

The complexity of the list homomorphism problem for graphs

László Egri

School of Computer Science
McGill University, Montréal, Canada
laszlo.egri@mail.mcgill.ca

Andrei Krokhin

School of Engineering and Computing Sciences
Durham University, Durham, UK
andrei.krokhin@durham.ac.uk

Benoit Larose

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

Pascal Tesson

Department of Computer Science
Laval University, Quebec City, Canada
pascal.tesson@ift.ulaval.ca

September 3, 2010

Abstract

We completely classify the computational complexity of the list \mathbf{H} -colouring problem for graphs (with possible loops) in combinatorial and algebraic terms: for every graph \mathbf{H} , the problem is either \mathbf{NP} -complete, \mathbf{NL} -complete, \mathbf{L} -complete or is first-order definable; descriptive complexity equivalents are given as well via Datalog and its fragments. Our algebraic characterisations match important conjectures in the study of constraint satisfaction problems.

1 Introduction

Homomorphisms of graphs, i.e. edge-preserving mappings, generalise graph colourings, and can model a wide variety of combinatorial problems dealing with mappings and assignments [21]. Because of the richness of the homomorphism

framework, many computational aspects of graph homomorphisms have recently become the focus of much attention. In the *list \mathbf{H} -colouring* problem (for a fixed graph \mathbf{H}), one is given a graph \mathbf{G} and a list L_v of vertices of \mathbf{H} for each vertex v in \mathbf{G} , and the goal is to determine whether there is a homomorphism, i.e. an edge-preserving map, h from \mathbf{G} to \mathbf{H} such that $h(v) \in L_v$ for all v . The complexity of such problems has been studied by combinatorial methods, e.g., in [17, 18]. In this paper, we study the complexity of the list homomorphism problem for graphs in the wider context of classifying the complexity of constraint satisfaction problems (CSP), see [4, 19, 21]. It is well known that the CSP can be viewed as the problem of deciding whether there exists a homomorphism from a relational structure to another, thus naturally extending the graph homomorphism problem.

One line of CSP research studies the non-uniform CSP, in which the target (or template) structure \mathbf{T} is fixed and the question is whether there exists a homomorphism from an input structure to \mathbf{T} . Over the last years, much work has been done on classifying the complexity of this problem, denoted $\text{Hom}(\mathbf{T})$ or $\text{CSP}(\mathbf{T})$, with respect to the fixed target structure, see surveys [8, 9, 12, 21]. Classification here is understood with respect to both computational complexity (i.e. membership in a given complexity class such as \mathbf{P} , \mathbf{NL} , or \mathbf{L} , modulo standard assumptions) and descriptive complexity (i.e. definability of the class of all positive, or all negative, instances in a given logic).

The best-known classification results in this direction concern the distinction between polynomial-time solvable and \mathbf{NP} -complete CSPs. For example, a classical result of Hell and Nešetřil (see [21]) shows that, for a graph \mathbf{H} , $\text{Hom}(\mathbf{H})$ (aka \mathbf{H} -colouring) is tractable if \mathbf{H} is bipartite or admits a loop, and is \mathbf{NP} -complete otherwise, while Schaefer's dichotomy [31] proves that any Boolean CSP is either in \mathbf{P} or \mathbf{NP} -complete. Recent work [1] established a more precise classification in the Boolean case: if \mathbf{T} is a structure on $\{0, 1\}$ then $\text{CSP}(\mathbf{T})$ is either \mathbf{NP} -complete, \mathbf{P} -complete, \mathbf{NL} -complete, $\oplus\mathbf{L}$ -complete, \mathbf{L} -complete or in \mathbf{AC}^0 (see [2] or [30] for the definitions of complexity classes).

Much of the work concerning the descriptive complexity of CSPs is centred around the database-inspired logic programming language Datalog and its fragments (see [8, 13, 16, 19, 23]). Feder and Vardi initially showed [19] that a number of important tractable cases of $\text{CSP}(\mathbf{T})$ correspond to structures for which $\neg\text{CSP}(\mathbf{T})$ (the complement of $\text{CSP}(\mathbf{T})$) is definable in Datalog. Similar ties were uncovered more recently between the two fragments of Datalog known as linear and symmetric Datalog and structures \mathbf{T} for which $\text{CSP}(\mathbf{T})$ belongs to \mathbf{NL} and \mathbf{L} , respectively [13, 16]. Note that, for CSPs, definability in many interesting extensions of first-order logic or restrictions of second-order logic is known to be equivalent to definability in Datalog or one of its fragments (see the above papers).

Algebra, logic and combinatorics provide three angles of attack which have fueled progress in this classification effort [8, 9, 12, 21, 23]. The algebraic approach (see [9, 12]) links the complexity of $\text{CSP}(\mathbf{T})$ to the set of functions that preserve the relations in \mathbf{T} . In this framework, one associates to each \mathbf{T} an algebra $\mathbb{A}_{\mathbf{T}}$ and exploits the fact that the properties of $\mathbb{A}_{\mathbf{T}}$ completely determine

the complexity of $\text{CSP}(\mathbf{T})$. This angle of attack was crucial in establishing key results in the field (see, for example, [3, 6, 9]).

Tame Congruence Theory, a deep universal-algebraic framework first developed by Hobby and McKenzie in the mid 80's [22], classifies the local behaviour of finite algebras into five *types* (unary, affine, Boolean, lattice and semilattice.) It was recently shown (see [8, 9, 25]) that there is a strong connection between the computational and descriptive complexity of $\text{CSP}(\mathbf{T})$ and the set of types that appear in $\mathbb{A}_{\mathbf{T}}$ and its subalgebras. There are strong conditions involving types which are sufficient for NL-hardness, P-hardness and NP-hardness of $\text{CSP}(\mathbf{T})$ as well as for inexpressibility of $\neg\text{CSP}(\mathbf{T})$ in Datalog, linear Datalog and symmetric Datalog. These sufficient conditions are also suspected (and in some cases proved) to be necessary, under natural complexity-theoretic assumptions. For example, (a) the presence of unary type is known to imply NP-completeness, while its absence is conjectured to imply tractability (see [9]); (b) the absence of the unary and affine types was recently proved to be (unconditionally) equivalent to definability in Datalog [3]; (c) the absence of the unary, affine, and semilattice types is proved necessary, and suspected to be sufficient, for membership in NL and definability in linear Datalog [25]; (d) the absence of all types but Boolean is proved necessary, and suspected to be sufficient, for membership in L and definability in symmetric Datalog [25]. The strength of evidence varies from case to case and, in particular, the conjectured algebraic conditions concerning CSPs in NL and L (and, as mentioned above, linear and symmetric Datalog) still rest on relatively limited evidence [8, 10, 11, 13, 15, 14, 25].

The aim of the present paper is to show that these algebraic conditions are indeed sufficient and necessary in the special case of list \mathbf{H} -colouring for undirected graphs with possible loops, and to characterise, in this special case, the dividing lines in graph-theoretic terms. Note that our results provide the first complete classification of CSPs with a fixed template for a reasonably large class of structures outside the Boolean case (the above-mentioned [1]). One can view the list \mathbf{H} -colouring problem as a CSP where the template is the structure \mathbf{H}^L consisting of the binary (edge) relation of \mathbf{H} and all unary relations on H (i.e. every subset of H). Tractable list homomorphism problems for general structures were characterised in [6] in algebraic terms. The tractable cases for graphs were described in [18] in both combinatorial and (more specific) algebraic terms; the latter implies, when combined with a recent result [14], that in these cases $\neg\text{CSP}(\mathbf{H}^L)$ is definable in linear Datalog and therefore $\text{CSP}(\mathbf{H}^L)$ is in fact in NL. We complete the picture by refining this classification and showing that $\text{CSP}(\mathbf{H}^L)$ is either NP-complete, or NL-complete, or L-complete or in AC^0 (and in fact first-order definable). We also remark that the problem of recognising into which case the problem $\text{CSP}(\mathbf{H}^L)$ falls can be solved in polynomial time.

As we mentioned above, the distinction between NP-complete cases and those in NL follows from earlier work [14, 18], and the situation is similar with distinction between L-hard cases and those leading to membership in AC^0 [24, 25]. Therefore, the main body of technical work in the paper concerns the distinction between NL-hardness and membership in L. We give two equivalent characterisations of the class of graphs \mathbf{H} such that $\text{CSP}(\mathbf{H}^L)$ is in L. One

characterisation is via forbidden subgraphs (for example, the reflexive graphs in this class are exactly the (P_4, C_4) -free graphs, while the irreflexive ones are exactly the bipartite (P_6, C_6) -free graphs), while the other is via an inductive definition. The first characterisation is used to show that graphs outside of this class give rise to NL-hard problems; we do this by providing constructions witnessing the presence of a non-Boolean type in the algebras associated with the graphs. The second characterisation is used to prove positive results. We first provide operations in the associated algebra which satisfy certain identities; this allows us to show that the necessary condition on types is also sufficient in our case. We also use the inductive definition to demonstrate that the class of negative instances of the corresponding CSP is definable in symmetric Datalog, which implies membership of the CSP in L.

2 Preliminaries

2.1 Graphs and relational structures

A *signature* is a (finite) set of relation symbols, each symbol has an associated arity. A *structure* \mathbf{T} of signature τ consists of a set T , called the *universe* of \mathbf{T} , and a relation $R(\mathbf{T})$, on T , of the corresponding arity for each relation symbol $R \in \tau$. All structures in this paper are assumed to be finite, i.e. with finite universe. In the following we denote the underlying universe of a structure \mathbf{S} , \mathbf{T} , etc. by its roman equivalent S , T , etc. Let \mathbf{S} be a structure of the same signature as \mathbf{T} . A *homomorphism* from \mathbf{S} to \mathbf{T} is a map f from S to T such that $f(R(\mathbf{S})) \subseteq R(\mathbf{T})$ for each $R \in \tau$, i.e. we have $(f(a_1), \dots, f(a_r)) \in R(\mathbf{T})$ whenever $(a_1, \dots, a_r) \in R(\mathbf{S})$. In this case we write $f : \mathbf{S} \rightarrow \mathbf{T}$. A structure \mathbf{T} is called a *core* if every homomorphism from \mathbf{T} to itself is a permutation on T . We denote by $\text{CSP}(\mathbf{T})$ the class of all τ -structures \mathbf{S} that admit a homomorphism to \mathbf{T} , and by $\neg \text{CSP}(\mathbf{T})$ the complement of this class.

The *direct n -th power* of a τ -structure \mathbf{T} , denoted \mathbf{T}^n , is defined to have universe T^n and, for any (say m -ary) $R \in \tau$, $(\mathbf{a}_1, \dots, \mathbf{a}_m) \in R(\mathbf{T}^n)$ if and only if $(\mathbf{a}_1[i], \dots, \mathbf{a}_m[i]) \in R(\mathbf{T})$ for each $1 \leq i \leq n$. For a subset $I \subseteq T$, the *substructure induced by I on \mathbf{T}* is the structure \mathbf{I} with universe I and such that $R(\mathbf{I}) = R(\mathbf{T}) \cap I^m$ for every m -ary $R \in \tau$.

For the purposes of this paper, a *graph* is a relational structure $\mathbf{H} = \langle H; \theta \rangle$ where θ is a symmetric binary relation on H . The graph \mathbf{H} is *reflexive* (*irreflexive*) if $(x, x) \in \theta$ ($(x, x) \notin \theta$) for all $x \in H$. Given a graph \mathbf{H} , let S_1, \dots, S_k denote all subsets of H ; let \mathbf{H}^L be the relational structure obtained from \mathbf{H} by adding all the S_i as unary relations; more precisely, let τ be the signature that consists of one binary relational symbol θ and unary symbols R_i , $i = 1, \dots, k$. The τ -structure \mathbf{H}^L has universe H , $\theta(\mathbf{H}^L)$ is the edge relation of \mathbf{H} , and $R_i(\mathbf{H}^L) = S_i$ for all $i = 1, \dots, k$. It is easy to see that \mathbf{H}^L is a core. We call $\text{CSP}(\mathbf{H}^L)$ the *list homomorphism problem for \mathbf{H}* . Note that if \mathbf{G} is an instance of this problem then $\theta(\mathbf{G})$ can be considered as a digraph, but the directions of the arcs are unimportant because \mathbf{H} is undirected. Also, if an element $v \in G$ is

in $R_i(\mathbf{G})$ then this is equivalent to v having S_i as its list, so \mathbf{G} can be thought of as a digraph with \mathbf{H} -lists. Note that an element of \mathbf{G} can in principle have several \mathbf{H} -lists, which is equivalent to having their intersection as a single list.

In [18], a dichotomy result was proved, identifying bi-arc graphs as those whose list homomorphism problem is tractable, and others as giving rise to NP-complete problems. Bi-arc graphs are defined as follows. Fix a circle with two distinct specified points p and q . A bi-arc is a pair of arcs (N, S) on the circle such that N contains p but not q and S contains q but not p . A graph \mathbf{H} is a *bi-arc graph* if there is a family of bi-arcs $\{(N_x, S_x) : x \in H\}$ such that, for every $x, y \in H$, the following conditions hold: (i) if x and y are adjacent, then neither N_x intersects S_y nor N_y intersects S_x , and (ii) if x is not adjacent to y then both N_x intersects S_y and N_y intersects S_x . Equivalently, \mathbf{H} is a bi-arc graph if and only if the complement of the graph $\mathbf{H} \times \mathbf{K}_2$ is a circular arc graph (i.e., can be represented by arcs on a circle so that two vertices are adjacent if and only if the corresponding arcs intersect) [18].

2.2 Algebra

An n -ary operation on a set A is a map $f : A^n \rightarrow A$, a *projection* is an operation of the form $e_n^i(x_1, \dots, x_n) = x_i$ for some $1 \leq i \leq n$. Given an h -ary relation θ and an n -ary operation f on the same set A , we say that f *preserves* θ or that θ is *invariant* under f if the following holds: given any matrix M of size $h \times n$ whose columns are in θ , applying f to the rows of M will produce an h -tuple in θ . A *polymorphism* of a structure \mathbf{T} is an operation f that preserves each relation in \mathbf{T} ; in this case we also say that \mathbf{T} *admits* f . In other words, an n -ary polymorphism of \mathbf{T} is simply a homomorphism from \mathbf{T}^n to \mathbf{T} . For the special case of graphs, this means that if there is an edge between a_i and b_i for each $1 \leq i \leq n$ (where the a_i 's and b_i 's are not necessarily distinct) then there is an edge between $f(a_1, \dots, a_n)$ and $f(b_1, \dots, b_n)$.

An algebra is a pair $\mathbb{A} = \langle A; F \rangle$ where A is a set, and F is a family of finitary operations on A . With any structure \mathbf{T} , one associates an algebra $\mathbb{A}_{\mathbf{T}}$ whose universe is T and whose operations are all polymorphisms of \mathbf{T} . Given a graph \mathbf{H} , we let, for the ease of notation, \mathbb{H} denote the algebra associated with \mathbf{H}^L . An operation on a set is called *conservative* if it preserves all subsets of the set (as unary relations). So, the operations of \mathbb{H} are the conservative polymorphisms of \mathbf{H} . Polymorphisms can provide a convenient language when defining classes of graphs. For example, it was shown in [5] that a graph is a bi-arc graph if and only if it admits a conservative majority operation where a *majority* operation is a ternary operation m satisfying the identities $m(x, x, y) = m(x, y, x) = m(y, x, x) = x$.

In order to state some of our results, we need the following basic notions from universal algebra (see textbooks [22, 29] for more universal-algebraic background and [7, 12] for the basics of the connection between universal algebra and CSP). Let I be a signature, i.e. a set of operation symbols f each of a fixed arity; we use the term “signature” for both structures and algebras, this will cause no confusion. An *algebra* of signature I is a pair $\mathbb{A} = \langle A; F \rangle$ where A is a non-

empty set, the *universe* of \mathbb{A} , and $F = \{f^{\mathbb{A}} : f \in I\}$ is the set of *basic* operations (for each $f \in I$, $f^{\mathbb{A}}$ is an operation on A of the corresponding arity). The *term operations* of \mathbb{A} are the operations built from the operations in F and projections by using composition. The *polynomial operations* of \mathbb{A} are the operations built from the operations in F , the constant operations and projections by using composition. An algebra all of whose (basic or term) operations are conservative is called a *conservative algebra*. A *subalgebra* \mathbb{B} of an algebra \mathbb{A} consists of a subset B of A that is invariant under all operations of \mathbb{A} and the restrictions of the operations of \mathbb{A} to B . A *homomorphic image* of an algebra \mathbb{A} is an algebra \mathbb{C} which is similar to \mathbb{A} (i.e. with the same signature) and such that there is a surjective mapping $\psi : A \rightarrow C$ with $\psi(f^{\mathbb{A}}(a_1, \dots, a_r)) = f^{\mathbb{C}}(\psi(a_1), \dots, \psi(a_r))$ for all operations $f \in I$ and all tuples of elements of A . Direct products and powers of algebras are defined in a natural way, by taking direct product of universes and defining the operations to act component-wise. A class of similar algebras which is closed under formation of homomorphic images, subalgebras and direct products is called a *variety*. The *variety generated* by an algebra \mathbb{A} , denoted by $\mathcal{V}(\mathbb{A})$, is the smallest variety containing \mathbb{A} , it coincides with the class of all homomorphic images of subalgebras of direct powers of \mathbb{A} .

