbf{RR}$.

We now turn to the proof of the pumping lemma itself.

Proof. Let $A=(\Sigma, Q, q_0, F, W, T)$ be an FSDA s.t. $C=C(A)$ and let m_W be the number of register vectors which are not variants of each other pairwise. Thus, the number of configurations of A, up to similarity (of configurations), is bounded by $|Q| \cdot m_W$.

Let $n=|Q| \cdot m_W+1$ and let $c \in C$, s.t. $|c|>n$. During the computation of A on the prefix of length n of c, necesarily two similar configurations are reached. Let $\langle q, W^1\rangle$ and $\langle q, W^2\rangle$ be those configurations, s.t. $\langle q, W^1\rangle$ precedes $\langle q, W^2\rangle$.

Let c' be the prefix of c on which A reaches $\langle q, W^1\rangle$; let c'' be the "middle" part of c, starting at the end of c' and ending where A reaches, on that same computation, the configuration $\langle q, W^2\rangle$; and let c''' be the suffix of c, starting where c'' terminates. Therefore:

$$c=c'c''c''', \qquad |c''| \geqslant 1, \quad |c'c''| \leqslant n.$$

Since $\langle q, W^1\rangle$ and $\langle q, W^2\rangle$ are similar, there exist two (1–1) substitutions μ and η, s.t. $W^2\mu=W^1$ and $W^1\eta=W^2$.

First we prove the existence of θ as defined in (3). We derive θ from μ: $\theta=\{u_i/v_i \in \mu) \wedge (u_i \in W^2)\}$. Clearly, $W^2\theta=W^1$ and since μ is 1–1, θ is 1–1 as well. Every left variable u_i in θ is in W^2. Therefore u_i appears in the subchain $c'c''$. Since by the substitution's definition $u_i \neq v_i$, u_i appears in c''. $W^2\theta=W^1$, thus $v_i \in W^1$, which indicates that v_i appears in the prefix c'. Since there exists a computation of A on c''', starting in the configuration $\langle q, W^2\rangle$ and ending in the configuration which accepts the whole chain c (which therefore is a terminal configuration), there exists, by the similarity-invariance lemma, a computation of A on $c'''\theta$ starting in $\langle q, W^2\theta\rangle$ and ending on a variant configuration. Thus, reaching $\langle q, W^1\rangle$ on c', the computation on $c'''\theta$ may proceed and terminate in a terminal configuration on $c'(c'''\theta) \in \theta) \in C(A)$.

Next, we prove the existence of a set of 1–1 substitutions S as defined in (4).

We show the existence of each θ_{m+1} in S by proving a stronger claim:

CLAIM. *There exists a substitution θ_{m+1} as defined in (4). The computation on the relational chain $c'c''(c''\theta_1)\cdots c''(\theta_1\cdots\theta_m)$ terminates in the configuration $\langle q, W^1\theta_1\cdots\theta_m\theta_m\rangle$.*

Proof. By induction on m.

Basis: $m=1$. We derive a substitution θ_1' from η:

$$\theta_1'=\{u_i/v_i \mid (u_i/v_i \in \eta) \wedge (u_i \in W^1)\}.$$

Here again, $W^1\theta_1'=W^2$ and θ_1' is a 1–1 substitution.

Let θ_1 be the substitution $\theta_1 = \theta'_1 \cup \eta_1$, where

$$\eta_1 = \{u_i/t_i \mid (u_i \in c'') \wedge (u_i \notin c') \wedge \mathrm{Cond}(t_i)\},$$

where $\mathrm{Cond}(t_i)$ holds iff t_i is a new variable, which is not mentioned so far, indexed as low as possible. Since $u_i \notin c'$, u_i does not occur in W^1 and hence it is not a left variable in θ'_1. Thus θ_1 is well defined. Both θ'_1 and η_1 are 1–1 substitutions. Due to $\mathrm{Cond}(t_i)$, θ_1 is 1–1 as well. Since the expansion θ_1 of θ'_1 includes no left variable that appears in W^1, $W^1\theta_1 = W^2$.

