

Directed *st*-Connectivity is not Expressible in Symmetric Datalog ^{*}

László Egri¹, Benoît Larose², and Pascal Tesson³

¹ School of Computer Science, McGill University
legri1@cs.mcgill.ca

² Département d'Informatique et de Génie Logiciel, Université Laval
pascal.tesson@ift.ulaval.ca

³ Department of Mathematics and Statistics, Concordia University
larose@mathstat.concordia.ca

Abstract. We show that the directed *st*-connectivity problem cannot be expressed in symmetric Datalog, a fragment of Datalog introduced in [5]. It was shown there that symmetric Datalog programs can be evaluated in logarithmic space and that this fragment of Datalog captures logspace when augmented with negation, and an auxiliary successor relation S together with two constant symbols for the smallest and largest elements with respect to S . In contrast, undirected *st*-connectivity is expressible in symmetric Datalog and is in fact one of the simplest examples of the expressive power of this logic. It follows that undirected non-*st*-connectivity can be expressed in restricted symmetric monotone Krom SNP, whereas directed non-*st*-connectivity is only definable in the more expressive restricted monotone Krom SNP. By results of [8], the inexpressibility result for directed *st*-connectivity extends to a wide class of homomorphism problems that fail to meet a certain algebraic condition.

1 Introduction

Separating deterministic logspace from non-deterministic logspace remains an outstanding challenge of computational complexity. Because undirected and directed *st*-connectivity are respectively complete for L [10] and NL, the question is tied to the distinction between the hardness of these two problems. Ajtai and Fagin gave the first proof that *st*-connectivity is “harder” for directed graphs than for undirected graphs in a precise technical sense [2]. They showed that unlike undirected *st*-connectivity, directed *st*-connectivity is not definable in monadic Σ_1^1 . The result presented here is similar in spirit: we prove that unlike undirected *st*-connectivity, directed *st*-connectivity is not definable in *symmetric Datalog*.

This result is part of a research program investigating the descriptive complexity of constraint satisfaction problems or, equivalently, of the problem $\text{Hom}(\mathbf{B})$ of determining whether a homomorphism exists between an input relational structure \mathbf{A} and the fixed template \mathbf{B} . Feder and Vardi [6] showed that in a number of important cases, a polynomial time algorithm for the problem $\text{Hom}(\mathbf{B})$

^{*} Research supported in part by NSERC, FQRNT and CRM.

could be obtained by showing that the set of structures which are *not* homomorphic to \mathbf{B} is definable in Datalog (in the sequel, we abuse terminology and say simply that $\text{Hom}(\mathbf{B})$ is definable in Datalog). Pursuing that line of research, Dalmau proved that cases of the homomorphism problem that are known to lie in NL are all related to the linear fragment of Datalog. Finally, symmetric Datalog (itself a restriction of linear Datalog), was introduced in [5]. Symmetric Datalog programs can be evaluated in logspace and all cases of the homomorphism problem currently known to lie in L are in fact expressible in this fragment.

Over the last ten years, the complexity of $\text{Hom}(\mathbf{B})$ has also been studied through a powerful algebraic approach [3], whose description is beyond the scope of this paper. The algebraic angle of attack was initially developed in parallel to the aforementioned logical one, but the links between them have been increasingly apparent. In particular, it is conjectured in [8] that $\text{Hom}(\mathbf{B})$ (1) belongs to logspace iff (2) it is definable in symmetric Datalog iff (3) the algebra associated to \mathbf{B} satisfies a technical universal-algebraic condition. The implication (2) \rightarrow (1) is a consequence of Reingold’s theorem [10]. The inexpressibility of directed *st*-connectivity in symmetric Datalog is the missing piece of a puzzle establishing the implication (2) \rightarrow (3). Note that (1) \leftrightarrow (3) (or (1) \leftrightarrow (2)) can only hold if $L \neq \text{NL}$, whereas the equivalence of (2) and (3) may still hold if $L = \text{NL}$.

The second order logic fragments restricted monotone Krom SNP and restricted symmetric monotone Krom SNP [4, 5, 7] were shown to be equivalent to linear Datalog and symmetric Datalog respectively. From our main theorem, it follows that undirected non-*st*-connectivity can be expressed in both second order logic fragments while directed non-*st*-connectivity is only definable in restricted monotone Krom SNP. Grädel showed that a logic closely related to symmetric monotone Krom SNP captures Logspace^4 in the presence of an auxiliary successor relation S and constant symbols for the smallest and largest element with respect to S . Similarly, symmetric Datalog captures logarithmic space if it is augmented with negation, an auxiliary successor relation S and constant symbols for the smallest and largest elements with respect to S [5]. Separating the complexity classes L and NL is equivalent to extending our inexpressibility result and showing that directed *st*-connectivity is inexpressible in symmetric Datalog even in the presence of negation and an auxiliary successor relation.

In the remainder of this section, we review the basic notions required for the exposition of our results. Section 2 introduces the key technical ingredients of our arguments and Section 3 presents a sketch of the proof of our main result. Because of space restrictions, most technical arguments are omitted in this extended abstract.

1.1 Relational Structures and Homomorphisms

A *vocabulary* is a finite set of relation symbols. In the following, τ denotes a vocabulary. Every relation symbol R in τ has an associated *arity* r . A *relational structure* \mathbf{A} over the vocabulary τ , or simply a τ -*structure* consists of a set A

⁴ Grädel’s result uses the class co-SL now known to coincide with L.

called the *universe* or *domain* of \mathbf{A} , and a relation $R^{\mathbf{A}} \subseteq A^r$ for every relation symbol $R \in \tau$, where r is the arity of R . We use boldface letters to denote relational structures.