Tame Congruence Theory, as developed in [22], is a powerful tool for the analysis of finite algebras. Every finite algebra has a *typeset*, which describes (in a certain specified sense) the local behaviour of the algebra. It contains one or more of the following 5 *types*: (1) the *unary* type, (2) the *affine* type, (3) the *Boolean* type, (4) the *lattice* type and (5) the *semilattice* type. The numbering of the types is fixed, and they are often referred to by their numbers. The typeset of a variety \mathcal{V} , denoted $typ(\mathcal{V})$, is simply the union of typesets of all finite algebras in it. We note that there is a very tight connection between the kind of identities that are satisfied by the algebras in a variety and the types that are *admitted* or *omitted* by a variety, i.e. those types that do or do not appear in the typesets of algebras in the variety [22]. We will be mostly interested in type-omitting conditions for varieties of the form $\mathcal{V}(\mathbb{A}_{\mathbf{T}})$, and Corollary 3.2 of [32] says that in this case it is enough to consider the typesets of $\mathbb{A}_{\mathbf{T}}$ and its subalgebras. On the intuitive level, if \mathbf{T} is a core structure then the typeset $typ(\mathcal{V}(\mathbb{A}_{\mathbf{T}}))$ contains crucial information about the kind of relations that \mathbf{T} can or cannot simulate, thus implying lower/upper bounds on the complexity of $CSP(\mathbf{T})$.

The definitions of the types are rather technical in general, but they are simple enough for conservative algebras, and all algebras in this paper are conservative. Let $\mathbb{A} = \langle A, F \rangle$ be a conservative algebra and let $X = \{a, b\}$ be a two-element subset of A . By conservativity, every operation in F preserves X , so X is the universe of a subalgebra \mathbb{X} of \mathbb{A} . Identify a with 0 and b with 1, and think of operations on X as Boolean operations. Then X satisfies exactly one of the following five conditions (see [22]):

- The type of X (in \mathbb{A}) is *unary*, or **1**, if $f|_X$ is a projection for each $f \in F$.
- The type of X is *affine*, or **2**, if it is not unary and $f|_X$ is a linear operation

for each $f \in F$. Equivalently, the type of X is affine if the polynomial operations of \mathbb{X} are all linear Boolean operations.

- The type of X is *semilattice*, or **5**, if it is not unary and either each operation $f|_X$, $f \in F$, is the minimum of some of its arguments or each operation $f|_X$, $f \in F$, is the maximum of some of its arguments.
- The type of X is *lattice*, or **4**, if it is not semilattice, but all operations $f|_X$, $f \in F$, are monotone. Equivalently, the polynomial operations of \mathbb{X} are all monotone Boolean operations.
- The type of X is *Boolean*, or **3**, in all other cases, that is, if the family $\{f|_X \mid f \in F\}$ contains a non-linear operation and a non-monotone operation. Equivalently, the polynomial operations of \mathbb{X} are all possible Boolean operations.

The typeset of a conservative algebra \mathbb{A} is simply the union of types of two-element subsets of A . Consider the following ordering of the types: $\mathbf{1} < \mathbf{2} < \mathbf{3} > \mathbf{4} > \mathbf{5} > \mathbf{1}$. It can be easily derived from Corollary 3.2 of [32] that, for any type \mathbf{i} , the variety $\mathcal{V}(\mathbb{A})$ omits all types below \mathbf{i} (with respect to the above ordering) if and only if none of the two-element subsets of A has type below \mathbf{i} . Thus, for a structure of the form \mathbf{H}^L , the knowledge of how conservative polymorphisms of \mathbf{H} behave on two-elements subsets of H gives us all necessary information about the typeset of $\mathcal{V}(\mathbb{H})$.

In this paper, we use ternary operations f_1, \dots, f_n satisfying the following identities:

$$x = f_1(x, y, y) \tag{Id1}$$

$$f_i(x, x, y) = f_{i+1}(x, y, y) \text{ for all } i = 1, \dots, n-1 \tag{Id2}$$

$$f_n(x, x, y) = y. \tag{Id3}$$

The following lemma from [22] contains some type-omitting results that we use in this paper.

Lemma 1. *1. A finite algebra \mathbb{A} has term operations f_1, \dots, f_n , for some $n \geq 1$, satisfying identities (Id1)–(Id3) if and only if the variety $\mathcal{V}(\mathbb{A})$ omits the unary, lattice, and semilattice types.*

2. If a finite algebra \mathbb{A} has a majority term operation then $\mathcal{V}(\mathbb{A})$ omits the unary, affine, and semilattice types.

We remark in passing that operations satisfying identities (Id1)–(Id3) are also known to characterise a certain algebraic (congruence) condition called $(n+1)$ -permutability [22].

2.3 Datalog

Datalog is a query and rule language for deductive databases (see [23]). A Datalog program \mathcal{D} over a (relational) signature τ is a finite set of rules of the form

$h \leftarrow b_1 \wedge \dots \wedge b_m$ 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 $b_1 \wedge \dots \wedge b_m$ 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 signature τ . All other relational predicates are called *extensional database predicates (EDBs)* and are in τ . So, a Datalog program is a recursive specification of IDBs (from EDBs).

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 $I_1(\bar{x}) \leftarrow I_2(\bar{y}) \wedge E_1(\bar{z}_1) \wedge \dots \wedge E_k(\bar{z}_k)$ where I_1, I_2 are IDBs and the E_i are EDBs (note that the variables occurring in $\bar{x}, \bar{y}, \bar{z}_i$ are not necessarily distinct). For each such linear recursive rule the *symmetric* of that rule is defined as $I_2(\bar{y}) \leftarrow I_1(\bar{x}) \wedge E_1(\bar{z}_1) \wedge \dots \wedge E_k(\bar{z}_k)$. 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} .

A Datalog program \mathcal{D} takes a τ -structure \mathbf{A} as input and returns a structure $\mathcal{D}(\mathbf{A})$ over the signature $\tau' = \tau \cup \{I : I \text{ is an IDB in } \mathcal{D}\}$. The relations corresponding to τ are the same as in \mathbf{A} , while the new relations are recursively computed by \mathcal{D} , with semantics naturally obtained via least fixed-point of monotone operators. We also want to view a Datalog program as being able to accept or reject an input τ -structure and this is achieved by choosing one of the IDBs of \mathcal{D} as the *goal predicate*: the τ -structure \mathbf{A} is *accepted by* \mathcal{D} if the goal predicate is non-empty (or **true** if it is 0-ary) in $\mathcal{D}(\mathbf{A})$. Thus every Datalog program with a goal predicate defines a class of structures - those that are accepted by the program.

We illustrate the semantics of Datalog through an example, and a more formal treatment can be found, for example, in [13, 23], and other examples can be found in e.g. [8, 16]. It is well known and easy to see that the problem $\text{CSP}(\mathbf{K}_2)$, where \mathbf{K}_2 is the undirected edge, is the graph 2-colouring problem. As is well known, an undirected graph is not 2-colourable if and only if it contains a cycle of odd length. The following program \mathcal{D} defines (essentially) the class $\neg\text{CSP}(\mathbf{K}_2)$ because the goal predicate becomes non-empty (i.e. **true**) if and only if the input graph contains an odd cycle.

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

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 above program

works for graphs and, formally, inputs of $\text{CSP}(\mathbf{K}_2)$ are digraphs, but the above program can be easily modified to work for all digraphs.

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.

As illustrated above, when using Datalog to study $\text{CSP}(\mathbf{T})$, one usually speaks of the definability of $\neg\text{CSP}(\mathbf{T})$ in Datalog (i.e. by a Datalog program) or its fragments. This is because any class definable in Datalog must be closed under extension. As we mentioned before, any problem $\text{CSP}(\mathbf{T})$ is tractable if its complement is definable in Datalog, and all such structures were recently identified in [3]. Definability of $\neg\text{CSP}(\mathbf{T})$ in linear (symmetric) Datalog implies that $\text{CSP}(\mathbf{T})$ belongs to NL and L , respectively [13, 16]. As we discussed in Section 1, there is a connection between definability of CSPs in Datalog (and its fragments) and the presence/absence of types in the corresponding algebra (or variety).

Note that it follows from Lemma 1 and from the results in [25, 26] that if, for a core structure \mathbf{T} , $\neg\text{CSP}(\mathbf{T})$ is definable in symmetric Datalog then \mathbf{T} must admit, for some n , operations satisfying identities (Id1)–(Id3). The converse for $n = 1$ was proved in [15]. Moreover, with the result of [3], a conjecture from [25] can be restated as follows: for a core structure \mathbf{T} such that $\neg\text{CSP}(\mathbf{T})$ is definable in Datalog, \mathbf{T} admits operations satisfying (Id1)–(Id3) for some n if and only if $\neg\text{CSP}(\mathbf{T})$ is definable in symmetric Datalog.

3 Main results and proof outline

In this section we state our main results, Theorems 2, 4, and 5. Theorem 2 follows from known results (with a little help from Lemma 16), the proof of Theorem 5 is a relatively simple application of a result from [24], and the proof of Theorem 4 constitutes most of this paper.

Theorem 2. *Let \mathbf{H} be a graph.*

- *If $\mathcal{V}(\mathbb{H})$ admits the unary type, then $\neg\text{CSP}(\mathbf{H}^L)$ is not expressible in Datalog and $\text{CSP}(\mathbf{H}^L)$ is NP-complete (under first-order reductions);*
- *if $\mathcal{V}(\mathbb{H})$ omits the unary but admits the lattice type, then $\neg\text{CSP}(\mathbf{H}^L)$ is not expressible in symmetric Datalog but is expressible in linear Datalog, and $\text{CSP}(\mathbf{H}^L)$ is NL-complete (under first-order reductions).*

Proof. The first statement is shown in [25]. If $\mathcal{V}(\mathbb{H})$ omits the unary type, then \mathbf{H}^L admits a majority operation by Lemma 16 in Section 5 and then $\neg\text{CSP}(\mathbf{H}^L)$ is expressible in linear Datalog by [14]; in particular the problem is in NL . If, furthermore, the variety admits the lattice type, then $\neg\text{CSP}(\mathbf{H}^L)$ is not expressible in symmetric Datalog and is NL -hard by results in [25]. \square

By Lemma 1, the presence of a majority operation in \mathbb{H} implies that $\text{typ}(\mathcal{V}(\mathbb{H}))$ can contain only the Boolean and lattice types. The lattice type is dealt with in Theorem 2, so it remains to investigate graphs \mathbf{H} with $\mathcal{V}(\mathbb{H})$ admitting only the Boolean type. We will now define the class of graphs that plays a central role in our paper.

Definition 3. *The class \mathcal{F} consists of all graphs \mathbf{H} that contain none of the following as an induced subgraph:*

1. *the reflexive path of length 3 and the reflexive 4-cycle;*
2. *the irreflexive cycles of length 3, 5 and 6, and the irreflexive path of length 5;*
3. **B1, B2, B3, B4, B5 and B6** (see Figure 1.)

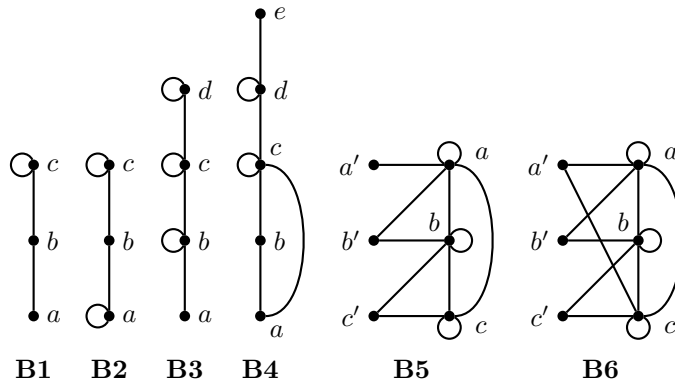


Figure 1: The forbidden graphs.

Notice that when only reflexive or only irreflexive graphs are of interest, then the only relevant forbidden subgraphs are those in Definition 3(1) or in Definition 3(2), respectively. Observe that all irreflexive graphs in \mathcal{F} are bipartite.

The next theorem is the main contribution of this paper.

Theorem 4. *Let \mathbf{H} be a graph. Then the following conditions are equivalent:*

1. \mathbf{H} admits conservative operations satisfying (Id1)–(Id3) for $n = 3$;
2. \mathbf{H} admits conservative operations satisfying (Id1)–(Id3) for some $n \geq 1$;
3. $\mathcal{V}(\mathbb{H})$ admits only the Boolean type;
4. $\mathbf{H} \in \mathcal{F}$;
5. $\neg\text{CSP}(\mathbf{H}^L)$ is definable in symmetric Datalog.

If the above holds then $\text{CSP}(\mathbf{H}^L)$ is in the complexity class L.

Proof. (1) trivially implies (2). If (2) holds then by Lemma 1 $\mathcal{V}(\mathbb{H})$ omits the unary, lattice, and semilattice types. By Lemma 16 in Section 5, \mathbf{H} admits a majority operation, so Lemma 1 implies that $\mathcal{V}(\mathbb{H})$ also omits the affine type; hence (3) holds. Implication (3) \Rightarrow (4) is the content of Lemma 17 in Section 5, and (5) implies (3) by a result of [25]. We give an inductive characterisation of the class \mathcal{F} in Theorem 14 in Section 4, and then use it to show that (4) implies both (1) and (5), in Subsection 5.1 and Section 6, respectively. Finally, definability in symmetric Datalog implies membership in \mathbf{L} by [16]. \square

For completeness' sake, we describe graphs whose list homomorphism problem is definable in first-order logic (equivalently, is in AC^0 , see [8].) By results in [25], any problem $\text{CSP}(\mathbf{T})$ is either first-order definable or \mathbf{L} -hard under FO reductions. Hence, it follows from Theorem 4 that, for a graph $\mathbf{H} \in \mathcal{F}$, the list homomorphism problem for \mathbf{H} is either first-order definable or \mathbf{L} -complete.

Theorem 5. *Let \mathbf{H} be a graph. Then $\text{CSP}(\mathbf{H}^L)$ is first-order definable if and only if \mathbf{H} has the following form: H is the disjoint union of two sets L and N such that (i) L is the set of loops of \mathbf{H} and induces a complete graph, (ii) N is the set of non-loops of \mathbf{H} and induces a graph with no edges, and (iii) $N = \{x_1, \dots, x_m\}$ can be ordered so that the neighbourhood of x_i is contained in the neighbourhood of x_{i+1} for all $1 \leq i \leq m - 1$.*

Remark 6. *Given a graph \mathbf{H} , it can be decided in polynomial time which of the different cases delineated in Theorems 2, 4, 5 the list homomorphism problem for \mathbf{H} satisfies. Indeed, it is known (see [18]) that \mathbf{H} is a bi-arc graph if and only if the complement of $\mathbf{H} \times \mathbf{K}_2$ is a circular arc graph which can be recognised in linear time [28]. Assume that \mathbf{H} is a bi-arc graph: the definition of \mathcal{F} (Definition 3) gives a polynomial time (in fact, even AC^0) algorithm to recognise them; and those graphs whose list homomorphism problem is first-order definable can be recognised in polynomial time by results of [24].*

The remaining sections are devoted to proving the lemmas used in the proof of the above theorems. Section 4 deals with the graph-theoretic proofs, Section 5 presents the proofs of the algebraic results, and Section 6 provides the symmetric Datalog expressibility proofs. Finally, Section 7 contains the proof of Theorem 5.

4 Combinatorial graph characterisations

In this section, we give an inductive characterisation of the class \mathcal{F} defined in the previous section. This characterisation is stated in Theorem 14. Before proving Theorem 14, we provide inductive characterisations for the reflexive and the irreflexive subclasses of \mathcal{F} in Lemmas 9 and 11, respectively. These lemmas will facilitate the proof of Theorem 14.

Let \mathcal{F}_{re} denote the reflexive graphs in \mathcal{F} (i.e. reflexive graphs that do not contain graphs in Definition 3(1) as induced subgraphs), and \mathcal{F}_{ir} the irreflexive graphs in \mathcal{F} (i.e. irreflexive graphs that do not contain graphs in Definition 3(2) as induced subgraphs).