Since there exists a computation of A on c'' from configuration $\langle q, W^1\rangle$ to configuration $\langle q, W^2\rangle$, according to the similarity-invariance lemma, there exists a computation over the relational chain $c''\theta_1$ from configuration $\langle q, W^1\theta_1\rangle$ (that is $\langle q, W^2\rangle$) to configuration $\langle q, W^2\theta_1\rangle$. There exists a compuation over c''' from configuration $\langle q, W^2\rangle$ to some configuration where c is accepted (hence, that configuration is terminal). Using the similarity-invariance lemma again, we conclude that there exists a computation over $c'''\theta_1$ from $\langle q, W^1\theta_1\rangle$ to a terminal configuration. Thus $c'c''(c''\theta_1)(c'''\theta_1) \in C$. Note that the computation over $c''\theta_1$ leads to the configuration $\langle q, W^1\theta_1\theta_1\rangle$.

Assumption: Assume the claim for m. Define θ_{m+1} by $\theta_{m+1} = \theta_m \cup \eta_m$, where $\eta_m = \{u_i/t_i \mid (u_i \in c''\theta_1 \cdots \theta_m) \wedge (u_i \notin c'\theta_1 \cdots \theta_m) \wedge \mathrm{Cond}(t_i)\}$. Since $u_i \notin c'\theta_1 \cdots \theta_m$, u_i does not appear in $W^1\theta_1 \cdots \theta_m$ and accordingly u_i is not a left variable in θ_m. Thus θ_{m+1} is well defined. Moreover, θ_{m+1} is a 1–1 substitution since both θ_m and η_m are 1–1 substitutions and the condition on the right variables $\mathrm{Cond}(t_i)$ preserves this attribute. The equality $W^1\theta_1 \cdots \theta_m\theta_m = W^1\theta_1 \cdots \theta_{m+1}$ holds because η_m contains no left variable that appears in $W^1\theta_1 \cdots \theta_m$.

The induction assumption on the existence of a computation over $c''\theta_1 \cdots \theta_m$ from configuration $\langle q, W^1\theta_1 \cdots \theta_m\rangle$ to the configuration $\langle q, W^1\theta_1 \cdots \theta_m\theta_m\rangle$ and the similarity-invariance lemma guarantee the existence of a compuation of $c''\theta_1 \cdots \theta_m\theta_{m+1}$ from configuration $\langle q, W^1\theta_1 \cdots \theta_m\theta_{m+1}\rangle$ to configuration $\langle q, W^1\theta_1 \cdots \theta_{m+1}\theta_{m+1}\rangle$. Since there exists a computation on $c'''\theta_1 \cdots \theta_m$ from configuration $\langle q, W^1\theta_1 \cdots \theta_m\theta_m\rangle$, that is, $\langle q, W^1\theta_1 \cdots \theta_m\theta_{m+1}\rangle$ which terminates in a terminal configuration, using the similarity-invariance lemma we may conclude that there exists a computation on $c'''\theta_1 \cdots \theta_{m+1}$ from $\langle q, W^1\theta_1 \cdots \theta_m\theta_{m+1}\theta_{m+1}\rangle$ ending in a terminal configuration. Thus $c'c''(c''\theta_1) \cdots (c''\theta_1, ..., \theta_{m+1})(c'''\theta_1 \cdots \theta_{m+1}) \in C$. ▌

We present another example of using the pumping lemma, this time with a downwards pumping.

EXAMPLE. Consider again the relational language

$$C=\{r(x_1,x_2)\cdots r(x_{m-1},x_m)\,r(x_m,x_{m-1})\cdots r(x_2,x_1)\mid m\geqslant 2\}.$$

We show that C is not *rr*. Suppose it is, and let $c=r(x_1,x_2)\cdots r(x_{l-1},x_l)$ $r(x_l,x_{l-1})\cdots r(x_2,x_1)$, for some $l>n+1$, for n as in the pumping lemma.