Definition 1. Let $\mathbf{A} = \langle A; R_1^{\mathbf{A}}, \dots, R_q^{\mathbf{A}} \rangle$ and $\mathbf{B} = \langle B; R_1^{\mathbf{B}}, \dots, R_q^{\mathbf{B}} \rangle$ be relational structures over the same vocabulary. A function $h : A \rightarrow B$ is called a homomorphism from \mathbf{A} to \mathbf{B} if $\langle h(a_1), \dots, h(a_{r_i}) \rangle \in R_i^{\mathbf{B}}$ whenever $\langle a_1, \dots, a_{r_i} \rangle \in R_i^{\mathbf{A}}$, $1 \leq i \leq q$. We write $\mathbf{A} \xrightarrow{h} \mathbf{B}$ if h is a homomorphism from \mathbf{A} to \mathbf{B} and simply $\mathbf{A} \rightarrow \mathbf{B}$ if such an h exists.

For a fixed τ -structure \mathbf{B} , the *homomorphism problem* for \mathbf{B} , denoted $\text{Hom}(\mathbf{B})$, consists of determining whether a given τ -structure \mathbf{A} is homomorphic to \mathbf{B} . Alternatively, one can think of $\text{Hom}(\mathbf{B})$ as the set of such structures, i.e. $\text{Hom}(\mathbf{B}) = \{\mathbf{A} : \mathbf{A} \xrightarrow{h} \mathbf{B}\}$. For a graph H with distinguished sets S and T of start and target vertices, the directed ST -connectivity problem consists of determining if there is a directed path from some $s \in S$ to some $t \in T$. We view the triple (H, S, T) as a relational structure \mathbf{H} over the set of vertices with a binary edge-relation E and unary relations S and T . One can easily verify that a directed path from some $s \in S$ to some $t \in T$ exists in H iff there is no homomorphism from \mathbf{H} to the two-element structure \mathbf{B} defined by $E^{\mathbf{B}} = \{(0, 0), (0, 1), (1, 1)\}$, $S^{\mathbf{B}} = \{1\}$ and $T^{\mathbf{B}} = \{0\}$. Similarly, there is an undirected ST -path in H iff \mathbf{H} is not homomorphic to the two-element structure \mathbf{C} defined as $E^{\mathbf{C}} = \{(0, 0), (1, 1)\}$, $S^{\mathbf{C}} = \{1\}$ and $T^{\mathbf{C}} = \{0\}$. In the sequel, we therefore regard (un)directed ST -connectivity as a homomorphism problem.

1.2 Datalog

Datalog is a query and rule language for deductive databases. A Datalog program \mathcal{D} over a vocabulary τ is a finite set of rules of the form $h \leftarrow b_1; \dots; b_m$ where h and each b_i are atomic formulas $R_j(v_1, \dots, v_k)$. We say that h is the *head* of the rule and that $b_1; \dots; b_m$ is its *body*. Relational predicates R_j which appear in the head of some rule of \mathcal{D} are called *intensional database predicates (IDBs)* and are not part of the vocabulary τ . All other relational predicates are called *extensional database predicates (EDBs)* and are in τ .

A rule of \mathcal{D} is *linear* if its body contains at most one IDB and is *non-recursive* if its body contains only EDBs. A linear but recursive rule is of the form $I_1(\bar{x}) \leftarrow I_2(\bar{y}); E_1(\bar{z}_1); \dots; E_k(\bar{z}_k)$ where I_1, I_2 are IDBs and the E_i are EDBs⁵. Each such rule has a *symmetric* $I_2(\bar{y}) \leftarrow I_1(\bar{x}); E_1(\bar{z}_1); \dots; E_k(\bar{z}_k)$. A Datalog program is *non-recursive* if all its rules are non-recursive, *linear* if all its rules are linear and *symmetric* if it is linear and if the symmetric of each recursive rule of \mathcal{D} is also a rule of \mathcal{D} . We further say that \mathcal{D} has *width* (j, k) if each rule of \mathcal{D} has at most k variables and at most j variables in the head.

A Datalog program \mathcal{D} takes a τ -structure \mathbf{A} as input and returns a structure $\mathcal{D}(\mathbf{A})$ over the vocabulary $\tau' = \tau \cup \{I : I \text{ is an IDB in } \mathcal{D}\}$. We also want to

⁵ Note that the variables occurring in $\bar{x}, \bar{y}, \bar{z}_i$ are not necessarily distinct.

view a Datalog program as being able to accept or reject an input τ -structure and this is achieved by choosing one of the IDB's of \mathcal{D} as the *goal predicate*: the τ -structure \mathbf{A} is *accepted by \mathcal{D}* if the goal predicate G is non-empty in $\mathcal{D}(\mathbf{A})$.

The semantics of Datalog are very intuitive and we only illustrate them through an example. A formal definition can be found, for example, in [4, 9]. Consider the problem of two-coloring. An undirected graph is two-colorable if and only if it contains no cycles of odd length. The following Datalog program \mathcal{D} defines two-coloring because the goal predicate becomes non-empty if and only if the input graph contains an odd cycle.

$$\begin{aligned} O(x, y) &\leftarrow E(x, y) \\ O(x, y) &\leftarrow O(x, w); E(w, z); E(z, y) \\ O(x, w) &\leftarrow O(x, y); E(w, z); E(z, y) \\ G &\leftarrow O(x, x) \end{aligned}$$

Here E is the binary EDB representing the adjacency relation in the input graph, O is a binary IDB whose intended meaning is “there exists an odd-length path from x to y ” and G is the 0-ary goal predicate. Intuitively, the program first finds a path of length one using the only non-recursive rule and then iteratively finds paths of higher odd lengths using the middle two rules. Whenever the path begins and ends at the same vertex x , the goal predicate becomes non-empty indicating the presence of a cycle of odd length.