We need the following two operations on graphs:

Definition 7. Let \mathbf{H}_1 and \mathbf{H}_2 be bipartite irreflexive graphs, with colour classes B_1, T_1 and B_2 and T_2 respectively, with T_1 and B_2 non-empty. We define the special sum $\mathbf{H}_1 \odot \mathbf{H}_2$ (which depends on the choice of the B_i and T_i)¹ as follows: it is the graph obtained from the disjoint union of \mathbf{H}_1 and \mathbf{H}_2 by adding all possible edges between the vertices in T_1 and B_2 . We say that an irreflexive graph \mathbf{H} is a special sum or expressed as a special sum if there exist two bipartite graphs and a choice of colour classes on each such that \mathbf{H} is isomorphic to the special sum of these two graphs.

Definition 8. Given two vertex-disjoint graphs \mathbf{H}_1 and \mathbf{H}_2 , the adjunction of \mathbf{H}_1 to \mathbf{H}_2 is the graph $\mathbf{H}_1 \circ \mathbf{H}_2$ obtained by taking the disjoint union of the two graphs, and adding every edge of the form (x, y) where x is a loop in \mathbf{H}_1 and y is a vertex of \mathbf{H}_2 .

We begin with the simple case of reflexive graphs.

4.1 The reflexive graphs in \mathcal{F}

Lemma 9. \mathcal{F}_{re} is the smallest class of reflexive graphs \mathcal{I}_{re} such that:

1. \mathcal{I}_{re} contains the one-element graph;
2. \mathcal{I}_{re} is closed under disjoint union;
3. if \mathbf{H}_1 is a single loop and $\mathbf{H}_2 \in \mathcal{I}_{\text{re}}$ then $\mathbf{H}_1 \circ \mathbf{H}_2 \in \mathcal{I}_{\text{re}}$.

Proof. It is easy to see that $\mathcal{I}_{\text{re}} \subseteq \mathcal{F}_{\text{re}}$. Suppose that $\mathcal{F}_{\text{re}} \not\subseteq \mathcal{I}_{\text{re}}$, and let \mathbf{H} be a graph of smallest size such that $\mathbf{H} \in \mathcal{F}_{\text{re}}$ and $\mathbf{H} \notin \mathcal{I}_{\text{re}}$, i.e. \mathbf{H} cannot be obtained from the one-element graph using the operations of disjoint union and adjunction of a loop. By minimality, \mathbf{H} is connected, contains no universal vertex (a vertex that is a neighbor of every other vertex including itself), it contains more than one vertex, and every of its proper induced connected subgraphs contains a universal vertex. Pick some edge (x, y) in \mathbf{H} ; since there is no universal vertex there exists some t not adjacent to y . Let \mathbf{G} be the subgraph induced by $H \setminus \{x\}$.

Assume first that \mathbf{G} is connected. Let u be a universal vertex of \mathbf{G} ; we have edges $(x, y), (y, u), (u, t)$. Since \mathbf{H} has no universal vertex then x is not adjacent to u . Thus $\{x, y, t, u\}$ is either a reflexive path of length 3 or a reflexive 4-cycle, a contradiction.

Suppose now that \mathbf{G} is not connected. Let C and D be distinct components of \mathbf{G} ; since x is not universal in \mathbf{H} there exists some z not adjacent to x , and without loss of generality suppose that $z \in C$. Since \mathbf{H} is connected there exists a path from z to some element in D , in particular we can find edges $(z', w), (w, x), (x, v)$, where $z', w \in C$ and w is a neighbor of x , z' is not adjacent to x , and $v \in D$. It is easy to verify that $\{z', w, x, v\}$ induces a reflexive path of length 3, a contradiction. \square

¹Notice that one can often divide a bipartite graph into parts in several ways, and even choose B_1 and/or T_2 to be empty.

Remark 10. Lemma 9 states that the reflexive graphs avoiding the path of length 3 and the 4-cycle are precisely those constructed from the one-element loop using disjoint union and adjunction of a universal vertex. These graphs can also be described by the following property: every connected induced subgraph of size at most 4 has a universal vertex. These graphs have been studied previously as those with NLCT width 1, which were proved to be exactly the trivially perfect graphs [20]. Our result provides an alternative proof of the equivalence of these conditions.

4.2 The irreflexive graphs in \mathcal{F}

The following result gives an inductive characterisation of the class of graphs \mathcal{F}_{ir} .

Lemma 11. \mathcal{F}_{ir} is the smallest class of irreflexive graphs \mathcal{I}_{ir} such that:

1. \mathcal{I}_{ir} contains the one-element graph;
2. \mathcal{I}_{ir} is closed under disjoint union;
3. \mathcal{I}_{ir} is closed under special sum.

Proof. We show that $\mathcal{I}_{\text{ir}} \subseteq \mathcal{F}_{\text{ir}}$. The class \mathcal{F}_{ir} obviously contains the one-element graph. In order to prove the inclusion, it is sufficient to show that if \mathbf{H}_1 and \mathbf{H}_2 are graphs that do not contain any cycles of length 3, 5 or 6, or a path of length 5 as an induced subgraph, then neither the disjoint union of \mathbf{H}_1 and \mathbf{H}_2 , nor the special sum of \mathbf{H}_1 and \mathbf{H}_2 contain any cycles of length 3, 5 or 6, or a path of length 5 as an induced subgraph. This is clearly the case for disjoint union, so now we concentrate on the special sum of \mathbf{H}_1 and \mathbf{H}_2 .

As it was observed after Definition 3, if an irreflexive graph does not contain cycles of length 3, 5 or a path of length 5, then it must be bipartite. It follows that $\mathbf{H}_1 \odot \mathbf{H}_2$ must be bipartite, so $\mathbf{H}_1 \odot \mathbf{H}_2$ contains no induced cycles of length 3 or 5. Assume then that \mathbf{C} is an induced subgraph of $\mathbf{H}_1 \odot \mathbf{H}_2$, where \mathbf{C} is a 6-cycle or a 5-path. We shall obtain a contradiction by showing that \mathbf{C} must be contained either in \mathbf{H}_1 or \mathbf{H}_2 . By assumption and definition of special sum, it is clear that, since \mathbf{C} is connected, it must contain at least one vertex in T_1 and at least one in B_2 ; on the other hand, since \mathbf{C} contains no induced 4-cycle, \mathbf{C} can have at most 2 vertices in T_1 and at most 1 in B_2 , without loss of generality. Suppose first that there is exactly one vertex of \mathbf{C} in T_1 . Since every vertex of \mathbf{C} has degree at most 2, it follows that no more than 1 vertex of \mathbf{C} can be in B_1 , and similarly no more than 1 vertex of \mathbf{C} can be in T_2 . Therefore \mathbf{C} cannot contain vertices both in T_1 and B_2 , so \mathbf{C} is either in \mathbf{H}_1 or \mathbf{H}_2 , a contradiction. On the other hand if \mathbf{C} has 2 vertices in T_1 , then \mathbf{C} has no vertex in T_2 and at most 2 in B_1 , so again, \mathbf{C} cannot contain vertices both in T_1 and B_2 , a contradiction. Hence, we conclude that $\mathcal{I}_{\text{ir}} \subseteq \mathcal{F}_{\text{ir}}$.

For the reverse inclusion, $\mathcal{F}_{\text{ir}} \subseteq \mathcal{I}_{\text{ir}}$, suppose for a contradiction that there exists a graph $\mathbf{H} \in \mathcal{F}_{\text{ir}}$ such that $\mathbf{H} \notin \mathcal{I}_{\text{ir}}$. Choose \mathbf{H} so that its set of vertices is of minimal size. Obviously \mathbf{H} is connected. We denote the usual graph distance

between vertices x and y by $d(x, y)$, i.e. the length of a shortest path in the graph between x and y . Let $N(x)$ denote the set of neighbours of x in \mathbf{H} , and let $N_2(x) = \{t \in T_1 : d(x, t) = 2\}$.

Claim 1. For every $x \in H$ there exists $y \in H$ such that $d(x, y) = 3$.

Proof. Otherwise, since \mathbf{H} is connected, we'd have some $x \in H$ with $d(x, y) \leq 2$ for all $y \in H$. Now let B_2 denote the set of all vertices adjacent to x , and let $T_2 = H \setminus (B_2 \cup \{x\})$. Furthermore let $B_1 = \emptyset$ and $T_1 = \{x\}$. Since \mathbf{H} is bipartite, (B_2, T_2) is a bipartition, and hence \mathbf{H} is expressed as a special sum, a contradiction. \square

Claim 2. There exists $x \in H$ such that the subgraph induced by $H \setminus \{x\}$ is connected.

Proof. Notice first that if for some x the subgraph \mathbf{G} induced by $H \setminus \{x\}$ is not connected, then it contains at most one connected component with 2 or more vertices. Indeed, by Claim 1 let $y \in H$ such that $d(x, y) = 3$; let y, w, z, x be an induced path of length 3 from y to x . Note that the connected component of y has size at least 2. Now choose a different connected component \mathbf{C} of \mathbf{G} that contains at least two vertices. Since \mathbf{H} is connected, \mathbf{C} clearly contains adjacent vertices u and v with u adjacent to x . But then the vertices y, w, z, x, u, v induce a path of length 5 in \mathbf{H} , a contradiction.

Now choose any vertex x in \mathbf{H} . If the subgraph induced by $H \setminus \{x\}$ is connected we are done; otherwise, one of its components must be trivial, i.e. \mathbf{H} has a vertex x' dangling from x . Then the subgraph induced by $H \setminus \{x'\}$ is connected. \square

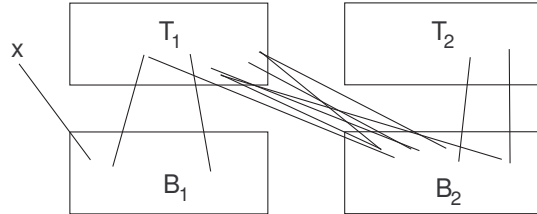


Figure 2: The graph \mathbf{H} .

So we may now suppose that \mathbf{H} has the following structure: there is some vertex x such that the subgraph \mathbf{G} induced by $H \setminus \{x\}$ is connected; by induction hypothesis, \mathbf{G} is a special sum, with subsets $B_i, T_i, (i = 1, 2)$ where T_1 and B_2 are non-empty. We suppose without loss of generality that x is adjacent to some vertex in $B_1 \cup B_2$ (see Figure 2.)

Case 1: T_2 is non-empty.

Claim 3. There exists an edge (u, v) with $u \in B_2$ and $v \in T_2$ such that u is not adjacent to x .

Proof. Suppose for a contradiction that this is not the case: then $B_2 = B_2^0 \cup B_2^1$ where B_2^0 consists of all elements of B_2 not adjacent to any vertex in T_2 , and since T_2 is non-empty, the set B_2^1 is non-empty, and contains by hypothesis only vertices adjacent to x . Then define a decomposition of H as follows: let $B'_1 = B_1 \cup B_2^0$, $T'_1 = T_1 \cup \{x\}$, $B'_2 = B_2^1$ and $T'_2 = T_2$. But then \mathbf{H} is a special sum, a contradiction. \square

Claim 4. The subgraph induced by $N(x) \cup N_2(x)$ is complete bipartite.

Proof. Otherwise, we may find elements $t \in N_2(x)$ and $z \in N(x)$ which are not adjacent. Let y be a vertex on a path of length 2 between x and t . Let u and v be the elements whose existence is guaranteed by the last claim: then it is easy to see that the sequence z, x, y, t, u, v is an induced path of length 5 in \mathbf{H} , a contradiction. \square

Consider the following decomposition of H : let $B'_1 = B_1 \setminus (N(x) \cap B_1)$, $T'_1 = N_2(x)$, $B'_2 = (N(x) \cap B_1) \cup B_2$ and $T'_2 = (T_1 \setminus N_2(x)) \cup \{x\} \cup T_2$. By Claim 4 this is a decomposition of \mathbf{H} as a special sum, unless there exists some edge (y, z) with $y \in B'_1$ and $z \in T'_2$, i.e. with $y \in B_1 \setminus N(x)$ and $z \in T_1 \setminus N_2(x)$. Suppose this occurs. Then we have the following:

Claim 5. (y, t) is an edge for every $t \in N_2(x)$.

Proof. If this is not the case, then choose some $t \in N_2(x)$ not adjacent to y ; let $n \in N(x)$ be adjacent to t . By Claim 3 we can find $u \in B_2$ not adjacent to x . Then the sequence y, z, u, t, n, x is an induced path of length 5 in \mathbf{H} , a contradiction. \square

It follows from Claim 5 that we can modify our last decomposition as follows: simply remove from B'_1 all the offending vertices such as y . More precisely, let Y be the set of all $y \in B'_1$ that have some neighbour $z \in T_1 \setminus N_2(x)$, and let $B''_1 = B'_1 \setminus Y$, $T''_1 = T'_1$, $B''_2 = Y \cup B'_2$ and $T''_2 = T'_2$. By Claim 5, this shows that \mathbf{H} is a special sum, a contradiction.

Case 2: T_2 is empty.

Notice that in this case we may assume that $N(x) \subseteq B_1$, by simply decomposing $H \setminus \{x\}$ if necessary as $B'_1 = B_1 \cup N(x)$, $B'_2 = B_2 \setminus N(x)$, and $T'_i = T_i$ for $i = 1, 2$. (Of course, if \mathbf{H} is not a special sum then there is at least one vertex in $B_2 \setminus N(x)$ for otherwise we could set $T'_1 = T_1 \cup \{x\}$.)

Claim 6. For every $y, z \in N(x)$ we have $N(y) \subseteq N(z)$ or $N(z) \subseteq N(y)$. Similarly, for every $u, v \in T_1$, either $N(u) \cap N(x) \subseteq N(v) \cap N(x)$ or $N(v) \cap N(x) \subseteq N(u) \cap N(x)$.

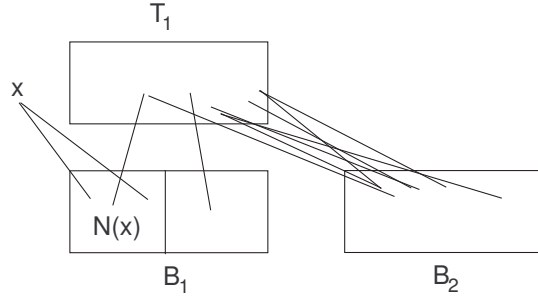


Figure 3: The graph \mathbf{H} with T_2 empty.

Proof. Suppose this is not the case: then we may find $y, z \in N(x)$ and $u \in N(y)$ and $v \in N(z)$ such that u is not adjacent to z and v is not adjacent to y . Let $b \in B_2$. Then clearly the subgraph of \mathbf{H} induced by $\{x, y, z, u, v, b\}$ is a 6-cycle, a contradiction. The argument for the second statement is identical. \square

By Claim 6, there exists an ordering of $N(x) = \{b_0, \dots, b_m\}$ such that $N(b_i) \subseteq N(b_j)$ if $i \leq j$, and an ordering of $T_1 = \{t_0, \dots, t_M\}$ such that $N(t_i) \cap N(x) \subseteq N(t_j) \cap N(x)$ if $i \leq j$. Since \mathbf{H} is connected, it is easy to see that b_m must be adjacent to t_M ; and by Claim 1, b_m cannot be adjacent to t_0 .

Claim 7. For every $t \in T_1$, either $N(t) \cap N(x) = N(x)$ or $N(t) \cap N(x) = \emptyset$.

Proof. Suppose this is not the case. Then there exists some $t \in T_1$ such that t is adjacent to b_m but not to b_0 . Then for any $b \in B_2$ the sequence b_0, x, b_m, t, b, t_0 is an induced path of length 5, a contradiction. \square

Let F denote the set of vertices $t \in T_1$ such that $N(t) \cap N(x) = N(x)$ and let E denote the set of vertices $t \in T_1$ such that $N(t) \cap N(x) = \emptyset$.

Claim 8. For every $y \in B_1 \setminus N(x)$, if y is adjacent to some vertex in E then it is adjacent to every vertex in F .

Proof. Otherwise we can find $t \in E$ and $t' \in F$ and $y \in B_1 \setminus N(x)$ such that (y, t) is an edge but (y, t') is not. Then for any $b \in B_2$ the sequence x, b_m, t', b, t, y is an induced path of length 5, a contradiction. \square

Let Y denote the set of vertices in $B_1 \setminus N(x)$ that are adjacent to some vertex in E . By the last claim, the subgraph induced by $(Y \cup B_2 \cup N(x)) \cup F$ is complete bipartite. Consider the following decomposition: let $B'_2 = Y \cup N(x) \cup B_2$, $B'_1 = B_1 \setminus B'_2$, $T'_1 = F$ and $T'_2 = E \cup \{x\}$. By the above argument, this shows that \mathbf{H} is in \mathcal{I}_{ir} , a contradiction. \square

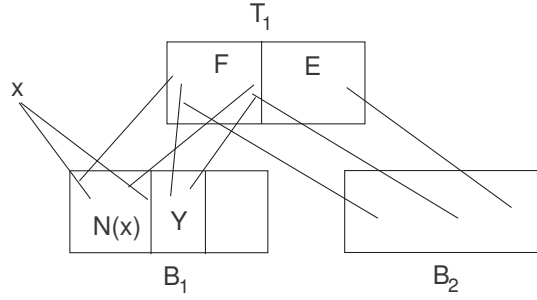


Figure 4: The graph \mathbf{H} .

