Symmetric Datalog and Constraint Satisfaction Problems in Logspace

László Egri
McGill University, Montréal, Canada
legril@cs.mcgill.ca

Benoit Larose
Concordia University, Montréal, Canada
larose@mathstat.concordia.ca

Pascal Tesson
Laval University, Québec, Canada
pascal.tesson@ift.ulaval.ca

Abstract

We introduce symmetric Datalog, a syntactic restriction of linear Datalog and show that its expressive power is exactly that of restricted symmetric monotone Krom SNP. The deep result of Reingold [17] on the complexity of undirected connectivity suffices to show that symmetric Datalog queries can be evaluated in logarithmic space. We show that for a number of constraint languages \( \Gamma \), the complement of the constraint satisfaction problem \( \text{CSP}(\Gamma) \) can be expressed in symmetric Datalog. In particular, we show that if \( \text{CSP}(\Gamma) \) is first-order definable and \( \Lambda \) is a finite subset of the relational clone generated by \( \Gamma \) then \( \neg\text{CSP}(\Lambda) \) is definable in symmetric Datalog. Over the two-element domain and under a standard complexity-theoretic assumption, expressibility of \( \neg\text{CSP}(\Gamma) \) in symmetric Datalog corresponds exactly to the class of CSPs solvable in logarithmic space. Finally, we describe a fairly general subclass of implicational (or 0/1/all) constraints for which the complement of the corresponding CSP is also definable in symmetric Datalog. Our results provide preliminary evidence that symmetric Datalog may be a unifying explanation for families of CSPs lying in \( L \).

1 Introduction

Constraint satisfaction problems (or CSPs) provide a unifying framework to study the complexity of various combinatorial problems arising naturally in optimization, graph theory, artificial intelligence and database theory. Loosely speaking an instance of a CSP consists of a list of variables and a set of constraints each specified by an ordered tuple of variables and a constraint relation over some specified domain. The goal is then to determine whether variables can be assigned domain-values such that all constraints are simultaneously satisfied. The problem is NP-complete in general and one thus seeks to identify restrictions of the problem for which the problem is tractable. Recent efforts have been directed at classifying the complexity of \( \text{CSP}(\Gamma) \), the restriction of the problem in which constraints are all constructed from some set of finitary relations \( \Gamma \) over a finite domain \( D \).

An important conjecture of [10] states that for each constraint language \( \Gamma \) the problem \( \text{CSP}(\Gamma) \) is either tractable (i.e. solvable in polynomial time) or NP-complete. This dichotomy conjecture is a central challenge in theoretical computer science and steady progress towards its establishment has been achieved in the last ten years. It has been verified for domains of size two [18] and three [3]. However, from a complexity-theoretic point of view, such classifications are rather coarse.

*Research supported in part by NSERC, FQRNT and CRM.
as they do not distinguish the relative complexity of CSPs that are tractable. Over the two-element domain, a much finer classification can be obtained: each CSP(Γ) is either in co-NLOGTIME or complete under AC⁰-isomorphisms for one of the classes L (logarithmic space), NL (non-deterministic logspace), ⊕L (parity-counting logspace), P or NP [1].

Efforts to classify the complexity of CSP(Γ) have been greatly facilitated by two complementary approaches. The first relates the complexity of CSP(Γ) with the algebraic properties of the operations which preserve relations in Γ (see e.g. [5] for a thorough introduction). This connection has allowed the use of sophisticated results in universal algebra [4].

A second, descriptive complexity approach relates the complexity of CSP(Γ) with the sophistication of logical frameworks required to describe the class of instances that are not satisfiable. In particular, it has been noticed that a number of tractable cases can be captured by definability of ¬CSP(Γ) in the database-inspired query language Datalog. If, furthermore, ¬CSP(Γ) is definable in linear Datalog then the corresponding problem can be solved in NL and some evidence was given in [7] that this condition is in fact necessary and sufficient.

We introduce symmetric Datalog, a natural syntactic restriction of linear Datalog and prove that its expressivity is exactly that of a specific fragment of symmetric Krom SNP (see [11, 7]). The evaluation of symmetric Datalog programs boils down to a reachability problem in a polynomial-sized symmetric graph. A breakthrough result of Reingold [17] recently established that undirected connectivity can be solved in logarithmic space and, consequently, CSP(Γ) also lies in L if ¬CSP(Γ) can be defined in symmetric Datalog.

We conjecture that expressibility of ¬CSP(Γ) in symmetric Datalog is a necessary and sufficient condition for membership of CSP(Γ) in L. More precisely we provide evidence that a CSP is either expressible in symmetric Datalog or hard for one of a number of complexity classes all believed to strictly contain L. This conjecture is verified over the two element domain. We also show that a specific subset of implicational [13] or 0/1/all constraints [6] can be captured by symmetric Datalog. Finally, we briefly expose the algebraic interpretation of these results. A more thorough discussion appears in [15].