Note that the two middle rules form a symmetric pair. In the above description, we have not included the symmetric of the last rule. In fact, the fairly counterintuitive rule $O(x, x) \leftarrow G$ can be added to the program without changing the class of structures accepted by the program since the rule only becomes relevant if an odd cycle has already been detected in the graph.

Assume that a program \mathcal{D} accepts a structure \mathbf{A} . Intuitively, a *derivation tree* over \mathbf{A} is a representation of the “proof” that \mathcal{D} accepts \mathbf{A} . Consider, for example, the following (linear) Datalog program \mathcal{D} over the vocabulary consisting of a binary relation symbol E and two unary relation symbols S and T :

$$\begin{aligned} I(y) &\leftarrow S(y) \\ I(y) &\leftarrow I(x); E(x, y) \\ G &\leftarrow I(y); T(y) \end{aligned}$$

We choose G as the goal predicate. One can verify that an input structure \mathbf{A} is accepted if and only if there is a path in the graph \mathbf{A} from a vertex in S to a vertex in T . Let \mathbf{A} be the input structure in Figure 1. Notice that \mathcal{D} accepts \mathbf{A} because it contains a path v_5, v_6, v_3, v_4 from a vertex in S to a vertex in T . Therefore one possible derivation tree for \mathcal{D} over \mathbf{A} is shown in Figure 1. Intuitively, the derivation tree “follows” the path from v_5 to v_4 .

Formally, a *derivation tree* \mathcal{T} for a Datalog program \mathcal{D} over a structure \mathbf{A} is a tree such that (1) the root of \mathcal{T} is the goal predicate; (2) an internal node (including the root) together with its children correspond to a rule \mathcal{R} of \mathcal{D} in the

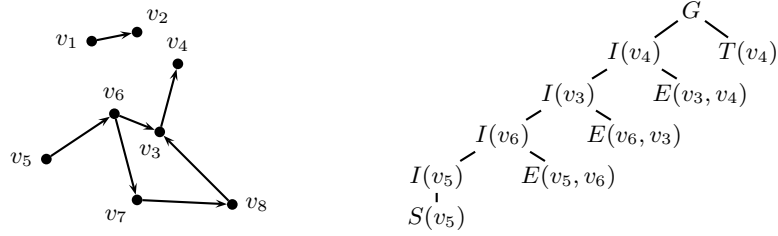


Fig. 1: The input structure \mathbf{A} with $S = \{\langle v_5 \rangle\}$ and $T = \{\langle v_4 \rangle\}$ and the corresponding derivation tree.

following sense. The internal node is the head IDB of \mathcal{R} and the children are the predicates in the body of \mathcal{R} ; (3) an internal node of \mathcal{T} is an IDB predicate $I(\bar{a})$ together with an instantiation of variables where if I has arity r then $\bar{a} \in A^r$; (4) the predicate children of a parent node inherit an instantiation to elements of A from their parents; (5) the leaf nodes of \mathcal{T} are EDB predicates $E(\bar{b})$ such that if E has arity s then $\bar{b} \in A^s$ and $\bar{b} \in E^{\mathbf{A}}$. By design, a derivation tree for a Datalog program \mathcal{D} and a structure \mathbf{A} exist if and only if \mathcal{D} accepts \mathbf{A} . For linear and symmetric Datalog we use the term *derivation path* instead of derivation tree because the IDBs in a derivation tree form a path.

Let τ consist of a single binary relation E . We say that a symmetric Datalog program \mathcal{D} *computes reflexive transitive closure* if \mathcal{D} has a binary IDB G such that for any τ -structure \mathbf{A} (i.e. a digraph) we have that $G^{\mathcal{D}(\mathbf{A})}$ is the reflexive transitive closure of $E^{\mathbf{A}}$. For this case, we slightly abuse terminology and call G the goal predicate. We show (Lemma 11) that there exist a symmetric \mathcal{D} computing reflexive transitive closure iff directed ST -connectivity is expressible in symmetric Datalog. Although our main objective is the second statement, our proof establishes the former statement for technical convenience.

In order to show that no symmetric Datalog program \mathcal{D} computes reflexive transitive closure, we use a form of pumping argument reminiscent of [1]. Roughly speaking, we prove that if \mathcal{D} is complete (i.e. if for all \mathbf{H} we have $\langle u, v \rangle \in G^{\mathcal{D}(\mathbf{H})}$ whenever $\langle u, v \rangle$ is in the reflexive transitive closure of $E^{\mathbf{H}}$) then it must be unsound (i.e. there exists \mathbf{H}' such that $G^{\mathcal{D}(\mathbf{H}')}$ contains a pair $\langle u, v \rangle$ which is *not* in the closure of $E^{\mathbf{H}'}$). We consider the behavior of \mathcal{D} on an input \mathbf{H} which is a “sufficiently long” simple path $s \rightarrow h_1 \rightarrow \dots \rightarrow h_r \rightarrow t$. If \mathcal{D} is complete then the pair $\langle s, t \rangle$ must be in $G^{\mathcal{D}(\mathbf{H})}$ and there must exist a derivation path \mathcal{P} witnessing this fact. Now, consider a subpath of \mathcal{P} , say $I_3 - I_2 - I_1$. For example, suppose that by successively using rules \mathcal{R}_1 and \mathcal{R}_2 , the program derives $I_3(a, b)$ from $I_1(c)$ as follows: $I_2(c, a) \xleftarrow{\mathcal{R}_1} I_1(c); E(c, a)$ and $I_3(a, b) \xleftarrow{\mathcal{R}_2} I_2(a, c); E(a, b)$. Notice that since the symmetric rules $\mathcal{R}'_1, \mathcal{R}'_2$ are also in \mathcal{D} we can use \mathcal{R}'_2 followed by \mathcal{R}'_1 to re-obtain $I_1(c)$ from $I_3(a, b)$ and, from there, re-derive $I_3(a, b)$ from $I_1(c)$. In other words, we can artificially lengthen \mathcal{P} by replacing the subpath $I_3 - I_2 - I_1$ of \mathcal{P} by $I_3 - I_2 - I_1 - I_2 - I_3 - I_2 - I_1$. By using this “pumping” trick on carefully chosen subpaths, we obtain from \mathcal{P} a new derivation path \mathcal{P}' and then