4.3 The case of general graphs

In this section we shall prove Theorem 14 which provides an inductive characterisation of \mathcal{F} , our main family of graphs.

Call graphs in \mathcal{I}_{ir} (see Lemma 11) *basic irreflexive*.

Definition 12. A connected graph \mathbf{H} is basic if

1. \mathbf{H} is a single loop, or
2. \mathbf{H} is a basic irreflexive graph, or
3. \mathbf{H} is obtained from a basic irreflexive graph \mathbf{H}_{ir} with colour classes B and T by adding every edge (including loops) of the form (t, t') where $t, t' \in T$.

Definition 13. Let \mathcal{I} be the smallest class of graphs such that:

1. \mathcal{I} contains the basic graphs;
2. \mathcal{I} is closed under disjoint union;
3. if \mathbf{H}_1 is a basic graph and $\mathbf{H}_2 \in \mathcal{I}$ then $\mathbf{H}_1 \circ \mathbf{H}_2 \in \mathcal{I}$.

Theorem 14. The graph classes \mathcal{F} and \mathcal{I} coincide.

Proof. To establish the inclusion $\mathcal{I} \subseteq \mathcal{F}$, we start by showing that every basic graph is in \mathcal{F} , i.e. that a basic graph does not contain any of the forbidden graphs. If \mathbf{H} is a single loop or a basic irreflexive graph, then this is immediate. Otherwise \mathbf{H} is obtained from a basic irreflexive graph \mathbf{H}_{ir} with colour classes B and T by adding every edge of the form (t_1, t_2) where $t_i \in T$. In particular, the loops form a clique and no edge connects two non-loops; it is clear in that case that \mathbf{H} contains none of **B1**, **B2**, **B3**, **B4**. On the other hand if \mathbf{H} contains **B5** or **B6**, then \mathbf{H}_1 contains the path of length 5 or the 6-cycle, contradicting the fact that \mathbf{H}_{ir} is basic.

Next we show that \mathcal{F} is closed under disjoint union and adjunction of basic graphs. It is obvious that the disjoint union of graphs that avoid the forbidden graphs will also avoid these. So suppose that an adjunction $\mathbf{H}_1 \circ \mathbf{H}_2$, where \mathbf{H}_1 is a basic graph, contains an induced forbidden graph \mathbf{B} whose vertices are neither all in H_1 nor H_2 ; without loss of generality H_1 contains at least one loop, its loops form a clique and none of its edges connects two non-loops. It is then easy to verify that \mathbf{B} contains both loops and non-loops. Because the other cases are similar, we prove only that \mathbf{B} is not **B3**. Observe that every loop in \mathbf{H}_1 is adjacent to every loop in $\mathbf{H}_1 \circ \mathbf{H}_2$. So b , c , and d (see Figure 1) must be in H_2 . But if a is in H_1 , then it cannot be adjacent to a loop in H_2 , so a is also in \mathbf{H}_2 , a contradiction.

Now we must show that $\mathcal{F} \subseteq \mathcal{I}$, i.e. every graph in \mathcal{F} can be obtained from the basic graphs by disjoint union and adjunction of basic graphs. Suppose this is not the case. If \mathbf{H} is a counterexample of minimum size, then obviously it is connected, and it contains at least one loop for otherwise it is a basic irreflexive graph. By Lemma 9, \mathbf{H} also contains at least one non-loop.

Let $\mathbf{R}(\mathbf{H})$ denote the subgraph of \mathbf{H} induced by its set $R(H)$ of loops, and let $\mathbf{J}(\mathbf{H})$ denote the subgraph induced by $J(H)$, the set of non-loops of \mathbf{H} . Since \mathbf{H} is connected and neither **B1** nor **B2** is an induced subgraph of \mathbf{H} , the graph $\mathbf{R}(\mathbf{H})$ is also connected, and furthermore every vertex in $J(H)$ is adjacent to some vertex in $R(H)$. By Lemma 9, we know that $\mathbf{R}(\mathbf{H})$ contains at least one universal vertex: let U denote the (non-empty) set of universal vertices of $\mathbf{R}(\mathbf{H})$. Let J denote the set of all $a \in J(H)$ such that $N(a) \cap R(H) \subseteq U$. Let us show that $J \neq \emptyset$. For every $u \in U$, there is $w \in J(H)$ not adjacent to u because otherwise \mathbf{H} is obtained by adjoining u to the rest of \mathbf{H} , a contradiction with the choice of \mathbf{H} . If this w has a neighbour $r \in R(H) \setminus U$ then there is some $s \in R(H) \setminus U$ not adjacent to r , and the graph induced by $\{w, u, s, r\}$ contains **B2** or **B3**, a contradiction. Hence, $w \in J$. Let \mathbf{S} denote the subgraph of \mathbf{H} induced by $U \cup J$. The graph \mathbf{S} is connected. We claim that the following properties also hold:

1. if a and b are adjacent non-loops, then $N(a) \cap U = N(b) \cap U$;
2. if a is in a connected component of the subgraph of \mathbf{S} induced by J with more than one vertex, then for any other $b \in J$, one of $N(a) \cap U, N(b) \cap U$ contains the other.

The first statement holds because **B1** is forbidden, and the second follows from the first because **B4** is also forbidden. Let J_1, \dots, J_k denote the different connected components of J in \mathbf{S} . By (1) we may let $N(J_i)$ denote the set of common neighbours of members of J_i in U . By (2), we can re-order the J_i 's so that for some $1 \leq m \leq k$ we have $N(J_i) \subseteq N(J_j)$ for all $i \leq m$ and all $j > m$, and, in addition, we have $m = 1$ or $|J_i| = 1$ for all $1 \leq i \leq m$. Let \mathbf{B} denote the subgraph of \mathbf{S} induced by $B = \bigcup_{i=1}^m (J_i \cup N(J_i))$, and let \mathbf{C} be the subgraph of \mathbf{H} induced by $H \setminus B$. We claim that $\mathbf{H} = \mathbf{B} \circ \mathbf{C}$. For this, it suffices to show that every element in $\bigcup_{i=1}^m N(J_i)$ is adjacent to every non-loop $c \in C$. By construction this holds if $c \in J \cap C$. Now suppose this does not hold: then some

$x \in J(H) \setminus J$ is not adjacent to some $y \in N(J_i)$ for some $i \leq m$. Since $x \notin J$ we may find some $z \in R(H) \setminus U$ adjacent to x ; it is of course also adjacent to y . Since $z \notin U$ there exists some $z' \in R(H) \setminus U$ that is not adjacent to z , but it is of course adjacent to y . If x is adjacent to z' , then $\{x, z, z'\}$ induces a subgraph isomorphic to **B2**, a contradiction. Otherwise, $\{x, z, y, z'\}$ induces a subgraph isomorphic to **B3**, also a contradiction.

If every J_i with $i \leq m$ contains a single element, notice that **B** is a basic graph: indeed, removing all edges between its loops yields a bipartite irreflexive graph which contains neither the path of length 5 nor the 6-cycle, since **B** contains neither **B5** nor **B6**. Since this contradicts our hypothesis on **H**, we conclude that $m = 1$. But this means that $N(J_1)$ is a set of universal vertices in **H**. Let u be such a vertex and let D denote its complement in **H**: clearly **H** is obtained as the adjunction of the single loop u to D , contradicting our hypothesis. This concludes the proof. \square

5 Algebraic results

We need the following well-known auxiliary result. Note that the assumption of conservativity of the algebra $\mathbb{A}_{\mathbf{T}}$ in it is not essential. Note also that the assumptions of this lemma effectively say that $\text{CSP}(\mathbf{T})$ can simulate the graph k -colouring problem (with $k = |U|$) or the directed st -connectivity problem.

Lemma 15. *Let \mathbf{S}, \mathbf{T} be structures such that the algebra $\mathbb{A}_{\mathbf{T}}$ is conservative, let $s_1, s_2 \in S$, and let $R = \{(f(s_1), f(s_2)) \mid f : \mathbf{S} \rightarrow \mathbf{T}\}$.*

1. *If $R = \{(x, y) \in U^2 \mid x \neq y\}$ for some subset $U \subseteq T$ with $|U| \geq 3$ then $\mathcal{V}(\mathbb{A}_{\mathbf{T}})$ admits the unary type.*
2. *If $R = \{(t, t), (t, t'), (t', t')\}$ for some distinct $t, t' \in T$ then $\mathcal{V}(\mathbb{A}_{\mathbf{T}})$ admits at least one of the following types: unary, lattice, semilattice.*

Proof. The assumption of this lemma implies that $\mathbb{A}_{\mathbf{T}}$ has a subalgebra (with universe U and $\{t, t'\}$, respectively) such that all operations of the subalgebra preserve the relation R . It is well known (see, e.g., [21]) that all conservative operations preserving the disequality relation on U are projections which proves the first statement, while it is easy to check that the order relation on a 2-element set (such as the relation R from the second statement) cannot admit operations satisfying identities (Id1)–(Id3), so one can use Lemma 1 to prove the second statement. \square

The following lemma connects the characterisation of bi-arc graphs given in [5] with a type-omitting condition.

Lemma 16. *Let \mathbf{H} be a graph. Then the following conditions are equivalent:*

1. *the variety $\mathcal{V}(\mathbb{H})$ omits the unary type;*
2. *the graph \mathbf{H} admits a conservative majority operation;*

3. the graph \mathbf{H} is a bi-arc graph.

Proof. The equivalence of (2) and (3) is from [5], and (2) implies (1) by Lemma 1, so the rest of this proof shows that (1) implies (3). We shall use the following construction from [18]. Given a graph \mathbf{H} , let \mathbf{K} denote the irreflexive bipartite graph obtained from \mathbf{H} as follows: its vertices consist of two copies of the vertex set of \mathbf{H} , say $H' = \{x' : x \in H\}$ and $H'' = \{x'' : x \in H\}$, with edges (x', y'') iff (x, y) is an edge of \mathbf{H} . In other words, $\mathbf{K} = \mathbf{H} \times \mathbf{K}_2$ where \mathbf{K}_2 is the irreflexive edge. Let \mathbb{K} denote the algebra associated with \mathbf{K}^L .

By putting together Proposition 3.1 of [18] and Corollary 4.6 of [17], one immediately obtains that \mathbf{H} is a bi-arc graph if and only if \mathbf{K} is chordal bipartite and contains no special edge-asteroids. We need not know what these two conditions on \mathbf{K} mean - it is shown in (proofs of) Theorems 3.1 and 3.2 of [17] that if \mathbf{K} fails to satisfy either of them then $\mathbf{T} = \mathbf{K}^L$ satisfies the conditions of Lemma 15(1) for suitable \mathbf{S} , s_1, s_2 , and so $\mathcal{V}(\mathbb{K})$ admits the unary type. Hence, it only remains to show that the variety $\mathcal{V}(\mathbb{K})$ omits the unary type whenever $\mathcal{V}(\mathbb{H})$ does so.

It is well known (see, e.g., Corollary 3.3 in [32]), that the unary type is present in the variety generated by a conservative algebra \mathbb{A} if and only if there exist elements a, b in the algebra such that each operation of \mathbb{A} is a projection when restricted to $\{a, b\}$. So we may assume now that, for every 2-element subset $\{a, b\}$ of H , there is an operation of \mathbb{H} that is not a projection when restricted to $\{a, b\}$, and we need to show the same for \mathbb{K} .

For each operation (say k -ary) operation f of \mathbb{H} , introduce an $(2k - 1)$ -ary operation g_f on K , as follows. Let $x = (x_1^{e_1}, \dots, x_{2k-1}^{e_{2k-1}})$ be an element of K^{2k-1} , where $x_1, \dots, x_{2k-1} \in H$ and $e_1, \dots, e_{2k-1} \in \{', ''\}$. Then obviously exactly one of ' or '' appears at least k times; let ϵ denote this symbol; let i_1, \dots, i_k denote the first k positions where it appears in the tuple x ; then define

$$g_f(x) = f(x_{i_1}, \dots, x_{i_k})^\epsilon.$$

It is clear that this is a well-defined operation on K , and it is easy to see that it preserves edges of \mathbf{K} ; since f is conservative, so is g_f . Hence, g_f is an operation of \mathbb{K} .

Let $\{x^u, y^v\}$ be a 2-element subset of K . Suppose first that x^u and y^v belong to different colour classes of K : then the restriction of g_f to this subset satisfies the property

$$g(x^u, \dots, x^u, y^v, x^u, \dots, x^u) = f(x, \dots, x)^u = x^u$$

and similarly for $g(y^v, \dots, y^v, x^u, y^v, \dots, y^v)$. On the other hand if x^u and y^v are in the same colour class, then the restriction of g_f to $\{x^u, y^v\}$ coincides with that of f (with $k - 1$ additional fictitious variables). It follows that in either case, the restriction of g_f is not a projection whenever the restriction of f is not. \square

The following lemma establishes the implication (3) \Rightarrow (4) in Theorem 4.

Lemma 17. *If $\mathbf{H} \notin \mathcal{F}$ then $\mathcal{V}(\mathbb{H})$ admits a non-Boolean type.*

Proof. By Theorem 9.15 of [22], $\mathcal{V}(\mathbb{H})$ admits only the Boolean type if and only if \mathbf{H} admits a sequence of conservative operations satisfying certain identities in the spirit of (Id1)–(Id3). (Note that Theorem 9.15 of [22] applies to the so-called locally finite varieties, but every variety generated by a single finite algebra, such as $\mathcal{V}(\mathbb{H})$, has this property [29]). By conservativity, such operations can be restricted to any subset of H while satisfying the same identities, so the property of having only the Boolean type in the variety generated by their conservative algebra is inherited by induced subgraphs of \mathbf{H} . It follows that it is enough to prove this lemma for the forbidden graphs from Definition 3.

For the irreflexive odd cycles, the lemma follows immediately from the main results of [4, 27]. The proof of Theorem 3.1 of [17] shows that the conditions of Lemma 15(1) are satisfied by (some \mathbf{S}, s_1, s_2 and) $\mathbf{T} = \mathbf{F}^L$ where \mathbf{F} is the irreflexive 6-cycle. One can easily check that the reflexive 4-cycle is not a bi-arc graph, so we can apply Lemma 16 in this case.

For the remaining forbidden graphs \mathbf{F} from Definition 3, we use Lemma 15(2) with $\mathbf{T} = \mathbf{F}^L$. In each case, the binary relation of the structure \mathbf{S} will be a short undirected path, and s_1, s_2 will be the endpoints of the path. We will represent such a structure \mathbf{S} by a sequence of subsets of F (indicating lists assigned to vertices of the path). It can be easily checked that, in each case, the relation R defined as in Lemma 15(2) is of the required form.

If \mathbf{F} is the reflexive path of length 3, say $a-b-c-d$, then $\mathbf{S} = ac-bc-ad-ac$. If \mathbf{F} is the irreflexive path of length 5, say $a-b-c-d-e-f$, then $\mathbf{S} = ae-bd-ce-bf-ae$. For graphs **B1** – **B6**, we use notation from Figure 1. For **B1**, $\mathbf{S} = bc-bc-ab-ab-bc$. For **B2**, $\mathbf{S} = bc-ac-ab-bc$. For **B3**, $\mathbf{S} = bc-ad-bd-bc$. For **B4**, $\mathbf{S} = ae-bd-cd-ae$. Finally, for both **B5** and **B6**, $\mathbf{S} = ac-b'c'-ab-a'c'-ac$. \square

5.1 Implication (4) \Rightarrow (1) in Theorem 4

We prove this implication in two steps: first for irreflexive graphs and then in general.

Recall the definition of basic irreflexive graphs from Lemma 11 and Definition 7.

Lemma 18. *If \mathbf{H} is a basic irreflexive graph then \mathbf{H} admits conservative operations satisfying (Id1)–(Id3) for $n = 3$.*

Proof. We shall show by induction on the size of \mathbf{H} that there exist conservative operations f_1, f_2, f_3 preserving the graph \mathbf{H} , obeying the identities (Id1)–(Id3) and furthermore that satisfy the following condition (D):

For every $x, y, z, n, m \in H$ such that n is adjacent to x and m is adjacent to z , $f_1(x, y, z)$ is adjacent to n and $f_3(x, y, z)$ is adjacent to m .