In the next section, we review the foundations of the study of CSP and the connection with Datalog. In section 3 we introduce symmetric Datalog, investigate its relation to restricted symmetric Krom monotone SNP and establish its fundamental properties. Finally, section 4 presents examples of the expressive power of symmetric Datalog and the resulting consequences for the complexity of CSP. Due to space restrictions, some proofs are omitted and appear in the appendix.

2 Constraint Satisfaction Problems and Datalog

Let D be a finite domain. A constraint language over D is a finite¹ set of finitary relations Γ = {R₁, ..., Rₖ} over D. An instance of the constraint satisfaction problem CSP(Γ) is given by a list of variables x₁, ..., xₙ and a set of constraints, where each constraint is of the form (x₁, ..., xₖ) ∈ Rᵢ with Rᵢ ∈ Γ. The task is to determine whether the variables can be assigned values in D such that all constraints are simultaneously satisfied.

It has been noted since the seminal work of Feder and Vardi [10] that the problem can conveniently be recast in terms of homomorphisms between relational structures. Let τ be a finite vocabulary of relational symbols R₁, ..., Rₖ of arity t₁, ..., tₖ. A τ-structure B consists of a set B (called the universe of B) and for each Rᵢ ∈ τ a relation Rᵢ ⊆ Bᵗᵢ. A homomorphism from a τ-structure A to a τ-structure B is a function h : X → B mapping the universe of A to that of B such that for all (a₁, ..., aₖ) ∈ Rᵢ A we have

¹We restrict our attention to the case where Γ is finite although most of the ensuing definitions can be adapted naturally to the case where Γ is infinite.
\((h(a_1), \ldots, h(a_{t_1})) \in R^B_1\).

In its full generality, the homomorphism problem \(\text{Hom}\) is the task of determining whether there exists a homomorphism between two \(\tau\) structures \(A\) and \(B\) given as input. A constraint language \(\Gamma = \{R_1, \ldots, R_k\}\) over \(D\) can be regarded as defining a \(\tau\)-structure \(D\) with universe \(D\) and we denote this structure as a boldface \(\Gamma\). An instance of \(\text{CSP}(\Gamma)\) similarly corresponds to a \(\tau\)-structure over the universe \(x_1, \ldots, x_n\). The program \(\text{CSP}(\Gamma)\) is then equivalent to the problem \(\text{Hom}(\Gamma)\) of determining if a given \(\tau\)-structure is homomorphic to \(\Gamma\).

2.1 Datalog and CSP

Datalog is a database inspired query-language whose connection to the complexity of constraint satisfaction problems has been thoroughly investigated (see e.g. [10, 9, 7]). Let \(\tau\) be some finite vocabulary of relational symbols. A Datalog program over \(\tau\) is specified by a finite set of rules of the form

\[ h \leftarrow b_1; \ldots; b_t \]

where \(h\) and the \(b_i\) are atomic formulas \(R(x_1, \ldots, x_k)\). Note that the variables occurring in a given rule are not assumed to be distinct. We distinguish two types of relational predicates occurring in the program: predicates \(R\) that occur at least once in the head of a rule (i.e. its left-hand side) are called intensional database predicates (IDBs) and are not part of \(\tau\). The other predicates which occur only in the body of a rule (its right-hand side) are called extensional database predicates and must all lie in \(\tau\).

Let \(Q\) be a Datalog program over \(\tau\) and let \(\tau'\) denote the union of \(\tau\) with the IDBs occurring in \(Q\). The program \(Q\) defines a function \(\Phi_Q\) from the set of \(\tau\) structures to the set of \(\tau'\) structures. Intuitively, \(\Phi_Q(A)\) is the smallest \(\tau'\) structure over the same universe as \(A\) with the property that for each rule \(P(\overline{x}) \leftarrow P_1(\overline{y}_1); \ldots; P_h(\overline{y}_k)\) of \(Q\), and any interpretation of the variables the implication defined by the rule is valid.

Formally, for a \(\tau\)-structure \(A\), let \(A^{Q[0]}\) denote the \(\tau'\) structure over the same universe and such that \(R^{A^{Q[0]}} = R^A\) if \(R \in \tau\) and \(R^{A^{Q[0]}} = \emptyset\) if \(R \notin \tau' - \tau\). We now inductively define \(A^{Q[n+1]}\) as follows. First if \(R \in \tau\) then \(R^{A^{Q[n+1]}} = R^{A^{Q[n]}} = R^A\).

Suppose now that \(R \in \tau' - \tau\) is an IDB of arity \(r\). Let \(h \leftarrow b_1; \ldots; b_t\) be a Datalog rule using variables \(x_1, \ldots, x_k\). An interpretation of the rule over the domain \(A\) is a function \(f : \{x_1, \ldots, x_k\} \rightarrow A\). We then define \(R^{A^{Q[n+1]}}\) as the union of \(R^{A^{Q[n]}}\) with all \(r\)-tuples \((a_1, \ldots, a_r)\) such that for some rule with head \(R(x_1, \ldots, x_r)\) and some interpretation \(f\) such that \(f(x_{i_j}) = a_j\) we have for all \(T(x_{j_1}, \ldots, x_{j_q})\) in the body of the rule \((f(x_{j_1}), \ldots, f(x_{j_q})) \in T^{A^{Q[n]}}\).