construct from \mathcal{P}' a new digraph \mathbf{H}' such that \mathcal{P}' witnesses the membership of a pair $\langle u, v \rangle$ in $G^{\mathcal{D}(\mathbf{H}')}$ even though no path from u to v exists in \mathbf{H}' . In the next section, we formalize the above intuition by introducing mirror operators and zig-zags: mirroring corresponds to the lengthening process just described and zig-zags allow us to describe the form of the derivation path obtained through a sequence of mirroring operations.

2 Zig-Zags, Mirror Operators and Free Derivation Paths

A *min-max pair*, denoted by $[a, b]$, is a pair of integers a, b such that $a < b$. Similarly, an *index pair* $\langle i, j \rangle$ is a pair of integers such that $i < j$. A *zig-zag* is a sequence of integers $Z = t_1, \dots, t_p$ such that $|t_i - t_{i+1}| = 1$ for each $1 \leq i \leq p-1$. Given a min-max pair $[a, b]$ and a zig-zag Z , $\text{MaxP}_{[a,b]}(Z)$ denotes the set of all index pairs $\langle k, \ell \rangle$ such that (i) if $t \in \{t_k, t_{k+1}, \dots, t_\ell\}$ then $a \leq t \leq b$, (ii) $a, b \in \{t_k, t_{k+1}, \dots, t_\ell\}$ and (iii) neither $a \leq t_{k-1} \leq b$ nor $a \leq t_{\ell+1} \leq b$ holds. Let \mathcal{Z} denote the set of all zig-zags. A *mirror operator* $\mu_{[a,b],r} : \mathcal{Z} \rightarrow \mathcal{Z}$ is a function with a min-max pair parameter and $r \in \mathbb{Z}^+$. Let Z be a zig-zag. Then $\mu_{[a,b],r}(Z)$ is the zig-zag such that for each index pair $\langle i, j \rangle \in \text{MaxP}_{[a,b]}(Z)$ we insert r consecutive copies of the sequence $t_{j-1}, t_{j-2}, \dots, t_i, t_{i+1}, \dots, t_j$ after t_j in Z .

Our main theorem relies on corollaries 6 and 8 which, in turn, rely on Lemma 2. The proof of Lemma 2 is not difficult but laborious and is omitted.

Lemma 2. *Let $\mu(Z) = \mu_{[a,b],r_1}(Z)$ and $\nu(Z) = \mu_{[c,d],r_2}(Z)$ be mirror operators, and let Z be a zig-zag. Then*

$$\nu(\mu(Z)) = \mu(\nu(Z)).$$

2.1 The Free Derivation Path

A *free derivation path* is obtained from a derivation path by replacing the domain elements with the underlying variables and renaming all quantified variables to different names. For example, let \mathcal{D} be the symmetric Datalog program in Figure 2a. Let the input structure \mathbf{A} be the graph in Figure 2b together with the unary relations $S = \{s\}$ and $T = \{t\}$. Then a derivation path \mathcal{P} obtained from \mathcal{D} over the input \mathbf{A} is shown in Figure 2c. In Figure 2d we obtain the corresponding free derivation path by renaming each variable of \mathcal{P} such that the variables of an IDB and EDBs in the body of a rule inherit the variables of the head IDB and all other variables are renamed to new elements.

Let τ be the input vocabulary of a symmetric Datalog program \mathcal{D} and let \mathcal{F} be a free derivation path. We can associate a τ -structure \mathbf{F} with \mathcal{F} as follows. First, the domain F of \mathbf{F} consists of all the variables appearing in \mathcal{F} . Second, let $R \in \tau$ have arity r and put a tuple $\langle x_1, \dots, x_r \rangle \in F^r$ into the relation $R^{\mathbf{F}}$ if $R(x_1, \dots, x_r)$ is present in \mathcal{F} . Observe that \mathcal{F} is a derivation path for \mathbf{F} . \mathbf{F} is called the *free structure* associated with \mathcal{F} .

Given a zig-zag $Z = t_1, \dots, t_q$ and a free derivation path \mathcal{F} having q occurrences of IDBs, we construct a corresponding *labeled free derivation path* \mathcal{F}^Z