The result is trivial for a one-element graph. If \mathbf{H} is not connected, then \mathbf{H} is the disjoint union of proper subgraphs \mathbf{H}_1 and \mathbf{H}_2 . Let f_1, f_2, f_3 and g_1, g_2, g_3 be the desired operations on \mathbf{H}_1 and \mathbf{H}_2 respectively; we define operations h_1, h_2, h_3 on \mathbf{H} as follows:

For every $1 \leq s \leq 3$, let $h_s(x, y, z) = f_s(x, y, z)$ if $(x, y, z) \in H_1^3$ and let $h_s(x, y, z) = g_s(x, y, z)$ if $(x, y, z) \in H_2^3$; if $(x, y, z) \in H_i \times H_j \times H_k$ with i, j, k not all equal, then let $h_1(x, y, z) = x$ and $h_3(x, y, z) = z$, and finally let $h_2(x, y, z) = z$ if $(i, j, k) \in \{(1, 1, 2), (2, 2, 1)\}$ and let $h_2(x, y, z) = x$ otherwise.

It is immediate that identities (Id1) and (Id3) are satisfied and that each h_s is a conservative homomorphism. For (Id2): we may assume that $x \neq y$; if x and y are in the same H_i then (Id2) follows from the fact that the f_i and g_i satisfy it; otherwise we have that $h_1(x, x, y) = x = h_2(x, y, y)$ and $h_2(x, x, y) = y = h_3(x, y, y)$. It is easy to see that condition (D) is satisfied by h_1 and h_3 .

Now suppose that the basic graph \mathbf{H} is connected, and hence is the special sum of two smaller graphs. For the moment, it will be convenient to denote the colour classes of \mathbf{H} by C_1 and C_2 ; our first task is to show it suffices to define our operations on $C_1^3 \cup C_2^3$. Indeed, suppose that we have functions $F'_1, F'_2, F'_3 : C_1^3 \cup C_2^3 \rightarrow H$ that satisfy all the required identities, are edge-preserving and conservative. Then we may extend these to full operations $F_1, F_2, F_3 : H^3 \rightarrow H$ as follows: let

$$F_1(x, y, z) = \begin{cases} F'_1(x, y, z), & \text{if } (x, y, z) \in C_1^3 \cup C_2^3; \\ x, & \text{otherwise.} \end{cases}$$

$$F_3(x, y, z) = \begin{cases} F'_3(x, y, z), & \text{if } (x, y, z) \in C_1^3 \cup C_2^3; \\ z, & \text{otherwise.} \end{cases}$$

$$F_2(x, y, z) = \begin{cases} F'_2(x, y, z), & \text{if } (x, y, z) \in C_1^3 \cup C_2^3; \\ z, & \text{if } (x, y, z) \in C_i \times C_i \times C_j \text{ for some } i \neq j; \\ x, & \text{otherwise.} \end{cases}$$

Notice that distinct sets $C_i \times C_j \times C_k$ and $C_{i'} \times C_{j'} \times C_{k'}$ are in different connected components of \mathbf{H}^3 , unless $i \neq i', j \neq j'$ and $k \neq k'$; it follows immediately that the F_i are edge-preserving; they are also clearly conservative. It is a simple matter to verify that all the required identities are satisfied. Hence, from now on, we assume without mention that in all triples (x, y, z) considered all the entries come from the same colour class of the graph under consideration.

So let \mathbf{H} be the special sum of two smaller graphs \mathbf{H}_i with colour classes B_i and T_i , $i = 1, 2$; by induction hypothesis \mathbf{H}_1 admits the required operations f_1, f_2, f_3 and \mathbf{H}_2 admits operations g_1, g_2, g_3 satisfying the necessary conditions. We define operations F_1, F_2, F_3 on \mathbf{H} as follows. For convenience, let $S = T_1 \cup B_2$, $B = B_1 \cup B_2$, $T = T_1 \cup T_2$. Notice that by definition of special sum S induces a complete bipartite graph in \mathbf{H} .

$$F_1(x, y, z) = \begin{cases} f_1(x, y, z), & \text{if } x, y, z \in H_1, \text{ else} \\ g_1(x, y, z), & \text{if } x, y, z \in H_2, \text{ else} \\ x, & \text{if } y = z \text{ or } x \in S, \text{ else} \\ u, & \text{where } u \text{ is the leftmost of } y, z \text{ element of } S. \end{cases}$$

$$F_3(x, y, z) = \begin{cases} f_3(x, y, z), & \text{if } x, y, z \in H_1, \text{ else} \\ g_3(x, y, z), & \text{if } x, y, z \in H_2, \text{ else} \\ z, & \text{if } x = y \text{ or } z \in S, \text{ else} \\ v, & \text{where } v \text{ is the leftmost of } x, y \text{ element of } S. \end{cases}$$

$$F_2(x, y, z) = \begin{cases} F_1(x, x, z), & \text{if } y = z, \text{ else} \\ F_3(x, z, z), & \text{if } x = y, \text{ else} \\ f_2(x, y, z), & \text{if } x, y, z \in H_1, \text{ else} \\ g_2(x, y, z), & \text{if } x, y, z \in H_2, \text{ else} \\ w, & \text{where } w \text{ is the leftmost of } x, y, z \text{ element of } S. \end{cases}$$

Obviously all three operations are conservative, and by definition they obey all the required identities. Now we verify that F_1 satisfies condition (D): let (x, n) be an edge of \mathbf{H} : we show that $F_1(x, y, z)$ is adjacent to n . If $x, y, z \in H_i$ for some $i = 1, 2$ then this follows by induction hypothesis, and it is clearly true if $F_1(x, y, z) = x$. Otherwise, $F_1(x, y, z) = u$ for some $u \in S$; if $x, y, z \in B$ then $x \in B_1$ so $n \in T_1$ is adjacent to u . Otherwise $x, y, z \in T$, so $x \in T_2$ hence $n \in B_2$ is adjacent to u . The proof that F_3 satisfies (D) is identical. It remains to show that each F_i is edge-preserving. Let (x, y, z) be adjacent to (x', y', z') and suppose without loss of generality that $x, y, z \in B$ and $x', y', z' \in T$. We start with F_1 . If $x, y, z \in B_1$ then $x', y', z' \in T_1$ and hence F_1 coincides with f_1 on both tuples and we are done by induction hypothesis. If $F_1(x, y, z) = x$ then by (D) we have $F_1(x', y', z')$ adjacent to x . Otherwise, we have that $x \in B_1$ (and thus $x' \in T_1$) and $F_1(x, y, z) = u \in B_2$; in any case $F_1(x', y', z') \in T_1$ so it is adjacent to u . The argument for F_3 is identical. Now we consider F_2 . Notice that by induction hypothesis and definition of the F_i , we have that F_2 coincides with f_2 (or with g_2) on tuples whose coordinates all lie in H_1 (respectively H_2). If $x, y, z \in B_1$, then certainly $x', y', z' \in T_1$, and then the result follows by induction hypothesis and the last remark. Now we require the following claim:

Claim. Suppose that a, b, c do not all lie in the same H_i . If $b = c$ or $a = b$ then $F_2(a, b, c) \in S$.

Proof. Suppose that $b = c$, so that $F_2(a, b, c) = F_1(a, a, c)$. By hypothesis, a and c do not lie in the same H_i , and in particular they are distinct, hence by definition of F_1 we have that $F_1(a, a, c) = a$ if $a \in S$ or $F_1(a, a, c) = u$ for some $u \in S$. The proof for the case $a = b$ is identical. \square

Now we can finish the proof. Suppose first that x, y, z are not all in the same H_i ; by the claim $F_2(x, y, z) \in S$. If x', y', z' are not all in the same H_i then $F_2(x', y', z') \in S$ also and we are done. Otherwise, x', y', z' all lie in T_1 (since one of them is a neighbour of an element of B_1) and hence $F_2(x', y', z') = f_2(x', y', z') \in S$ and we are done. Now suppose that x, y, z are all in B_2 (we dealt with the case B_1 earlier.) Then $F_2(x, y, z) = g_2(x, y, z) \in S$, so if x', y', z' are not all in the same H_i we are done by the claim again. Otherwise either $x', y', z' \in T_1$ so $F_2(x', y', z') = f_2(x', y', z') \in S$, or else $x', y', z' \in T_2$: then $F_2(x', y', z') = g_2(x', y', z')$ and we are done by induction hypothesis. \square

Lemma 19. *If $\mathbf{H} \in \mathcal{F}$ then \mathbf{H} admits conservative operations satisfying (Id1)–(Id3) for $n = 3$.*

Proof. We invoke the characterisation of \mathcal{F} from Theorem 14. We will prove that \mathbf{H} has the required polymorphisms when \mathbf{H} is a basic graph, and show that this property is preserved under disjoint union and adjunction of basic graphs.

Let \mathbf{H} be a basic graph. The result is trivial if \mathbf{H} is a single loop, and if \mathbf{H} is a basic irreflexive graph then we invoke Lemma 18. So now assume that \mathbf{H} is obtained from some basic irreflexive graph \mathbf{H}_1 with colour classes B and T by adding all edges (t, t') with $t, t' \in T$. By Lemma 18 there exist operations f_1, f_2, f_3 on \mathbf{H}_1 satisfying the required identities; furthermore recall that we can assume that the f_i satisfy condition (D):

For every $x, y, z, n, m \in H$ such that n is adjacent to x and m is adjacent to z , $f_1(x, y, z)$ is adjacent to n and $f_3(x, y, z)$ is adjacent to m .

For convenience of notation, define, on triples (x_1, x_2, x_3) such that $\{x_1, x_2, x_3\}$ intersects the set T , two ternary operations μ^L and μ^R by $\mu^L(x_1, x_2, x_3) = x_j$ where $j = \min\{i : x_i \in T\}$ and $\mu^R(x_1, x_2, x_3) = x_k$ where $k = \max\{i : x_i \in T\}$. Notice that both of these operations trivially preserving the edges of \mathbf{H} . We define operations F_1, F_2 and F_3 on \mathbf{H} as follows:

$$F_1(x_1, x_2, x_3) = \begin{cases} x_1, & \text{if } x_2 = x_3, \text{ else} \\ f_1(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \text{ intersects only one of } B \text{ and } T, \\ \mu^L(x_1, x_2, x_3), & \text{otherwise.} \end{cases}$$

$$F_3(x_1, x_2, x_3) = \begin{cases} x_3, & \text{if } x_1 = x_2, \text{ else} \\ f_3(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \text{ intersects only one of } B \text{ and } T, \\ \mu^R(x_1, x_2, x_3), & \text{otherwise.} \end{cases}$$

$$F_2(x_1, x_2, x_3) = \begin{cases} f_2(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \text{ intersects only one of } B \text{ and } T, \\ \mu^L(x_1, x_2, x_3), & \text{otherwise.} \end{cases}$$

It is clear that all three operations are conservative, and that identities (Id1) and (Id3) are satisfied. To prove (Id2), suppose without loss of generality that $x \neq y$: if $\{x, y\}$ intersects only one of B and T then $F_1(x, x, y) = f_1(x, x, y) = f_2(x, y, y) = F_2(x, y, y)$; on the other hand if $\{x, y\}$ intersects both B and T then $F_1(x, x, y) = F_2(x, y, y)$ is the unique element in $\{x, y\}$ that belongs to T . The proof that $F_2(x, x, y) = F_3(x, y, y)$ is similar.

Next we prove that property (D) holds for F_1 (the proof for F_3 is identical.) Let n be a neighbour of x_1 . If $F_1(x_1, x_2, x_3) = x_1$ the result is trivial, and if $F_1(x_1, x_2, x_3) = f_1(x_1, x_2, x_3)$ then we are done because f_1 satisfies (D). If $F_1(x_1, x_2, x_3) = \mu^L(x_1, x_2, x_3)$, there are two cases: if $x_1 \in T$ then $F_1(x_1, x_2, x_3) = x_1$, otherwise $x_1 \in B$ forces $n \in T$ so n is necessarily adjacent to $\mu^L(x_1, x_2, x_3) \in T$.

Finally we show that F_1 is edge-preserving (the proof for F_2 and F_3 is identical.) Let (x_1, x_2, x_3) and (y_1, y_2, y_3) be adjacent. If one of them has elements from both B and T , then so does the other, in which case we are done. If $x_2 = x_3$ then $F_1(x_1, x_2, x_3) = x_1$ and $F_1(y_1, y_2, y_3)$ is adjacent to x_1 by property (D). Otherwise, we may assume without loss of generality that $F_1(x_1, x_2, x_3) = f_1(x_1, x_2, x_3)$ and the x_i are all in B or all in T . If $F_1(y_1, y_2, y_3) = f_1(y_1, y_2, y_3)$ we are done since f_1 is edge-preserving; hence we may assume that $\{y_1, y_2, y_3\}$ intersects both B and T and hence every x_i is in T . In that case $F_1(x_1, x_2, x_3)$ is in T and it is adjacent to $F_1(y_1, y_2, y_3) = \mu_L(y_1, y_2, y_3) \in T$. This completes the proof for all basic graphs.

The proof for disjoint union is identical to the one in the irreflexive case (Lemma 18).

Finally we show that the property of admitting conservative operations satisfying (Id1)–(Id3) for $n = 3$ is preserved under adjunction of a basic graph. Let \mathbf{H}_1 be a basic graph, where L_1 and N_1 denote the set of loops and non-loops of \mathbf{H}_1 respectively, and let \mathbf{H}_2 satisfy our induction hypothesis, and let L_2 and N_2 denote the set of loops and non-loops of \mathbf{H}_2 respectively. We may assume that L_1 is non-empty, and hence it is a clique. Let g_1, g_2, g_3 be operations on \mathbf{H}_2 that satisfy all required identities and property (D). By our earlier analysis, we know there exist operations f_1, f_2, f_3 on the basic graph \mathbf{H}_1 that satisfy all required identities and property (D), and moreover satisfy the following condition (E):

*If $\{x, y, z\}$ intersects L_1 and $y \neq z$ then
 $f_1(x, y, z)$ and $f_3(y, z, x)$ belong to L_1 .*

For convenience of notation, define two ternary operations λ^L, λ^R on triples (x_1, x_2, x_3) such that $\{x_1, x_2, x_3\}$ intersects the set L_1 by $\lambda^L(x_1, x_2, x_3) = x_j$ where $j = \min\{i : x_i \in L_1\}$ and $\lambda^R(x_1, x_2, x_3) = x_k$ where $k = \max\{i : x_i \in L_1\}$. Define two ternary operations ν^L and ν^R on triples (x_1, x_2, x_3) such that $\{x_1, x_2, x_3\}$ intersects the set H_2 by $\nu^L(x_1, x_2, x_3) = x_j$ where $j = \min\{i : x_i \in H_2\}$ and $\nu^R(x_1, x_2, x_3) = x_k$ where $k = \max\{i : x_i \in H_2\}$. Notice that λ^L and λ^R are trivially edge-preserving, and so are ν^L and ν^R if we restrict them to triples (x_1, x_2, x_3) such that $\{x_1, x_2, x_3\} \subseteq N_1 \cup H_2$.

We define operations F_1, F_2 and F_3 on \mathbf{H} as follows:

$$F_1(x_1, x_2, x_3) = \begin{cases} x_1, & \text{if } x_2 = x_3, \text{ else} \\ f_1(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \subseteq H_1, \text{ else} \\ g_1(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \subseteq H_2, \text{ else} \\ \lambda^L(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \text{ intersects both } L_1 \text{ and } H_2, \\ \nu^L(x_1, x_2, x_3), & \text{otherwise.} \end{cases}$$

$$F_3(x_1, x_2, x_3) = \begin{cases} x_3, & \text{if } x_1 = x_2, \text{ else} \\ f_3(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \subseteq H_1, \text{ else} \\ g_3(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \subseteq H_2, \text{ else} \\ \lambda^R(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \text{ intersects both } L_1 \text{ and } H_2, \\ \nu^R(x_1, x_2, x_3), & \text{otherwise.} \end{cases}$$

$$F_2(x_1, x_2, x_3) = \begin{cases} f_2(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \subseteq H_1, \text{ else} \\ g_2(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \subseteq H_2, \text{ else} \\ \lambda^L(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \text{ intersects both } L_1 \text{ and } H_2, \\ \nu^L(x_1, x_2, x_3), & \text{otherwise.} \end{cases}$$

It is clear that each F_i is conservative and that identities (Id1) and (Id3) are satisfied. To prove (Id2), suppose without loss of generality that $x \neq y$: if $\{x, y\}$ is contained in H_1 or contained in H_2 then the result follows from the fact that the f_i and g_i satisfy (Id2); if $\{x, y\}$ intersects both L_1 and H_2 then $F_1(x, x, y) = F_2(x, y, y)$ is the unique element in $\{x, y\}$ that belongs to L_1 ; if $\{x, y\}$ intersects both N_1 and H_2 then $F_1(x, x, y) = F_2(x, y, y)$ is the unique element in $\{x, y\}$ that belongs to H_2 . The proof that $F_2(x, x, y) = F_3(x, y, y)$ is similar.