Once again, we can present c as $c=c_1c_2$, where the sequence of variables of c_2 is the mirror image of that of c_1. Clearly $|c_1|=l-1>n$. Thus, any decomposition $c=c'c''c'''$ satisfying the pumping lemma will have that $c'c''$ is a prefix of c_1. By the lemma (Clause 3), there exists a 1–1 substitution θ s.t. $pc=^{\mathrm{df}}c'(c'''\theta)\in C$, having a natural decomposition into $pc_1\,pc_2$ as above. Note that x_l (the "middle" variable) is the only internal variable occurring only twice in any $c\in C$.

By the properties of θ from the lemma, x_l occurs exactly twice also in pc. Thus, attempting to decompose pc into two parts as above, around the occurrence of x_l, will yield $|pc_1|<|pc_2|$ a contradiction. ∎

We now present several more results about the Datalog automata and their class **RR** of recognizable languages.

THEOREM. **RR** *is closed under union.*

Proof. We define an FSDA A which accepts $C(A_1)\cup C(A_2)$, where

$$A_1=(\Sigma^1,Q^1,q_0^1,F^1,W^1,T^1) \qquad \text{and} \qquad A_2=(\Sigma^2,Q^2,q_0^2,F^2,W^2,T^2).$$

We assume that $Q^1\cap Q^2=\varnothing$, and hence also that $q_0^1\neq q_0^2$ and $F^1\cap F^2=\varnothing$. Also, let $k^i=|W^i|$, $i=1,2$. $A=((\Sigma^1\cup\Sigma^2),Q,q_0,F,\bar{W},T)$, where $\bar{W}$ is the register vector which is the concatenation of W^1 and W^2:

$$\bar{w}_i \text{ is } w_i^1 \text{ for } 1\leqslant i\leqslant k^1, \qquad \text{and} \qquad \bar{w}_i \text{ is } w_{i-k^1}^2 \text{ for } k^1+1\leqslant i\leqslant k^1+k^2$$

$$Q=\{Q^1-\{q_0^1\}\}\cup\{Q^2-\{q_0^2\}\}\cup\{q_0,q_{01},q_{02}\}$$

$$F=\{F^1-\{q_0^1\}\}\cup\{F^2-\{q_0^2\}\}\cup\{q_0\mid q_0^1\in F^1\vee q_0^2\in F^2\}$$
$$\cup\{q_{01}\mid q_0^1\in F^1\}\cup\{q_{02}\mid q_0^2\in F^2\}.$$

Each transition in A_1 starting in q_0^1 is duplicated in A: one starting in q_0 and the other in q_{02}. Symmetrically, each transition in A_2, starting in q_0^2 is duplicated in A: one starting in q_0 and the other in q_{01}. Furthermore, the transitions in A_1 which end in q_0^1, end in q_{01} in A. The transitions in A_2 which end in q_0^2, end in q_{02} in A. The other transitions in A^1 and A^2 remain with no change except that in those from A^2, each w_i is changed into w_{i+k^1}.

We leave out the details of verifying this construction, which is similar to standard ones for union automata. ∎

At this time, we do not know whether **RR** is closed under intersection.

Observation. The set of expanded relational languages is not closed under complementation. We denote here by $\bar{C}$ the complement of the relational language C.

Let $C = \{r(x_1, x_1)\}$. Then, $r(x_1, x_3) \in \bar{C}$.

Assume that $\bar{C}$ is an expanded relational language. By definition, all the instances of $r(x_1, x_3)$ are also members of $\bar{C}$, and in particular $r(x_1, x_1) \in \bar{C}$, a contradiction.

THEOREM. **RR** is not closed under complementation.

Proof. Since $C = \{r(x_1, x_1)\}$ is rr, the claim is consequence of the above observation. ▌

Finally, we show that the **RR** class is not too large, and is contained in the class of languages generated by Horn-clauses when viewed as generative grammars of relational languages. Due to the examples presented above, which are generable by Horn-grammars but are not rr, the containment is proper.