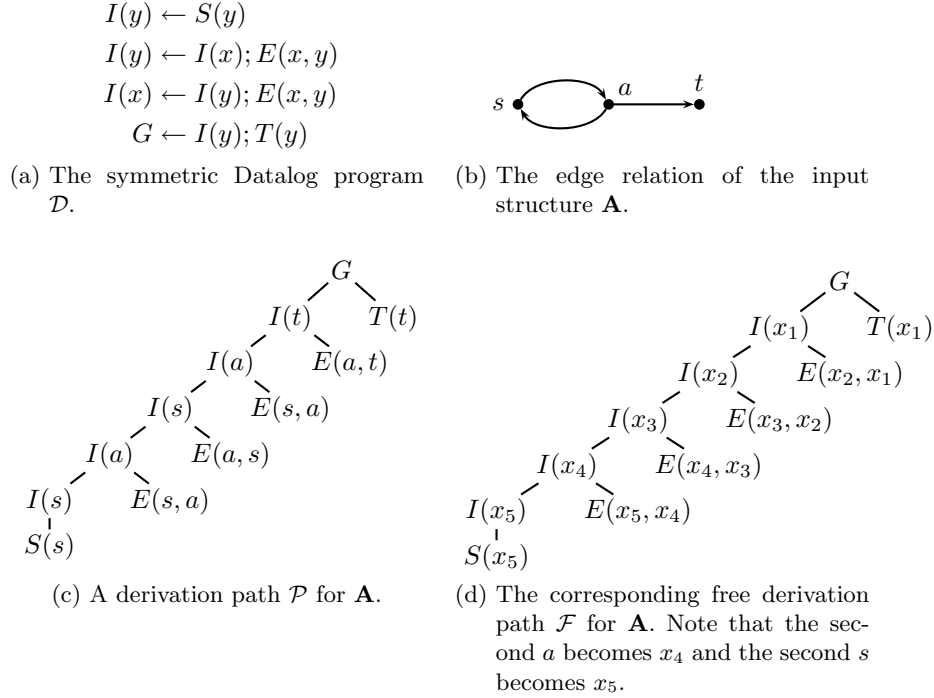


Fig. 2: The construction of a free derivation path.

as follows. Label the i -th IDB of \mathcal{F} starting from the goal predicate with t_i , $1 \leq i \leq q$. If an IDB I is labeled with t_i we denote it by I_{t_i} . Let \mathcal{F}^Z be a labeled free derivation path and let $\mu = \mu_{[a,b],r}$ be a mirror operator. We extend the action of μ to labeled free derivation paths in a natural way. First, for each index pair $\langle i, j \rangle \in \text{MaxP}_{[a,b]}(Z)$ we insert r consecutive copies of the sequence $I_{t_{j-1}}, I_{t_{j-2}}, \dots, I_{t_i}, I_{t_{i+1}}, \dots, I_{t_j}$ after I_{t_j} in \mathcal{F}^Z . Second, let $I_{t_k} - I_{t_{k+1}}$ be a segment of \mathcal{F}^Z and let \mathcal{R} be the corresponding rule. Whenever $I_{t_k} - I_{t_{k+1}}$ is inserted in the new derivation path the corresponding rule is \mathcal{R} and the parent of the EDBs of \mathcal{R} is I_{t_k} . On the other hand, whenever we insert $I_{t_{k+1}} - I_{t_k}$ the rule corresponding to $I_{t_{k+1}} - I_{t_k}$ is the symmetric rule \mathcal{R}' of \mathcal{R} and accordingly, the parent of the EDBs of \mathcal{R}' is $I_{t_{k+1}}$. Third, $\mu(\mathcal{F}^Z)$ is labeled with $\mu(Z)$. Finally, starting at the goal predicate, traverse the variables of $\mu(\mathcal{F}^Z)$ and rename them to a new name whenever possible. This ensures that $\mu(\mathcal{F}^Z)$ is free. Clearly, if \mathcal{F}^Z is a labeled free derivation path constructed from the rules of a symmetric program \mathcal{D} then $\mu(\mathcal{F}^Z)$ can also be constructed from the rules of \mathcal{D} . The definition of the free structure associated with a labeled free derivation path is analogous to the definition of the free structure associated with a free derivation path.

For instance, consider again the program \mathcal{D} in Figure 2a; a free derivation path \mathcal{F} for this program is shown in Figure 3a. Let $Z = 1, 2, 3$ be a zig-zag.

Then \mathcal{F}^Z is shown in Figure 3b. Figure 3c shows the intermediate step when the variables are not yet renamed to new elements and Figure 3d shows $\mu_{[2,3],1}(\mathcal{F}^Z)$.

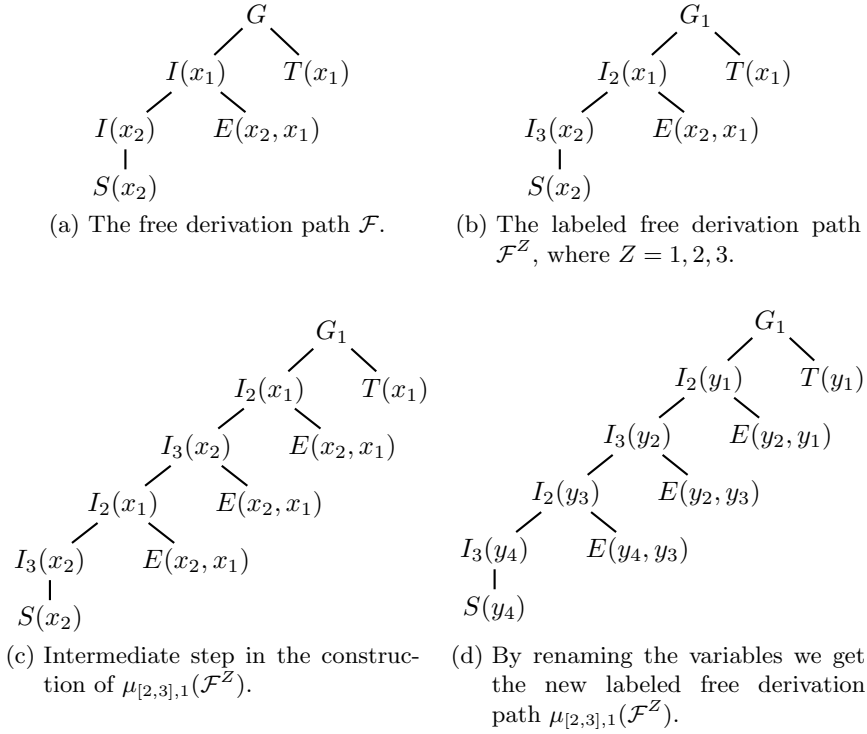


Fig. 3: Constructing $\mu_{[2,3],1}(\mathcal{F}^Z)$.