Next we prove that property (D) holds for F_1 (the proof for F_3 is identical.) Let n be a neighbour of x_1 . If $F_1(x_1, x_2, x_3) = x_1$ the result is trivial, and if $\{x_1, x_2, x_3\}$ is contained in H_1 or contained in H_2 then the result follows from the fact that both f_i and g_i satisfy (D). Suppose now that $\{x_1, x_2, x_3\}$ intersects both L_1 and H_2 . Then $F_1(x_1, x_2, x_3) \in L_1$; in particular if $n \in H_2 \cup L_1$ we are done. If on the other hand $n \in N_1$ then $x_1 \in L_1$ so $F_1(x_1, x_2, x_3) = \lambda^L(x_1, x_2, x_3) = x_1$. If $F_1(x, y, z) = \nu^L(x, y, z) \neq x$ then $x \in N_1$, $n \in L_1$, and $\nu^L(x, y, z) \in H_2$, so we are done.

Finally we show that F_1 is edge-preserving (the proof for F_2 and F_3 is identical.) Let (x_1, x_2, x_3) and (y_1, y_2, y_3) be adjacent. We analyse the different cases. Without loss of generality we may assume throughout that $y_2 \neq y_3$.

(1) Suppose first that $x_2 = x_3$ so that $F_1(x_1, x_2, x_3) = x_1$. (a) If $\{y_1, y_2, y_3\} \subseteq H_1$, then $F_1(y_1, y_2, y_3) = f_1(y_1, y_2, y_3)$; either $x_1 \in H_2$, forcing $y_1 \in L_1$ so by property (E) we have that $F_1(y_1, y_2, y_3) \in L_1$ adjacent to x_1 , or else $x_1 \in H_1$ and so property (D) guarantees $F_1(y_1, y_2, y_3)$ adjacent to x_1 . (b) If $\{y_1, y_2, y_3\} \subseteq H_2$, then $F_1(y_1, y_2, y_3) = g_1(y_1, y_2, y_3)$; if $x_1 \in H_1$ then it is in L_1 and is adjacent to $F_1(y_1, y_2, y_3)$; otherwise $x_1 \in H_2$ and property (D) applies. (c) If $\{y_1, y_2, y_3\}$ intersects both L_1 and H_2 , then $F_1(y_1, y_2, y_3)$ returns the leftmost entry which is in L_1 ; hence if x_1 is not in N_1 then it is adjacent to $F_1(y_1, y_2, y_3)$. If $x_1 \in N_1$ then $y_1 \in L_1$ so $F_1(y_1, y_2, y_3) = y_1$ and we are done. (d) Suppose that $\{y_1, y_2, y_3\}$ intersects both H_1 and H_2 but not L_1 : then $F_1(y_1, y_2, y_3)$ returns the leftmost entry in H_2 ; if x_1 is in H_1 then it must be in L_1 ; otherwise $x_1 \in H_2$ forces $y_1 \in H_2$; in both cases x_1 is adjacent to $F_1(y_1, y_2, y_3)$.

From now on we may assume that $x_2 \neq x_3$.

(2) Suppose $\{x_1, x_2, x_3\} \subseteq H_1$. (a) If $\{y_1, y_2, y_3\} \subseteq H_2$, then $x_i \in L_1$ for all i , so $F_1(x, y, z) \in L_1$ by property (E), so we are done. (b) If $\{y_1, y_2, y_3\}$ intersects both L_1 and H_2 , then $F_1(y_1, y_2, y_3)$ returns the leftmost entry which is in L_1 . There is some i such that $y_i \in H_2$, hence $x_i \in L_1$; by (E) it follows that $F_1(x_1, x_2, x_3) \in L_1$ and is adjacent to $F_1(y_1, y_2, y_3)$. (c) Suppose that $\{y_1, y_2, y_3\}$ intersects both H_1 and H_2 but not L_1 ; then $F_1(y_1, y_2, y_3)$ returns the leftmost entry in H_2 . There is some i such that $y_i \in H_2$, hence $x_i \in L_1$; by (E) again $F_1(x_1, x_2, x_3) \in L_1$ and is adjacent to $F_1(y_1, y_2, y_3)$.

(3) Suppose $\{x_1, x_2, x_3\} \subseteq H_2$. By the previous cases we may assume that $\{y_1, y_2, y_3\}$ intersects both H_1 and H_2 ; in this case it must also intersect L_1 , hence $F_1(y_1, y_2, y_3)$ returns the leftmost entry which is in L_1 , which is adjacent to every vertex in H_2 .

(4) Suppose $\{x_1, x_2, x_3\}$ intersects both L_1 and H_2 . If the same holds for $\{y_1, y_2, y_3\}$, the result follows from the fact that λ^L is edge-preserving. We may now assume that $\{y_1, y_2, y_3\}$ intersects both H_1 and H_2 but not L_1 . Then by definition $F_1(x_1, x_2, x_3) \in L_1$ and $F_1(y_1, y_2, y_3) \in H_2$, and hence are adjacent.

(5) Finally, suppose that each of $\{x_1, x_2, x_3\}$ and $\{y_1, y_2, y_3\}$ intersects both H_1 and H_2 , but not L_1 . Then the result follows from the fact that ν^L is edge-preserving. \square

6 Symmetric Datalog constructions

The goal of this section is to prove the following lemma.

Lemma 20. *If $\mathbf{H} \in \mathcal{F}$ then $\neg \text{CSP}(\mathbf{H}^L)$ is in symmetric Datalog.*

Recall that $\mathcal{F} = \mathcal{I}$ by Theorem 14, and we will use the inductive definition of this class in this section. We start by describing a method to compose symmetric Datalog programs.

6.1 Composing symmetric Datalog programs

The output of a Datalog program over τ with a set of IDBs I is a structure over the extended signature $\tau \cup I$. Such a structure can naturally be fed as input to another Datalog program working over this extended signature and using a set J of IDBs disjoint from I . The result is a structure over the signature $\tau \cup I \cup J$. Of course, the effect of this composition can be obtained by simply merging the list of rules of the two programs. However, this naive construction does not preserve the linearity or symmetricity of the programs. The following lemma shows that in fact symmetricity can be preserved at the cost of an increase in the arity of the IDBs.

Lemma 21. *Let \mathcal{D} be a symmetric Datalog program over the signature $\tau = \{E_1, \dots, E_k\}$ that outputs IDBs I_1, \dots, I_t of respective arities a_1, \dots, a_t . Let $\lceil I_{j_1} \wedge \dots \wedge I_{j_s} \rceil$ denote the relation of arity $a_{j_1} + \dots + a_{j_s}$ which is the Cartesian product of the (not necessarily distinct) relations I_{j_1}, \dots, I_{j_s} . For any c ,*

there exists a program \mathcal{D}_c over the signature τ which correctly derives all the $\lceil I_{j_1} \wedge \dots \wedge I_{j_s} \rceil$ with $s \leq c$.

Proof. It suffices to show that this holds when $c = 2$ since the more general statement can be obtained by iterating the construction detailed below. Note that we consider $\lceil I_k \wedge I_\ell \rceil$ as a single² IDB. We use \bar{x}, \bar{y} and so on to denote tuples of variables and say that \bar{x} and \bar{y} are *disjoint* if they share no variable. We also use $E(\bar{w})$ to denote some conjunction of EDBs. We construct \mathcal{D}_2 with the following rules.

1. Original rules of \mathcal{D} are kept.
2. If $I_j(\bar{x}) \leftarrow E(\bar{w})$ is a non-recursive rule in \mathcal{D} , we include for any $1 \leq q \leq t$ the rule

$$\lceil I_j(\bar{x}) \wedge I_q(\bar{y}) \rceil \leftarrow I_q(\bar{y}) \wedge E(\bar{w})$$

where \bar{y} is disjoint from \bar{x} and \bar{w} . We also include the symmetric rule

$$I_q(\bar{y}) \leftarrow \lceil I_j(\bar{x}) \wedge I_q(\bar{y}) \rceil \wedge E(\bar{w}).$$

3. Finally, if $I_j(\bar{x}) \leftarrow I_k(\bar{y}) \wedge E(\bar{w})$ is a recursive rule of \mathcal{D} , we include for any $1 \leq q \leq t$ the rule

$$\lceil I_q(\bar{z}) \wedge I_j(\bar{x}) \rceil \leftarrow \lceil I_q(\bar{z}) \wedge I_k(\bar{y}) \rceil \wedge E(\bar{w})$$

where \bar{z} is disjoint from \bar{x}, \bar{y} and \bar{w} . Because \mathcal{D} is symmetric, we know that the symmetric of the above rule also appears in \mathcal{D}_2 .

By construction \mathcal{D}_2 is symmetric. We claim that it computes the product relations correctly. Note first that all rules above are sound, i.e. there is a derivation in \mathcal{D}_2 for $\lceil I_j(\bar{x}) \wedge I_k(\bar{y}) \rceil$ only if there are derivations for $I_j(\bar{x})$ and $I_k(\bar{y})$ in \mathcal{D} . Indeed, one can verify that if this property is true at some stage of the derivation then it still holds after the application of any of the above rules. The same argument shows that \mathcal{D} and \mathcal{D}_2 derive the same IDBs I_1, \dots, I_t . In fact, it is convenient to view the execution of \mathcal{D}_2 as a two-stage process where the original IDBs are derived first.

It remains to show that if there are derivations in \mathcal{D} for $I_j(\bar{x})$ and $I_k(\bar{y})$ then there is a derivation of $\lceil I_j(\bar{x}) \wedge I_k(\bar{y}) \rceil$ in \mathcal{D}_2 . Note first that since there is a derivation of $I_k(\bar{y})$ in \mathcal{D} , that same derivation exists in \mathcal{D}_2 (this is the purpose of rules of type (1)). Now let

$$\rightarrow I_{j_1}(\bar{x}_1) \rightarrow \dots \rightarrow I_{j_n}(\bar{x}_n) \rightarrow I_j(\bar{x})$$

denote the sequence of IDBs used in the derivation of $I_j(\bar{x})$ in \mathcal{D} . Suppose that $I_{j_1}(\bar{x}_1)$ is derived in \mathcal{D} by instantiating a first-order rule

$$I_{j_1}(\bar{x}_1) \leftarrow E(\bar{w}_0).$$

²As stated, the lemma distinguishes the IDBs $\lceil I_j \wedge I_k \rceil$ and $\lceil I_k \wedge I_j \rceil$. However, the two are clearly equivalent from a computational perspective. To simplify our description of \mathcal{D}_2 , we thus implicitly assume that any rule involving $\lceil I_j \wedge I_k \rceil$ is accompanied by the counterpart rule using $\lceil I_k \wedge I_j \rceil$.

The rules of type (2) provide a corresponding derivation of $\lceil I_{j_1}(\bar{x}_1) \wedge I_k(\bar{y}) \rceil$ in \mathcal{D}_2 through

$$\lceil I_{j_1}(\bar{x}_1) \wedge I_k(\bar{y}) \rceil \leftarrow I_k(\bar{y}) \wedge E(\bar{w}_0).$$

Similarly, suppose that the derivation of $I_{j_2}(\bar{x}_2)$ in \mathcal{D} is given by

$$I_{j_2}(\bar{x}_2) \leftarrow I_{j_1}(\bar{x}_1) \wedge E(\bar{w}_1).$$

The rules of type (3) provide a corresponding derivation of $\lceil I_{j_2}(\bar{x}_2) \wedge I_k(\bar{y}) \rceil$ in \mathcal{D}_2 through

$$\lceil I_{j_2}(\bar{x}_2) \wedge I_k(\bar{y}) \rceil \leftarrow \lceil I_{j_1}(\bar{x}_1) \wedge I_k(\bar{y}) \rceil \wedge E(\bar{w}_1).$$

Thus, we can successively derive $\lceil I_{j_{t+1}}(\bar{x}_{t+1}) \wedge I_k(\bar{y}) \rceil$, from $\lceil I_{j_t}(\bar{x}_t) \wedge I_k(\bar{y}) \rceil$ and ultimately obtain a derivation of $\lceil I_j(\bar{x}) \wedge I_k(\bar{y}) \rceil$. \square

This construction is used to prove the following lemma which, intuitively, proves that symmetric Datalog programs can be composed in a way that preserves the symmetry.

Lemma 22. *Let \mathcal{D} be a symmetric Datalog program over the signature $\tau = \{E_1, \dots, E_q\}$, and assume that the set of IDBs of \mathcal{D} is $I = \{I_1, \dots, I_s\}$, with respective arities a_1, \dots, a_s . Further, let \mathcal{E} be a symmetric Datalog program over the signature $\tau' = \tau \cup I$, and assume that the set of IDBs of \mathcal{E} is $J = \{J_1, \dots, J_t\}$, with respective arities b_1, \dots, b_t . For a τ -structure \mathbf{H} , let \mathbf{H}' denote the τ' -structure defined by $\mathcal{D}(\mathbf{H})$. One can construct a symmetric Datalog program \mathcal{F} over the original signature τ with the following properties:*

- a) the IDBs I and J of \mathcal{D} and \mathcal{E} are IDBs in \mathcal{F} ;
- b) $I_k(\bar{x})$ is derived in $\mathcal{F}(\mathbf{H})$ iff $I_k(\bar{x})$ is derived in $\mathcal{D}(\mathbf{H})$;
- c) $J_\ell(\bar{x})$ is derived in $\mathcal{F}(\mathbf{H})$ iff $J_\ell(\bar{x})$ is derived in $\mathcal{E}(\mathbf{H}')$.

Proof. Let c be such that each rule of \mathcal{E} uses at most c EDBs in $\tau' - \tau$, i.e. at most c of the IDBs of \mathcal{D} . By the previous lemma, we can assume that $c = 1$ since we can otherwise construct \mathcal{D}_c and thus get relations that represent any conjunction of the I_j .

The IDBs of our new program \mathcal{F} include the IDBs of \mathcal{D} and \mathcal{E} (i.e. the IDBs in $I \cup J$) as well as auxiliary IDBs of the form $\lceil J_\ell \wedge I_k \rceil$ for any $1 \leq k \leq s$ and $1 \leq \ell \leq t$. The rules of \mathcal{F} are constructed as follows.

1. Every rule of \mathcal{D} is kept.
2. All rules of \mathcal{E} which use only EDBs of the original signature τ are kept.
3. For any non-recursive rule of \mathcal{D} , say $I_k(\bar{x}) \leftarrow E(\bar{w})$, we include for each $1 \leq \ell \leq t$ the symmetric pair of rules

$$\lceil J_\ell(\bar{y}) \wedge I_k(\bar{x}) \rceil \leftarrow J_\ell(\bar{y}) \wedge E(\bar{w})$$

$$J_\ell(\bar{y}) \leftarrow \lceil J_\ell(\bar{y}) \wedge I_k(\bar{x}) \rceil \wedge E(\bar{w})$$

where \bar{y} is disjoint from \bar{x} and \bar{w} .