THEOREM. *If $C \in$ **RR**, then there exists a Datalog program H_C generating C (when viewed as a Horn-grammar).*

Proof. Let $A = (\Sigma, Q, q_0, F, W, T)$, with $|W| = k$ be an FSDA s.t. $C = C(A)$. H_C has $|Q| + 1$ program predicates:

(a) C of arity 0.

(b) For each $q_i \in Q$ a program predicate Q_i of arity k.

Also, H_C has $|\Sigma|$ base predicates (with the same names), each of arity 2.

The program clauses are constructed according to T. Consider a transition

$$q \xrightarrow[w_{t_1}, ..., w_{t_m} \leftarrow nil]{r(w_i, w_j)} q'.$$

Let Q, Q' be the grammar predicates corresponding to q, q', respectively. Then the corresponding program clause is $Q(v_1, ..., v_k) \rightarrow r(v_i, v_j)$ $Q'(y_1, ..., y_k)$ such that, for each $1 \leqslant l \leqslant k$,

$$y_l = \begin{cases} u_l & l \in \{t_1, ..., t_m\} \\ v_l & \text{otherwise.} \end{cases}$$

If $q' \in F$, the following program clause is added: $q(v_1, ..., v_k) \rightarrow r(v_i, v_j)$.

H_C consists of the above program clauses, together with the initial clause $C \rightarrow Q_0(v_1, \ldots, v_k)$. Note that the direction of the arrows is reversed, to emphasize the view of the clauses as grammar rules.

We show that H_C derives $C(A)$ by an appeal to the following claim.

CLAIM. *There exists a computation π on c in A from configuration $\gamma_1 = \langle q_a, V^1 \rangle$ to configuration $\gamma_2 = \langle q_b, V^2 \rangle$ iff $Q_a(V_1^1, \ldots, V_k^1) \xrightarrow{|\pi|} cQ_b(V_1^2, \ldots, V_k^2)$, such that each nil value is viewed as a new variable.*

Proof. $\Leftarrow$ By induction on $|\pi|$.

Basis: $|\pi| = 1$. Then π is of the form

$$q_a \xrightarrow[w_{l_1}, \ldots, w_{l_m} \leftarrow nil]{r(w_i, w_j)} q_b$$

and hence c is a chain of the form $r(x_d, x_e)$. The construction of the rules in H_C guarantees the existence of the rule $R'\colon Q_a(v_1, \ldots, v_k) \rightarrow r(v_i, v_j)\, Q_b(y_1, \ldots, y_k)$ (with the same restriction on the variables as in the definition of the rule). It follows from the conditions on a transition execution that for variables' values $v_l = V_l^1$, $1 \leqslant l \leqslant k$, the rule R' is applicable and $y_l = V_l^2$, $1 \leqslant l \leqslant k$ holds.

Assumption. Assume the claim for $|\pi| \leqslant n$, $n \geqslant 1$. Let $c = c_1 c_2$ where $|c_1| = 1$ and $|c_2| \geqslant 1$, and let $\bar{\gamma} = \langle \bar{q}, \bar{V} \rangle$ be the configuration that is reached by computing c_1 (starting from γ_1).

By the assumption

$$Q_a(V_1^1, \ldots, V_k^1) \rightarrow c_1 \bar{Q}(\bar{V}_1, \ldots, \bar{V}_k).$$

The computation on c_2 leads from configuration $\langle \bar{q}, \bar{V} \rangle$ to γ_2. Its length is n; hence by the induction assumption

$$\bar{Q}(\bar{V}_1, \ldots, \bar{V}_k) \xrightarrow{n} c_2 Q_b(V_1^2, \ldots, V_k^2).$$

Therefore

$$Q_a(V_1^1, \ldots, V_k^1) \xrightarrow{n+1} c_1 c_2 Q_b(V_1^2, \ldots, V_k^2);$$

thus

$$Q_a(V_1^1, \ldots, V_k^1) \xrightarrow{n+1} cQ_b(V_1^2, \ldots, V_k^2).$$

$\Rightarrow$ We prove by induction on the length of the derivation that if

$$Q_a(v_1^1, \ldots, v_k^1) \xrightarrow{n} cQ_b(v_1^2, \ldots, v_k^2),$$

then there exists a computation of A on c of length n from configuration $\gamma_1 = \langle q_a, V^1 \rangle$ to configuration $\gamma_2 = \langle q_b, V^2 \rangle$.