Before we present an outline of the general proof, we demonstrate the proof idea by showing that the symmetric (1, 2)-Datalog program \mathcal{D} in Figure 2a cannot decide the directed ST -connectivity problem. Program \mathcal{D} is complete in the sense that it accepts any structure \mathbf{A} in which there exist vertices $s \in S^{\mathbf{A}}$ and $t \in T^{\mathbf{A}}$ such that there is a path from s to t in $E^{\mathbf{A}}$. In fact, \mathcal{D} decides the undirected ST -connectivity problem.

Observe that \mathcal{D} accepts the directed ST -connectivity instance $\langle U; S, T, E \rangle$, where $U = \{u, v\}$, $S = \{u\}$, $T = \{v\}$, $E = \{\langle u, v \rangle\}$. There could be many corresponding derivation paths but pick the one that contains only a single application of a recursive rule. Then the corresponding free derivation path is the one in Figure 3a and the corresponding free structure is \mathbf{F}_1 in Figure 4a. Applying $\mu_{[2,3],1}$ and constructing the free structure yields \mathbf{F}_2 in Figure 4b which is also accepted by \mathcal{D} even though there is no path from y_4 to y_1 .

This trick can also be applied to any (1, k)-Datalog program. The difficulty is to generalize this argument to (j, k)-programs for $j > 1$. To see this challenge,

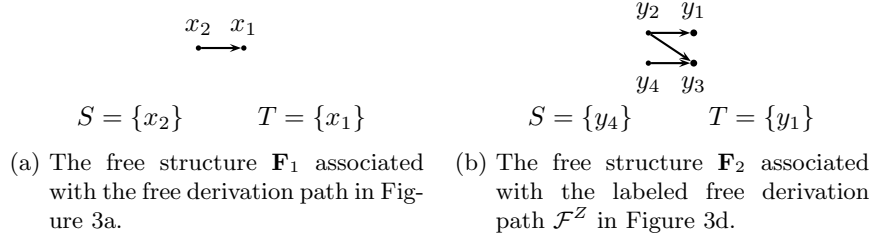


Fig. 4: Fooling a simple symmetric program by “pumping” the input structure.

assume that we input a directed path from a vertex u to a vertex v together with $S = \{u\}$ and $T = \{v\}$ to a symmetric (j, k) -Datalog program with $j > 1$. Then the free structure \mathbf{F} can be rather more complicated than before. First, \mathbf{F} could contain many different paths from u to v . For each path p we can find a mirror operator that produces a labeled free derivation path such that in the corresponding free structure p “disappears”. After we compose these mirror operators to obtain a new operator. Using the commutativity property stated in Corollaries 6 and 8 we show that this new operator produces a labeled free derivation path that is still accepted but in the corresponding free structure each path “disappears”. Second, notice that if $j = 1$ then when we traverse the edges of a path in a free structure we must either move in the free derivation path monotonically towards the goal predicate or monotonically away from the goal predicate. When $j > 1$ this is not the case any more. The location of the edges of the path in the free structure can be much more “disordered” in the free derivation path. This is why we need to mirror a segment of a labeled free derivation path more than once, i.e. to set r greater than 1 in a mirror operator $\mu_{[a,b],r}$.

3 Proof of the Main Theorems: Outline

We provide in this section an overview of the proof of the main theorems. Due to space restrictions, we simply describe the main technical results and show how they can be assembled to obtain Theorem 3 below and its alternative formulation Theorem 12.

Theorem 3. *No symmetric Datalog program \mathcal{D} computes reflexive transitive closure.*

As noted at the end of the previous section, our proof revolves around a generalization of the basic argument described through Figure 4.

Definition 4. *Let \mathcal{F}^Z and $\mathcal{F}^{Z'}$ be labeled free derivation paths and \mathbf{F} and \mathbf{F}' be the corresponding relational structures, respectively. We say that there is a homomorphism from \mathcal{F}^Z to $\mathcal{F}^{Z'}$ if there is a homomorphism h from \mathbf{F} to \mathbf{F}' , and we denote this by $\mathcal{F}^Z \xrightarrow{h} \mathcal{F}^{Z'}$.*

Lemma 5. Let \mathcal{F}^Z be a labeled free derivation path and $\mu_1, \mu_2, \dots, \mu_n$ be a sequence of mirror operators. Let $\mathcal{F}_0^Z, \mathcal{F}_1^Z, \dots, \mathcal{F}_n^Z$ be a sequence of labeled free derivation paths defined by $\mathcal{F}_0^Z = \mathcal{F}^Z$ and $\mathcal{F}_{i+1}^Z = \mu_{i+1}(\mathcal{F}_i^Z)$, $0 \leq i \leq n-1$. Then $\mathcal{F}_n^Z \xrightarrow{h} \mathcal{F}^Z$.

Proof. It clearly suffices to show that $\mathcal{F}_{i+1}^Z \xrightarrow{h} \mathcal{F}_i^Z$. Notice that when we apply μ_i to \mathcal{F}_i^Z to obtain \mathcal{F}_{i+1}^Z we insert additional sequences into \mathcal{F}_i^Z and we create new variables by renaming the original variables to new names whenever possible. Let h be the function that maps each new variable in \mathcal{F}_{i+1}^Z to the original in \mathcal{F}_i^Z . Clearly, h is a homomorphism.

For example, we obtained the labeled free derivation path in Figure 3d by applying $\mu_{[2,3],1}$ to the labeled free derivation path in Figure 3b. Define h to be $y_1 \mapsto x_1, y_2 \mapsto x_2, y_3 \mapsto x_1, y_4 \mapsto x_2$. Observe that h is a homomorphism from $\mu_{[2,3],1}(\mathcal{F}^Z)$ to \mathcal{F}^Z .

One of our main tools is Corollary 6, which follows directly from Lemma 2.