4. For any recursive rule of \mathcal{D} , say $I_{k_1}(\bar{x}_1) \leftarrow I_{k_2}(\bar{x}_2) \wedge E(\bar{w})$, we include for each $1 \leq \ell \leq t$ the rule

$$\lceil J_\ell(\bar{y}) \wedge I_{k_1}(\bar{x}_1) \rceil \leftarrow \lceil J_\ell(\bar{y}) \wedge I_{k_2}(\bar{x}_2) \rceil \wedge E(\bar{w})$$

where \bar{y} is disjoint from \bar{x}_1, \bar{x}_2 and \bar{w} . The symmetricity of \mathcal{D} ensures that \mathcal{F} contains the symmetric rule:

$$\lceil J_\ell(\bar{y}) \wedge I_{k_2}(\bar{x}_2) \rceil \leftarrow \lceil J_\ell(\bar{y}) \wedge I_{k_1}(\bar{x}_1) \rceil \wedge E(\bar{w}).$$

5. For any non-recursive rule of \mathcal{E} that uses one of the I_k as an EDB, say $J_\ell(\bar{y}) \leftarrow I_k(\bar{x}) \wedge E(\bar{w})$, we include the symmetric pair of (recursive) rules

$$\lceil J_\ell(\bar{y}) \wedge I_k(\bar{x}) \rceil \leftarrow I_k(\bar{x}) \wedge E(\bar{w})$$

$$I_k(\bar{x}) \leftarrow \lceil J_\ell(\bar{y}) \wedge I_k(\bar{x}) \rceil \wedge E(\bar{w}).$$

6. For any non-recursive rule of \mathcal{E} that does not use any of the I_k as an EDB, say $J_\ell(\bar{y}) \leftarrow E(\bar{w})$, we include for each $1 \leq k \leq s$ the symmetric pair of (recursive) rules

$$\lceil J_\ell(\bar{y}) \wedge I_k(\bar{x}) \rceil \leftarrow I_k(\bar{x}) \wedge E(\bar{w})$$

$$I_k(\bar{x}) \leftarrow \lceil J_\ell(\bar{y}) \wedge I_k(\bar{x}) \rceil \wedge E(\bar{w}).$$

7. For any recursive rule of \mathcal{E} that use some I_k as an EDB, say $J_{\ell_1}(\bar{y}_1) \leftarrow J_{\ell_2}(\bar{y}_2) \wedge I_k(\bar{x}) \wedge E(\bar{w})$, we include the rule

$$\lceil J_{\ell_1}(\bar{y}_1) \wedge I_k(\bar{x}) \rceil \leftarrow \lceil J_{\ell_2}(\bar{y}_2) \wedge I_k(\bar{x}) \rceil \wedge E(\bar{w}).$$

Because \mathcal{E} is symmetric, we know that \mathcal{F} also includes the rule

$$\lceil J_{\ell_2}(\bar{y}_2) \wedge I_k(\bar{x}) \rceil \leftarrow \lceil J_{\ell_1}(\bar{y}_1) \wedge I_k(\bar{x}) \rceil \wedge E(\bar{w}).$$

8. For any recursive rule of \mathcal{E} that does not use an I_j as an EDB, say $J_{\ell_1}(\bar{y}_1) \leftarrow J_{\ell_2}(\bar{y}_2) \wedge E(\bar{w})$, we include for each $1 \leq k \leq s$ the rule

$$\lceil J_{\ell_1}(\bar{y}_1) \wedge I_k(\bar{x}) \rceil \leftarrow \lceil J_{\ell_2}(\bar{y}_2) \wedge I_k(\bar{x}) \rceil \wedge E(\bar{w})$$

where \bar{x} is disjoint from \bar{y}_1, \bar{y}_2 and \bar{w} . Because \mathcal{E} is symmetric, we know that \mathcal{F} also includes the rule

$$\lceil J_{\ell_2}(\bar{y}_2) \wedge I_k(\bar{x}) \rceil \leftarrow \lceil J_{\ell_1}(\bar{y}_1) \wedge I_k(\bar{x}) \rceil \wedge E(\bar{w}).$$

We claim that \mathcal{F} has the desired properties. Again, we first note that all the rules are sound: if there is a derivation in $\mathcal{F}(\mathbf{H})$ for $I_k(\bar{x})$ (resp. $J_\ell(\bar{y})$; $\lceil J_\ell(\bar{y}) \wedge I_k(\bar{x}) \rceil$) then there is a derivation for $I_k(\bar{x})$ in $\mathcal{D}(\mathbf{H})$ (resp. a derivation for $J_\ell(\bar{y})$ in $\mathcal{E}(\mathbf{H}')$; derivations for $I_k(\bar{x})$ in $\mathcal{D}(\mathbf{H})$ and for $J_\ell(\bar{y})$ in $\mathcal{E}(H')$). In other words, none of the above rules can produce unwanted tuples.

It remains to show that \mathcal{F} is complete, i.e. that if there exists a derivation for $I_k(\bar{x})$ in $\mathcal{D}(\mathbf{H})$ and a derivation for $J_\ell(\bar{y})$ in $\mathcal{E}(\mathbf{H}')$ then there are derivations in $\mathcal{F}(\mathbf{H})$ for $I_k(\bar{x})$, $J_\ell(\bar{y})$ and $\lceil J_\ell(\bar{y}) \wedge I_k(\bar{x}) \rceil$. This is obvious for $I_k(\bar{x})$ since the original rules of \mathcal{D} are also rules of \mathcal{F} . Similarly, rules of type (2) yield the claim if the derivation of $J_\ell(\bar{y})$ in $\mathcal{E}(\mathbf{H}')$ never uses one of the I_j as an EDB.

The non-trivial case consists of derivations of $\mathcal{E}(\mathbf{H}')$ which use the I_j as EDBs. The crux of the argument rests on the possibility of “inverting” any sequence of derivation steps in a symmetric program. Consider the sequence of IDBs in a valid derivation path in $\mathcal{D}(\mathbf{H})$:

$$I_{k_1}(\bar{x}_1) \rightarrow I_{k_2}(\bar{x}_2) \rightarrow \dots \rightarrow I_{k_n}(\bar{x}_n).$$

If the i th step is obtained as $I_{k_{i+1}}(\bar{x}_{i+1}) \leftarrow I_{k_i}(\bar{x}_i) \wedge E(\bar{w}_i)$ then the symmetricity of \mathcal{D} ensures that $I_{k_i}(\bar{x}_i) \leftarrow I_{k_{i+1}}(\bar{x}_{i+1}) \wedge E(\bar{w}_i)$ is also a valid derivation step in $\mathcal{D}(\mathbf{H})$ and the sequence

$$I_{k_n}(\bar{x}_n) \rightarrow I_{k_{n-1}}(\bar{x}_{k_{n-1}}) \rightarrow \dots \rightarrow I_{k_1}(\bar{x}_1)$$

also corresponds to a valid derivation path. In other words, if $\mathcal{D}(\mathbf{H})$ can produce a derivation of $I_{k_n}(\bar{x}_n)$ from $I_{k_1}(\bar{x}_1)$ then it can also produce a derivation of $I_{k_1}(\bar{x}_1)$ from $I_{k_n}(\bar{x}_n)$.

We begin with the following claim.

Claim 1. Assume that there exists a derivation of $I_k(\bar{x})$ in $\mathcal{D}(H)$. Then there exists a derivation of $J_\ell(\bar{y})$ in $\mathcal{F}(H)$ iff there exists a derivation of $\lceil J_\ell(\bar{y}) \wedge I_k(\bar{x}) \rceil$ in $\mathcal{F}(H)$.

Proof. In the left to right implication, we assume that $J_\ell(\bar{y})$ is derived in $\mathcal{F}(H)$ and use a simple induction on the length of the derivation of $I_k(\bar{x})$ in $\mathcal{D}(H)$. If $I_k(\bar{x})$ is derived from a non-recursive rule then the derivation of $\lceil J_\ell(\bar{y}) \wedge I_k(\bar{x}) \rceil$ in $\mathcal{F}(H)$ is obtained through a rule of type (3). The induction step is obtained through rules of type (4).

The right to left implication is established through the same basic idea but using the inverse path of derivation. Assume that $\lceil J_\ell(\bar{y}) \wedge I_k(\bar{x}) \rceil$ can be derived in $\mathcal{F}(\mathbf{H})$. If $I_k(\bar{x})$ is derived from a non-recursive rule then we can derive $J_\ell(\bar{y})$ from $\lceil J_\ell(\bar{y}) \wedge I_k(\bar{x}) \rceil$ because of the symmetric rule of type (3). If the derivation of $I_k(\bar{x})$ in $\mathcal{D}(\mathbf{H})$ is of length at least 2 then consider the last derivation step, say:

$$I_k(\bar{x}) \leftarrow I_{k'}(\bar{x}') \wedge E(\bar{w}).$$

By induction, there exists a derivation from $\lceil J_\ell(\bar{y}) \wedge I_{k'}(\bar{x}') \rceil$ to $J_\ell(\bar{y})$ and the symmetric rule of type (4) provides the missing derivation step:

$$\lceil J_\ell(\bar{y}) \wedge I_{k'}(\bar{x}') \rceil \leftarrow \lceil J_\ell(\bar{y}) \wedge I_k(\bar{x}) \rceil \wedge E(\bar{w}).$$

This completes the proof of the claim. □

Note that Claim 1 only involves derivations in $\mathcal{F}(\mathbf{H})$ but to complete the proof of the lemma, we need to prove that any derivation of $J_\ell(\bar{y})$ in $\mathcal{E}(\mathbf{H}')$ has a corresponding derivation of $J_\ell(\bar{y})$ in $\mathcal{F}(\mathbf{H})$.

Suppose that the derivation of $J_\ell(\bar{y})$ in $\mathcal{E}(\mathbf{H}')$ has length n . If $n = 1$ then $J_\ell(\bar{y})$ is derived in $\mathcal{E}(\mathbf{H}')$ from a non-recursive rule which may or may not use one of the I_j as an EDB. By using a rule of type (5) or (6), we can obtain in $\mathcal{F}(\mathbf{H})$ a derivation for some $\lceil J_\ell(\bar{y}) \wedge I_k(\bar{x}) \rceil$ where $I_k(\bar{x})$ has a derivation³ in $\mathcal{D}(\mathbf{H})$. It then follows from Claim 1 that we can obtain in $\mathcal{F}(\mathbf{H})$ a derivation for $J_\ell(\bar{y})$ as well as derivations for *any* of the $\lceil J_\ell(\bar{y}) \wedge I_{k'}(\bar{x}') \rceil$ when $I_{k'}(\bar{x}')$ has a derivation in $\mathcal{D}(\mathbf{H})$.

For the induction step, take $n \geq 2$ and suppose the last step in the derivation of $J_\ell(\bar{y})$ in $\mathcal{E}(\mathbf{H}')$ is given by

$$J_\ell(\bar{y}) \leftarrow J_{\ell_n}(\bar{y}_n) \wedge I_{k_n}(\bar{x}_n) \wedge E(\bar{w}_n).$$

We know by the induction hypothesis that there is a derivation in $\mathcal{F}(\mathbf{H})$ for $\lceil J_{\ell_n}(\bar{y}_n) \wedge I_{k_n}(\bar{x}_n) \rceil$. A rule of type (7) can now complete the derivation of $\lceil J_\ell(\bar{y}) \wedge I_{k_n}(\bar{x}_n) \rceil$ in $\mathcal{F}(\mathbf{H})$:

$$\lceil J_\ell(\bar{y}) \wedge I_{k_n}(\bar{x}_n) \rceil \leftarrow \lceil J_{\ell_n}(\bar{y}_n) \wedge I_{k_n}(\bar{x}_n) \rceil \wedge E(\bar{w}_n).$$

Finally, Claim 1 ensures that $J_\ell(\bar{y})$ itself can be derived in $\mathcal{F}(\mathbf{H})$. The case where the last derivation step of $J_\ell(\bar{y})$ in $\mathcal{E}(\mathbf{H}')$ does not rely on one of the I_j as an EDB is covered by rules of type (8). \square

6.2 Symmetric programs for the list-homomorphism problem for graphs in \mathcal{F}

Our objective is to show that for any undirected graph \mathbf{H} in \mathcal{F} the set $\neg \text{CSP}(\mathbf{H}^L)$ of digraphs with \mathbf{H} -lists that do not map homomorphically to \mathbf{H}^L is definable in symmetric Datalog. We proceed by induction on the structure of \mathbf{H} , i.e. using the inductive definition of \mathcal{F} . If \mathbf{H} consists of a single loop or non-loop, $\neg \text{CSP}(\mathbf{H}^L)$ is trivially definable in symmetric Datalog and it remains to show that this property is preserved by the operators disjoint union, basic graph adjunction, formation of a basic graph by completion of a colour class and special sum.

We begin with simple but useful observations that allow more concise and intuitive descriptions of our constructions. These remarks and basic tricks all rely on Lemma 22.

1. In a number of constructions below, we want to obtain from two symmetric Datalog programs \mathcal{D}_1 and \mathcal{D}_2 with goal predicates T_1 and T_2 , respectively, a new symmetric program \mathcal{D} which accepts an input \mathbf{G} if \mathbf{G} is accepted

³Note that we can assume without loss of generality that there is at least one such $I_k(\bar{x})$. If all the I_k are empty in $\mathcal{D}(\mathbf{H})$ then derivations in $\mathcal{E}(\mathbf{H}')$ can only be constructed from rules that do not use the I_k as EDBs and, as we remarked earlier, the rules of type (2) cover this case.

by \mathcal{D}_1 or if \mathbf{G} is accepted by \mathcal{D}_2 . This can be done effortlessly since we can simply take the union of the rules of \mathcal{D}_1 and \mathcal{D}_2 , create a new goal predicate T and include the rules $T \leftarrow T_1$ and $T \leftarrow T_2$.

If instead we want \mathcal{D} to accept \mathbf{G} if both \mathcal{D}_1 and \mathcal{D}_2 accept \mathbf{G} , we can use Lemma 22 as follows. Consider the relational structure output by \mathcal{D}_1 : this structure includes the relation T_1 which we can now use as an EDB in \mathcal{D} . It now suffices to add T_1 to the body of any rule of \mathcal{D}_2 which has T_2 as its head.

2. When analysing programs we always assume that the input structure \mathbf{G} is connected⁴. This is possible without loss of generality. Indeed, consider Datalog programs over a signature τ that contains a binary relation E (this is the case in all our constructions). It is straightforward to define a k -ary relation C_E in symmetric Datalog which contains the tuple (x_1, \dots, x_k) iff all x_i are in the same connected component of E . Moreover, Lemma 22 allows us to assume that any program \mathcal{D} has access to this relation as an EDB.

Suppose that the body of each rule in \mathcal{D} includes the EDB condition $C_E(x_1, \dots, x_k)$ where the x_i are the variables occurring in the rule. Note that if \mathbf{G} is a digraph given as input to \mathcal{D} , then any derivation of $\mathcal{D}(\mathbf{G})$ must now take place within a single connected component of \mathbf{G} . So the digraphs accepted by \mathcal{D} are disjoint unions of connected digraphs (recall Footnote 4), one of which is accepted by \mathcal{D} . In our case, we construct Datalog programs which accept a digraph \mathbf{G} with \mathbf{H} -lists iff there is no homomorphism from \mathbf{G}^L to \mathbf{H}^L and this of course holds iff there is some connected component of \mathbf{G} that does not map to \mathbf{H}^L . In proving the correctness of a given program, we can therefore assume connectivity of the input structure without loss of generality.

3. Let \mathbf{G} be a digraph with \mathbf{H} -lists. Any $v \in G$ is potentially bound by more than one unary predicate but of course this amounts to imposing a list on v which is the intersection of all such unary predicates. Clearly, we can construct a simple symmetric Datalog program which returns a digraph \mathbf{G}' with \mathbf{H} -lists over the same set of vertices but such that every v is bound by a unique list L_v . In the same vein, if T is a subset of vertices of \mathbf{H} and if \mathbf{G} is a digraph with \mathbf{H} -lists, we can construct in symmetric Datalog a digraph $\mathbf{G}^{\cap T}$ in which every vertex v is bound by a list L'_v which is the intersection of the original L_v with T .

Furthermore, in the constructions below we typically assume that symmetric programs $\mathcal{D}_1, \mathcal{D}_2$ exist for $\neg \text{CSP}(\mathbf{H}_1^L)$ and $\neg \text{CSP}(\mathbf{H}_2^L)$ and construct a symmetric program \mathcal{D} for $\neg \text{CSP}(\mathbf{H}^L)$ where \mathbf{H} is a graph obtained by combining \mathbf{H}_1 and \mathbf{H}_2 through some operator. Strictly speaking, the inputs of \mathcal{D} are digraphs with \mathbf{H} -lists and thus cannot be fed as inputs to

⁴Note that because the target graph is an undirected graph, the direction of the edges of \mathbf{G} can be ignored. Therefore we say that two vertices u and v in \mathbf{G} are *connected* if u and v are connected in \mathbf{G} once the direction of the edges are ignored.

\mathcal{D}_1 or \mathcal{D}_2 since the latter only deal with lists contained in \mathbf{H}_1 and \mathbf{H}_2 , respectively. Note however that $\mathbf{G}^{\cap H_1}$ can be used as an input to \mathcal{D}_1 and we use this trick repeatedly. If it is needed, we can also use symmetric programs to construct digraphs \mathbf{G}_1 and \mathbf{G}_2 with, respectively, \mathbf{H}_1 -lists and \mathbf{H}_2 -lists and new edge relations denoted E_1 and E_2 , respectively. We can further modify the rules of \mathcal{D}_1 and \mathcal{D}_2 by replacing the occurrences of E by E_1 and E_2 and by our first remark, we can then construct a symmetric program \mathcal{D} that accepts \mathbf{G} iff \mathcal{D}_1 accepts \mathbf{G}_1 and \mathcal{D}_2 accepts \mathbf{G}_2 or a program \mathcal{D} that accepts \mathbf{G} iff \mathcal{D}_1 accepts \mathbf{G}_1 or \mathcal{D}_2 accepts \mathbf{G}_2 .

By the inductive definition of \mathcal{F} , i.e. the definition of \mathcal{I} , the following lemmas complete the proof of Lemma 20.