Basis. The existence of a derivation of length 1 and the structure of H_C implies that the rule

$$Q_a(V^1_1, ..., V^1_k) \rightarrow r(v_i, v_j)\, Q_b(V^2_1, ..., V^2_k)$$

is used as the first step of the derivation. The existence of such a rule guarantees that a transition T of the form

$$q_a \xrightarrow[w_{t_1}, ..., w_{t_m} \leftarrow nil]{r(w_i, w_j)} q_b$$

exists in A. This rule is applicable according to the rules of unification; therefore using T as the first step of the computation from $\langle q_a, V^1 \rangle$ to $\langle q_b, V^2 \rangle$ is allowed.

Assumption. If $Q_a(V^1_1, ..., V^1_k) \xrightarrow{n} cQ_b(V^2_1, ..., V^2_k)$ then there exists a computation over c of length n in A from configuration $\langle q_a, V^1 \rangle$ to configuration $\langle q_b, V^2 \rangle$.

We prove the claim holds for a derivation of length $n+1$. Let $c = c_1 c_2$, where c_1 is of length 1, and let $Q_a(V^1_1, ..., V^1_k) \rightarrow c_1 \bar{Q}(\bar{V}_1, ..., \bar{V}_k)$ be the first step in the above derivation of c. From the assumption it follows that there exists a computation of length 1 in A which leads from $\langle q_a, V^1 \rangle$ to $\langle \bar{q}, \bar{V} \rangle$. Since $\bar{Q}(\bar{V}_1, ..., \bar{V}_k) \xrightarrow{n} c_2 Q_b(V^2_1, ..., V^2_k)$ again, the assumption guarantees that there exists a computation of length n in A from $\langle \bar{q}, \bar{V} \rangle$ to $\langle q_b, V^2 \rangle$.

This completes the proof of the claim. ∎

Next, note that a computation π of c in A from configuration $\langle q_a, V^1 \rangle$ to $\langle q_b, V^2 \rangle$ for $q_b \in F$ guarantees the existence of a derivation $Q_a(V^1_1, ..., V^1_k) \xrightarrow{*} c$ in H_C, since the last transition that is used in π may be replaced by a rule in H_C which derives nothing but the last relational symbol $r(x_i, x_j)$ in c, instead of the one which derives $r(x_i, x_j)$ together with

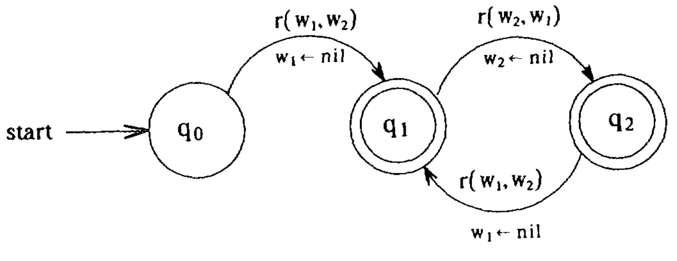

FIG. 4. An FSDA for transitive closure.

$Q_b(V_1^2, ..., V_k^2)$. On the other hand, the derivation $Q_a(V_1^1, ..., V_k^1) \xrightarrow{*} c$ in H_C guarantees a computation on c in A from configuration $\langle q_a, V^1 \rangle$ which terminates with a terminal state, according to the construction of H_C. To obtain the theorem let $q_a = q_0$ and $V^1 = \overline{nil}$, and let the derivation start with C (that is, $C \rightarrow Q_0(v_1, ..., v_k)$). ▌

EXAMPLE. The transitive closure of a base relation r, $C_{tr} = \{r(x_1, x_2) \cdots r(x_{n-1}, x_n) \mid n \geqslant 2\}$, is recognized by the FSDA in Fig. 4.