Corollary 6. Let $\mu(Z) = \mu_{[a,b],r_1}(Z)$ and $\nu(Z) = \mu_{[c,d],r_2}(Z)$ be mirror operators, let Z be a zig-zag, and let \mathcal{F} be a free derivation path. Then, up to renaming variables,

$$\nu(\mu(\mathcal{F}^Z)) = \mu(\nu(\mathcal{F}^Z)).$$

Given a labeled free derivation path \mathcal{F}^Z over τ we define $\mathcal{E}(\mathcal{F}^Z)$ as follows. Index the EDBs in \mathcal{F}^Z by their distance from the root $G(u, v)$ and let $\mathcal{E}(\mathcal{F}^Z)$ be the set of all indexed EDBs that appear in \mathcal{F}^Z . Furthermore, we say that a set $X \subseteq \mathcal{E}(\mathcal{F}^Z)$ contains a path from x_1 to x_n if there are indexed EDBs in X such that $E_{i_1}(x_1, x_2), E_{i_2}(x_2, x_3), \dots, E_{i_k}(x_{n-1}, x_n)$ for some variables x_2, x_3, \dots, x_{n-1} . The indices of the EDBs are used to differentiate between paths which would be the same if we removed the indices. For example, a labeled free derivation path \mathcal{F}^Z could contain two paths p_1 and p_2 whose EDBs are exactly the same but an EDB $E(x, y)$ of p_1 appears at a different level in \mathcal{F}^Z than the same EDB of p_2 . If we had no indices this difference would be lost.

Definition 7. Let $\mu_1, \mu_2, \dots, \mu_n$ be mirror operators, let M_i be the operator $M_i = \mu_i \circ \dots \circ \mu_2 \circ \mu_1$ and let $M = M_n$. Let $X \subseteq \mathcal{E}(\mathcal{F}^Z)$. We define the M -expansion $X' \subseteq \mathcal{E}(M(\mathcal{F}^Z))$ of X inductively as follows. The M_0 -expansion of X is X . Assume that the M_i -expansion of X is X_i . Consider the construction of $M_{i+1}(\mathcal{F}^Z)$ from $M_i(\mathcal{F}^Z)$. For each indexed EDB E_ℓ in X_i add the corresponding indexed EDB in $M_{i+1}(\mathcal{F}^Z)$ to X_{i+1} and any new copies of E_ℓ . (Note that indices of the EDBs in $M_{i+1}(\mathcal{F}^Z)$ are recomputed.) The M -expansion of X is X_n .

For example, the $\mu_{[2,3],1}$ -expansion of $\{E_2(x_2, x_1)\}$ in Figure 3b (after indexing the EDBs) is $\{E_2(y_2, y_1), E_3(y_2, y_3), E_4(y_4, y_3)\}$ in Figure 3d. We need the following corollary of Lemma 2.

Corollary 8. Let Z be a zig-zag, let μ, ν be mirror operators, and let \mathcal{F} be a free derivation path. Let $\mathcal{F}' = (\nu \circ \mu)(\mathcal{F}^Z) = (\mu \circ \nu)(\mathcal{F}^Z)$. Let $X \subseteq \mathcal{E}(\mathcal{F}^Z)$. Then in \mathcal{F}' , the $(\nu \circ \mu)$ -expansion of X is the same as the $(\mu \circ \nu)$ -expansion of X .

Finally, we also need the following two lemmas.

Lemma 9. *Let \mathcal{F}^Z be a labeled free derivation path and $M = \mu_n \circ \dots \circ \mu_2 \circ \mu_1$ where $\mu_1, \mu_2, \dots, \mu_n$ are mirror operators. Let u, v be the variables appearing in the goal predicate $G(u, v)$, and let $E_{u \not\rightarrow v} \subseteq \mathcal{E}(\mathcal{F}^Z)$ be a set that contains no path from u to v . Let $E'_{u \not\rightarrow v} \subseteq \mathcal{E}(M(\mathcal{F}^Z))$ be the M -expansion of $E_{u \not\rightarrow v}$. Then $E'_{u \not\rightarrow v}$ contains no path from u to v .*

Proof. Assume that $E'_{u \not\rightarrow v}$ contains a path from u to v . Notice that if $\mu_1(\mathcal{F}^Z)$ contains a path from u to v then so does \mathcal{F}^Z . Repeating this argument for all $i \geq 2$, we obtain a path in $E_{u \not\rightarrow v}$ and this leads to a contradiction.

Lemma 10. *Let \mathcal{F} be a labeled free derivation path originating from a symmetric (j, k) -Datalog program which has a binary goal predicate G . Assume that the top IDB of \mathcal{F} is $G(u, v)$. Let q be the number of IDBs in \mathcal{F} and consider the zig-zag $Z = 1, 2, \dots, q$. Define a function $L_y(x)$ recursively by setting $L_y(1) = 3(y - 1)$ and $L_y(x) \geq 4L_y(x - 1) + 6$. Assume that $\mathcal{E}(\mathcal{F}^Z)$ contains a path p from u to v of length at least ℓ where $\ell = L_k(j)$. Then there exists a mirror operator μ such that the μ -expansion of p in $\mathcal{E}(\mu(\mathcal{F}^Z))$ does not contain any path from u to v .*

We now have the intermediate results required to prove Theorem 3 which we restate for convenience.