Lemma 23. *Suppose $\neg \text{CSP}(\mathbf{H}_1^L)$ and $\neg \text{CSP}(\mathbf{H}_2^L)$ are definable in symmetric Datalog and let \mathbf{H} be the disjoint union of \mathbf{H}_1 and \mathbf{H}_2 . Then $\neg \text{CSP}(\mathbf{H}^L)$ is also definable in symmetric Datalog.*

Proof. Suppose $\neg \text{CSP}(\mathbf{H}_1^L)$ and $\neg \text{CSP}(\mathbf{H}_2^L)$ are recognized by symmetric Datalog programs \mathcal{D}_1 and \mathcal{D}_2 with respective goal predicates K_1 and K_2 . If \mathbf{H} is the disjoint union of \mathbf{H}_1 and \mathbf{H}_2 , it is clear that a connected digraph \mathbf{G} with \mathbf{H} -lists maps into \mathbf{H}^L iff \mathbf{G} maps to \mathbf{H}_1^L or to \mathbf{H}_2^L . Of course \mathbf{G} maps to \mathbf{H}_i^L iff $\mathbf{G}^{\cap H_i}$ does. In other words, \mathbf{G} does not map into \mathbf{H}^L iff \mathcal{D}_1 accepts $\mathbf{G}^{\cap H_1}$ and \mathcal{D}_2 accepts $\mathbf{G}^{\cap H_2}$. As we noted in the above remarks, this can be tested with a symmetric program \mathcal{D} . \square

Lemma 24. *Let \mathbf{H}_1 be an irreflexive basic graph with colour classes B and T and assume that $\neg \text{CSP}(\mathbf{H}_1^L)$ is recognized by a symmetric Datalog program \mathcal{D}_1 . If \mathbf{H} is the graph obtained by transforming T into a reflexive clique then $\neg \text{CSP}(\mathbf{H}^L)$ is also definable in symmetric Datalog.*

Proof. Let \mathbf{G} be a digraph with \mathbf{H} -lists. Suppose h is a homomorphism from \mathbf{G}^L to \mathbf{H}^L . Because \mathbf{H}_1 is bipartite, any $g \in G$ mapped to some $b \in B$ must have its neighbors mapped to T . So if g has some element t of T in its list we still have a homomorphism if we set $h(g) = t$. We can therefore assume that all lists of \mathbf{G} are either contained in T or contained in B and, more precisely, we can use a symmetric Datalog program to trim the lists of \mathbf{G} accordingly. Denote as G_T and G_B the resulting partition of \mathbf{G} 's vertices. If G_B contains a loop there can be no homomorphism from \mathbf{G}^L to \mathbf{H}^L and this condition is trivial to check with symmetric Datalog. Let \mathbf{G}' be the digraph obtained from \mathbf{G} by deleting all edges between vertices in G_T . One can verify that \mathbf{G}^L maps to \mathbf{H}^L iff \mathbf{G}' maps to \mathbf{H}_1^L . It remains to show that we can construct a symmetric program \mathcal{D} which, on input \mathbf{G}^L , outputs \mathbf{G}' . We have already shown that \mathbf{G}^L can be assumed to have lists either contained in B or T and, accordingly, we can assume that \mathcal{D} is given unary EDBs G_T and G_B . The edge relation of \mathbf{G}' can now be defined with the non-recursive rules $E'(x, y) \leftarrow E(x, y) \wedge G_T(x) \wedge G_B(y)$ and $E'(x, y) \leftarrow E(x, y) \wedge G_T(y) \wedge G_B(x)$. \square

Lemma 25. *Let \mathbf{H}_1 be a basic graph and assume that $\neg\text{CSP}(\mathbf{H}_1^L)$ and $\neg\text{CSP}(\mathbf{H}_2^L)$ are recognized by symmetric Datalog programs $\mathcal{D}_1, \mathcal{D}_2$. If \mathbf{H} is the result of adjoining \mathbf{H}_1 to \mathbf{H}_2 then $\neg\text{CSP}(\mathbf{H}^L)$ is also definable in symmetric Datalog.*

Proof. For $i \in \{1, 2\}$, let R_i, I_i respectively denote the set of loops and non-loops of \mathbf{H}_i . Recall that the adjunction of \mathbf{H}_1 to \mathbf{H}_2 is the graph obtained by taking the disjoint union of the two graphs and adding every edge from R_1 to H_2 . Moreover, because \mathbf{H}_1 is basic, its loops R_1 form a clique and there are no edges between I_1 and H_2 . Let \mathbf{G}^L be a digraph with \mathbf{H} -lists and consider some vertex g whose list contains some element of $r \in R_1$. If there is any homomorphism from \mathbf{G}^L to \mathbf{H}^L then there is one such that $h(g) \in R_1$. Indeed, if $u \in G$ is such that $h(u) \in I_1$ then if (u, g) is an edge in \mathbf{G} we *must* have $h(g) \in R_1$ and if $h(u) \notin I_1$ then $h(u)$ has an edge to any $r \in R_1$. Similarly, if the list of g contains no element of R_1 but contains elements of both I_1 and H_2 , then we can assume that h maps g to some element of H_2 because mapping g to I_1 forces its neighbors to be mapped in R_1 . These observations allow us to assume that each list in \mathbf{G}^L is either contained in R_1 , contained in I_1 or contained in H_2 . We can use a symmetric program to trim the lists accordingly and obtain unary predicates r_1, i_1 and h_2 that represent R_1, I_1 and H_2 , respectively.

Let \mathbf{G}_1 and \mathbf{G}_2 be the subdigraphs of \mathbf{G} induced respectively by vertices in $r_1 \cup i_1$ and vertices in h_2 . It is now clear that \mathbf{G}^L maps to \mathbf{H}^L iff \mathbf{G}_1^L maps to \mathbf{H}_1^L , \mathbf{G}_2^L maps to \mathbf{H}_2^L and \mathbf{G} contains no edge with one endpoint in i_1 and the other in h_2 . As in the previous proofs, this condition can be easily checked because \mathbf{G}_1 and \mathbf{G}_2 can be constructed in symmetric Datalog. \square

Lemma 26. *Let \mathbf{H}_1 and \mathbf{H}_2 be irreflexive graphs such that $\neg\text{CSP}(\mathbf{H}_1^L)$ and $\neg\text{CSP}(\mathbf{H}_2^L)$ are recognized by symmetric Datalog programs $\mathcal{D}_1, \mathcal{D}_2$. If \mathbf{H} is the special sum of \mathbf{H}_1 and \mathbf{H}_2 then $\neg\text{CSP}(\mathbf{H}^L)$ is also definable in symmetric Datalog.*

Proof. Recall that the special sum of bipartite irreflexive graphs \mathbf{H}_1 and \mathbf{H}_2 with colour classes B_i, T_i consists of the disjoint union of the graphs in which all edges between T_1 and B_2 are added. Note first that \mathbf{G} must be bipartite in order to map to \mathbf{H} and since bipartiteness can be checked in symmetric Datalog, we can assume that any input \mathbf{G} is indeed bipartite. More importantly, we can construct a symmetric Datalog program that outputs unary relations B_G, T_G giving the colour classes of \mathbf{G} and use them as EDBs in the sequel. Any homomorphism from \mathbf{G} to \mathbf{H} must either map all $g \in B_G$ to $B_1 \cup B_2$ and all $g \in T_G$ to $T_1 \cup T_2$ or map all $g \in B_G$ to $T_1 \cup T_2$ and all $g \in T_G$ to $B_1 \cup B_2$. To check if \mathbf{G}^L is in $\neg\text{CSP}(\mathbf{H}^L)$ it suffices to verify that neither of these options is viable. We show how to rule out the existence of a homomorphism h mapping B_G to $B_1 \cup B_2$ and T_G to $T_1 \cup T_2$; the other case is handled symmetrically (no pun intended). First we construct a digraph \mathbf{G}' by trimming the lists of \mathbf{G}^L , i.e. by intersecting the list of any $g \in T_G$ with $T_1 \cup T_2$ and the list of any $g \in B_G$ with $B_1 \cup B_2$. We claim that if h is a homomorphism from $(\mathbf{G}')^L$ to \mathbf{H}^L and if the list of $g \in T_G$ contains elements from both T_1 and T_2 then there exists a homomorphism h' which maps g to some element of T_1 . Indeed, every

$v \in T_2$ only has neighbors in B_2 , all of which are connected to every vertex in T_1 . We can therefore trim our lists further so that every list of a $g \in T_G$ is either contained in T_1 or contained in T_2 . Similarly, we trim the lists so that every list of a $g \in B_G$ is either contained in B_1 or contained in B_2 . We can therefore construct in symmetric Datalog a digraph \mathbf{G}'' with \mathbf{H} -lists contained in one of the four B_i, T_i . Since there are edges in \mathbf{H} between any vertex in T_1 and any vertex in B_2 we can safely ignore the edges in \mathbf{G}'' that connect vertices whose lists are respectively contained in T_1 and B_2 . On the other hand, if \mathbf{G}'' contains an edge between vertices whose lists are respectively contained in B_1 and T_2 then there can be no homomorphism from \mathbf{G}'' to \mathbf{H}^L . With this possibility ruled out, we construct in symmetric Datalog the digraphs \mathbf{G}_1 and \mathbf{G}_2 induced by the vertices whose lists are in $T_1 \cup B_1$ and $T_2 \cup B_2$, respectively. There is a homomorphism from \mathbf{G}'' to \mathbf{H} iff \mathbf{G}_1 maps to \mathbf{H}_1^L and \mathbf{G}_2 maps to \mathbf{H}_2^L and this can be checked using \mathcal{D}_1 and \mathcal{D}_2 , respectively. \square

7 List homomorphism problems definable in first-order logic

In this section we prove Theorem 5. We need the following characterisation of structures whose CSP is first-order definable [24]. Let \mathbf{T} be a relational structure and let $a, b \in T$. We say that b *dominates* a in \mathbf{T} if, for any relation $R(\mathbf{T})$, and any tuple $\bar{t} \in R(\mathbf{T})$, replacement of any occurrence of a by b in \bar{t} will yield a tuple of $R(\mathbf{T})$. Recall the definition of a direct power of a structure from Subsection 2.1. If \mathbf{T} is a relational structure, we say that the structure \mathbf{T}^2 *dismantles to the diagonal* if there exists a sequence of elements $\{a_0, \dots, a_n\} = T^2 \setminus \{(a, a) : a \in T\}$ such that, for all $0 \leq i \leq n$, a_i is dominated in \mathbf{T}_i , where $\mathbf{T}_0 = \mathbf{T}^2$ and \mathbf{T}_i is the substructure of \mathbf{T}^2 induced by $T^2 \setminus \{a_0, \dots, a_{i-1}\}$ for $i > 0$.

Lemma 27 ([24]). *Let \mathbf{T} be a core relational structure. Then $\text{CSP}(\mathbf{T})$ is first-order definable if and only if \mathbf{T}^2 dismantles to the diagonal.*

Proof. (of Theorem 5). We first prove that conditions (i) and (ii) are necessary. Notice that if $\text{CSP}(\mathbf{H}^L)$ is first-order definable then so is $\text{CSP}(\mathbf{K}^L)$ for any induced substructure \mathbf{K} of \mathbf{H} . Let x and y be distinct vertices of \mathbf{H} and let \mathbf{K}^L be the substructure of \mathbf{H}^L induced by $\{x, y\}$. If x and y are non-adjacent loops, then $\theta(\mathbf{K}) = \{(x, x), (y, y)\}$ is the equality relation on $\{x, y\}$; if x and y are adjacent non-loops, then $\theta(\mathbf{K}) = \{(x, y), (y, x)\}$, the adjacency relation of the complete graph on 2 vertices. It is well known (and can be easily derived from Lemma 27) that neither of these classes $\text{CSP}(\mathbf{K}^L)$ is first-order definable. It follows that the loops of \mathbf{H} induce a complete graph and the non-loops induce a graph with no edges.

Now we prove (iii) is necessary. Suppose for a contradiction that there exist distinct elements x and y of N and elements n and m of L such that m is adjacent to x but not to y , and n is adjacent to y but not to x . Then $\text{CSP}(\mathbf{G})$ is

first-order definable, where \mathbf{G} is the substructure of \mathbf{H}^L induced by $\{x, y, m, n\}$. By Lemma 27, \mathbf{G}^2 dismantles to the diagonal. Then (x, y) must be dominated by one of (x, x) , (y, x) or (y, y) , since domination respects the unary relation $\{x, y\}^2$ (on G^2). But (m, n) is a neighbour of (x, y) and none of the other three, a contradiction.

For the converse: we show that we can dismantle $(\mathbf{H}^L)^2$ to the diagonal. Let $x \in H$: then (x_1, x) and (x, x_1) are dominated by (x, x) . Suppose that we have dismantled every element containing a coordinate equal to x_i with $i \leq j - 1$: if x is any element of H such that the elements (x_j, x) and (x, x_j) remain, then either x is a loop or $x = x_k$ with $k \geq j$; in any case the elements (x_j, x_k) and (x_k, x_j) are dominated by (x, x) . In this way we can remove all pairs (x, y) with one of x or y a non-loop. For the remaining pairs, notice that if u and v are any loops then (u, v) is dominated (in what remains of $(\mathbf{H}^L)^2$) by (u, u) . \square

References

- [1] E. Allender, M. Bauland, N. Immerman, H. Schnoor, and H. Vollmer. The complexity of satisfiability problems: Refining Schaefer’s theorem. *Journal of Computer and System Sciences*, 75(4):245–254, 2009.
- [2] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
- [3] L. Barto and M. Kozik. Constraint satisfaction problems of bounded width. In *FOCS’09*, 2009.
- [4] L. Barto, M. Kozik, and T. Niven. The CSP dichotomy holds for digraphs with no sources and no sinks (A positive answer to a conjecture of Bang-Jensen and Hell). *SIAM J. Comput.*, 38(5):1782–1802, 2009.
- [5] R. Brewster, T. Feder, P. Hell, J. Huang, and G. MacGillavray. Near-unanimity functions and varieties of reflexive graphs. *SIAM J. Discrete Math.*, 22:938–960, 2008.
- [6] A. Bulatov. Tractable conservative constraint satisfaction problems. In *LICS’03*, pages 321–330, 2003.
- [7] A. Bulatov, P. Jeavons, and A. Krokhin. Classifying complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, 2005.
- [8] A. Bulatov, A. Krokhin, and B. Larose. Dualities for constraint satisfaction problems. In *Complexity of Constraints*, volume 5250 of *LNCS*, pages 93–124. 2008.
- [9] A. Bulatov and M. Valeriote. Recent results on the algebraic approach to the CSP. In *Complexity of Constraints*, volume 5250 of *LNCS*, pages 68–92. 2008.

- [10] C. Carvalho, V. Dalmau, and A. Krokhin. Caterpillar duality for constraint satisfaction problems. In *LICS'08*, pages 307–316, 2008.
- [11] C. Carvalho, V. Dalmau, and A. Krokhin. CSP duality and trees of bounded pathwidth. *Theoretical Computer Science*, accepted for publication, 2010.
- [12] D. Cohen and P. Jeavons. The complexity of constraint languages. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 8. Elsevier, 2006.
- [13] V. Dalmau. Linear Datalog and bounded path duality for relational structures. *Logical Methods in Computer Science*, 1(1), 2005. (electronic).
- [14] V. Dalmau and A. Krokhin. Majority constraints have bounded pathwidth duality. *European Journal of Combinatorics*, 29(4):821–837, 2008.
- [15] V. Dalmau and B. Larose. Maltsev + Datalog \Rightarrow Symmetric Datalog. In *LICS'08*, pages 297–306, 2008.
- [16] L. Egri, B. Larose, and P. Tesson. Symmetric Datalog and constraint satisfaction problems in Logspace. In *LICS'07*, pages 193–202, 2007.
- [17] T. Feder, P. Hell, and J. Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19:487–505, 1999.
- [18] T. Feder, P. Hell, and J. Huang. Bi-arc graphs and the complexity of list homomorphisms. *Journal of Graph Theory*, 42:61–80, 2003.
- [19] 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.*, 28:57–104, 1998.
- [20] F. Gurski. Characterizations of co-graphs defined by restricted NLC-width or clique-width operations. *Discrete Mathematics*, 306(2):271–277, 2006.
- [21] P. Hell and J. Nešetřil. *Graphs and Homomorphisms*. Oxford University Press, 2004.
- [22] D. Hobby and R.N. McKenzie. *The Structure of Finite Algebras*. AMS, Providence, R.I., 1988.
- [23] Ph.G. Kolaitis and M.Y. Vardi. A logical approach to constraint satisfaction. In *Complexity of Constraints*, volume 5250 of *LNCS*, pages 125–155. 2008.
- [24] B. Larose, C. Loten, and C. Tardif. A characterisation of first-order constraint satisfaction problems. *Logical Methods in Computer Science*, 3(4), 2007. (electronic).
- [25] B. Larose and P. Tesson. Universal algebra and hardness results for constraint satisfaction problems. *Theoretical Computer Science*, 410(18):1629–1647, 2009.

- [26] B. Larose and L. Zádori. Bounded width problems and algebras. *Algebra Univ.*, 56(3-4):439–466, 2007.
- [27] M. Maróti and R. McKenzie. Existence theorems for weakly symmetric operations. *Algebra Univ.*, 59(3-4):463–489, 2008.
- [28] R. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147, 2003.
- [29] R.N. McKenzie, G.F. McNulty, and W.F. Taylor. *Algebras, Lattices and Varieties, Vol.I*. Wadsworth and Brooks, 1987.
- [30] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [31] T.J. Schaefer. The complexity of satisfiability problems. In *STOC'78*, pages 216–226, 1978.
- [32] M. Valeriote. A subalgebra intersection property for congruence-distributive varieties. *Canadian Journal of Mathematics*, 61(2):451–464, 2009.