The corresponding H_{tr} is

$$\begin{aligned} &C \rightarrow Q_0(v_1, v_2) \\ &Q_0(v_1, v_2) \rightarrow r(v_1, v_2)\, Q_1(u_1, v_2) \\ &Q_1(v_1, v_2) \rightarrow r(v_2, v_1)\, Q_2(v_1, u_2) \\ &Q_2(v_1, v_2) \rightarrow r(v_1, v_2)\, Q_1(u_1, v_2) \\ &Q_1(v_1, v_2) \rightarrow r(v_2, v_1) \\ &Q_2(v_1, v_2) \rightarrow r(v_1, v_2) \\ &Q_0(v_1, v_2) \rightarrow r(v_1, v_2). \end{aligned}$$

Note that a much simpler Horn-grammar recognizing C_{tr} is

$$\begin{aligned} &R(v_1, v_2) \rightarrow r(v_1, v_3)\, R(v_3, v_2) \\ &R(v_1, v_2) \rightarrow r(v_1, v_2). \end{aligned}$$

4. CONCLUSION

In this paper we have introduced the (finite state) Unification automata model of computation, capable of recognizing relational languages. Several results about such automata and the languages recognizable by them were presented. To our knowledge, this a novel approach to Datalog queries. At this point, the results were not yet connected to the semantics of queries, e.g., to their efficient evaluation. In particular, a systematic way for designating the free variables in a relational chain is needed (when interpreting it logically), to connect our results with semantic properties of **RR**. This issues are now under investigation. Further development of the theory is also needed. For example, some decidability results of equivalence, containment, etc., are expected. Emptiness is trivially decidable due to the inexpressibility of inequalities and the absence of constants. Thus $C(A) = \varnothing$ iff A has no path from the initial state to an accepting state. It might also be interesting to find the natural analog of relational regular expressions, capable of denoting all and only relational regular languages. This issues are also under current research.

Finally, we mention that after having finished this work we learned about [La88], which presents a version of a pushdown Datalog automaton. However, the automaton considered by him is a computational device, computing queries by accessing the tables of the base relations, and not a recognizing device for "free" relational languages. Clearly the pda analog of our FSDA is worth pursuing.

ACKNOWLEDGMENTS

We wish to thank Oded Shmueli for some valuable remarks regarding the results reported in this paper. The part of the second author was partially supported by a Grant from the Israeli Academy of Science (basic research), by the Fund for the Promotion of Research in the Technion, and by a contribution of Aluf Benny Pelled.

RECEIVED December 12, 1988; FINAL MANUSCRIPT RECEIVED April 23, 1992

REFERENCES

[HU 79] HOPCROFT, J. E., AND ULLMAN, J. D. (1979), "Introduction to Automata Theory, Languages and Computations," Addison-Wesley, Reading, MA.

[KF 90] KAMINSKI, M., AND FRANCEZ, N. (1990), Finite memory automata, *in* "Proceedings 31th FOCS, St. Louis," Vol. 2.

[La 88] LANG, B. (1988), Datalog automata, *in* "Proceedings, 3rd International Conference on Data and Knowledge Base: Improving Usability and Responsiveness, Jerusalem."

[Lo 87] LLOYD, J. W. (1981), "Foundations of Logic Programming," Springer-Verlag, Berlin/New York; rev. ed. 1987.

[MW 88] MAIER, D., AND WARREN, D. S. (1988), "Computing with Logic (Logic Programming with Prolog)," Benjamin-Cummings, Menlo, CA/Reading, MA.

[Sh 87] SHMUELI, O. (1987), Decidability and expressiveness aspects of logic queries, *in* "ACM-PODS."

[UG 86] ULLMAN, J. D., AND VAN GELDER, J. (1986), Parallel complexity of logical query programs, *in* "Proceedings, 1986 FOCS Conference."