Theorem 3. *No symmetric Datalog program \mathcal{D} computes reflexive transitive closure.*

Proof. Suppose for contradiction that \mathcal{D} is a symmetric Datalog program that computes transitive closure. Let \mathbf{B} be a structure with an edge relation $E^{\mathbf{B}}$ such that $E^{\mathbf{B}}$ is a simple path from a to b . Let the length ℓ of this path satisfy the length condition of Lemma 10. Obtain the free derivation path \mathcal{F} from the derivation path that witnesses the fact that \mathcal{D} derives $\langle a, b \rangle$. Assume that the variables in the binary goal G at the top of \mathcal{F} are u and v . Let q be the number of IDBs in \mathcal{F} and let Z be the zig-zag $1, 2, \dots, q$ be a zig-zag. Observe that any path from u to v in $\mathcal{E}(\mathcal{F}^Z)$ must have length exactly ℓ . Let $P = \{p_1, \dots, p_n\}$ be the set of all paths from u to v in $\mathcal{E}(\mathcal{F}^Z)$. For each $1 \leq i \leq n$, use Lemma 10 to find a mirror operator μ_i such that the μ_i -expansion of p_i does not contain a path from u to v . Let $\mathcal{F}^{\not\rightarrow} = (\mu_n \circ \mu_{n-1} \circ \dots \circ \mu_1)(\mathcal{F}^Z)$. Observe that by Corollary 6, $\mathcal{F}^{\not\rightarrow} = (\mu_n \circ \dots \circ \mu_{i+1} \circ \mu_{i-1} \circ \dots \circ \mu_1 \circ \mu_i)(\mathcal{F})$ for each $1 \leq i \leq n$. We claim that $\mathcal{E}(\mathcal{F}^{\not\rightarrow})$ does not contain any path from u to v .

For the sake of contradiction assume that $\mathcal{E}(\mathcal{F}^{\not\rightarrow})$ contains a path w from u to v . Let h be the homomorphism defined in Lemma 5 from $\mathcal{F}^{\not\rightarrow}$ to \mathcal{F}^Z . Then $h(w)$ is a path in $\mathcal{E}(\mathcal{F}^Z)$ and therefore $h(w) = p_i$ for some i . By construction, the μ_i -expansion of p_i in $\mathcal{E}(\mu_i(\mathcal{F}^Z))$ does not contain a path from u to v . Using Lemma 9 we have that the $(\mu_n \circ \dots \circ \mu_{i+1} \circ \mu_{i-1} \circ \dots \circ \mu_1 \circ \mu_i)$ -expansion of p_i in $\mathcal{E}(\mathcal{F}^{\not\rightarrow})$ does not contain a path from u to v . By Corollary 8 the $(\mu_n \circ \dots \circ \mu_{i+1} \circ \mu_{i-1} \circ \dots \circ \mu_1 \circ \mu_i)$ -expansion and the $(\mu_n \circ \mu_{n-1} \circ \dots \circ \mu_1)$ -expansion of p_i are the same, hence the $(\mu_n \circ \mu_{n-1} \circ \dots \circ \mu_1)$ -expansion of p_i in $\mathcal{F}^{\not\rightarrow}$ does not contain a path from u to v . This leads to a contradiction since w is such a path.

Lemma 11. *The following statements are equivalent:*

- (1) *Let $\tau = \{E\}$ where E is a binary relation symbol. There exists a symmetric Datalog program with binary goal predicate G such that for any τ -structure \mathbf{A} , after running the program on input \mathbf{A} , G is the reflexive transitive closure of $E^{\mathbf{A}}$.*
- (2) *Let $\sigma = \{S, T, E\}$ where E is a binary and S and T are unary relation symbols. Then there exists a symmetric Datalog program \mathcal{D} such that for any σ -structure \mathbf{B} , \mathcal{D} accepts if and only if there exist two vertices $s \in S^{\mathbf{B}}$ and $t \in T^{\mathbf{B}}$ such that there is a path from s to t in $E^{\mathbf{B}}$.*

We have thus proved the following theorem, the second half of which relies on results of [4] and [5] linking fragments of Datalog and monotone Krom SNP.

Theorem 12.

- (a) *Directed ST -connectivity is not definable in symmetric Datalog. More formally, let σ be a vocabulary as in Lemma 11. There is no symmetric Datalog program \mathcal{D} such that \mathcal{D} accepts an input σ -structure \mathbf{H} iff $E^{\mathbf{H}}$ contains a directed path from some $s \in S^{\mathbf{H}}$ to some $t \in T^{\mathbf{H}}$.*
- (b) *The complement of directed ST -connectivity cannot be defined in restricted symmetric monotone Krom SNP.*

Minor modifications to our arguments show that the above theorem still holds if we allow negation of EDBs in the Datalog programs.

Acknowledgment We thank the anonymous referees for their helpful comments on an earlier version of the paper.

References

1. F. Afrati, S. S. Cosmadakis, and M. Yannakakis. On Datalog vs. polynomial time. *J. Comput. Syst. Sci.*, 51(2):177–196, 1995.
2. M. Ajtai and R. Fagin. Reachability is harder for directed than for undirected finite graphs. *J. Symb. Log.*, 55(1):113–150, 1990.
3. D. Cohen and P. Jeavons. The complexity of constraint languages. In *Handbook of Constraint Programming*, pages 245–280, 2006.
4. V. Dalmau. Linear Datalog and bounded path duality of relational structures. *Logical Methods in Computer Science*, 1(1), 2005.
5. L. Egri, B. Larose, and P. Tesson. Symmetric Datalog and constraint satisfaction problems in logspace. In *LICS '07: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science*, pages 193–202, 2007.
6. T. Feder and M. Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1999.
7. E. Grädel. Capturing complexity classes by fragments of second-order logic. *Theor. Comput. Sci.*, 101(1):35–57, 1992.
8. B. Larose and P. Tesson. Universal algebra and hardness results for constraint satisfaction problems.⁶ In *ICALP*, pages 267–278, 2007.
9. L. Libkin. *Elements of finite model theory*. Springer, 2004.
10. O. Reingold. Undirected st-connectivity in log-space. pages 376–385, 2005.

⁶ Extended version is to appear in Theoretical Computer Science.