By definition, \(R^{A^{Q[n]}} \subseteq R^{A^{Q[n+1]}}\) and so the iterative process above is monotone and has a minimal fixed point which we denote as \(R^{A^Q}\). Accordingly, we define \(\Phi_Q(A)\) as the \(\tau'\)-structure given by the \(R^{A^Q}\).

As defined above, the output of a Datalog program is a \(\tau'\)-structure but we want to view a Datalog program \(Q\) primarily as a way to define a class of \(\tau\)-structures. For this purpose, we choose in \(Q\) an IDB \(G\) known as the goal predicate and say that the \(\tau\)-structure \(A\) is accepted by \(Q\) if \(G^{A^Q}\) is non-empty. A class \(C\) is definable in Datalog if there exists a program \(Q\) such that \(A \in C\) iff \(Q\) accepts \(A\). Note that any such \(C\) is homomorphism closed, i.e. if \(h : A \rightarrow B\) is a homomorphism of \(\tau\)-structures and \(A \in C\) then \(B \in C\).

3 Symmetric Datalog

A rule of a Datalog program \(Q\) is said to be linear if its body contains at most one IDB and is said to be non-recursive if its body contains only EDBs. A linear but recursive rule is of the form

\[ I_1(\overline{x}) \leftarrow I_2(\overline{y}); E_1(\overline{x}); \ldots; E_k(\overline{x}) \]
where $I_1, I_2$ are IDBs and the $E_j$ are EDBs. Each such rule has a symmetric rule

$$I_2(\overline{y}) \leftarrow I_1(\overline{x}); E_1(\overline{x_1}); \ldots; E_k(\overline{x_k}).$$

A Datalog program $Q$ is said to be linear if all its rules are linear. We further say that $Q$ is a symmetric Datalog program if the symmetric of any non-recursive rule of $Q$ is also a rule of $Q$.

Let us consider for example the constraint satisfaction problem two-coloring. In this case, the domain is boolean and $\Gamma$ contains a single binary relation $\neq$. Equivalently, an undirected graph is two-colorable if it is isomorphic to an undirected edge. Because a graph is two-colorable iff it contains no undirected cycle of odd length, we can define $\neg\text{CSP}(\Gamma)$ using the following symmetric Datalog program:

$$O(x, y) \leftarrow E(x, y)$$
$$O(x, y) \leftarrow O(x, w); E(w, z); E(z, y)$$
$$O(x, w) \leftarrow O(x, y); E(w, z); E(z, y)$$
$$G \leftarrow O(x, x)$$

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. 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.

**Theorem 1** A symmetric Datalog query $Q$ can be evaluated in logarithmic space. In other words, there exists a logspace transducer which on input $A$ produces some representation of $\Phi_Q(A)$.

**Proof:** Suppose that $I_1, \ldots, I_s$ are the IDBs in $Q$. Let $A$ be the input $\tau$-structure and let $n$ denote the size of its universe $A$. We define the execution graph $G_A$ as follows: for each $I_j$ of arity $k$, we introduce for each of the $n^k$ $k$-tuples of $A^k$ a vertex labeled $I_j(a_1, \ldots, a_k)$. Furthermore, we add an extra vertex labeled $S$. Edges are now determined by the EDBs in $A$ and the rules of $Q$. We add an edge from $I_j(a_1, \ldots, a_k)$ to $I_j'(b_1, \ldots, b_k)$ if $Q$ contains a rule of the form

$$I_j'(\overline{y}) \leftarrow I_j(\overline{x}); E_{s_1}(\overline{x_1}); \ldots; E_{s_n}(\overline{x_n})$$

such that there exists an interpretation $f$ of this rule over $A$ such that $f(\overline{x}) = (b_1, \ldots, b_k)$, $f(\overline{y}) = (a_1, \ldots, a_k)$ and for each $f(\overline{z}_i) \in E_{s_i}^A$. Informally, such an edge represents the fact that if $Q$ places the tuple $(a_1, \ldots, a_k)$ in $I_j$ then it will also add the tuple $(b_1, \ldots, b_k)$ to $I_j'$. We further add a bi-directional edge between our special vertex $S$ and $I_j(a_1, \ldots, a_k)$ if there exists a non-recursive rule

$$I_j(\overline{x}) \leftarrow E_{s_1}(\overline{x_1}); \ldots; E_{s_n}(\overline{x_n})$$

and an interpretation $f$ such that $f(\overline{x}) = (a_1, \ldots, a_k)$ and $f(\overline{z}_i) \in E_{s_i}^A$ for each EDB occurring in the body. Since $Q$ is symmetric, the graph $G_A$ is symmetric and can thus be regarded as an undirected graph. Moreover, the graph can clearly be constructed in logarithmic space and we have $(a_1, \ldots, a_k) \in I_j^A \iff$ the vertex $I_j(a_1, \ldots, a_k)$ is reachable from $S$ in $G_A$. Since undirected connectivity can be computed in logarithmic space [17], a representation of $\Phi_Q(A)$ can be produced in logarithmic space.

**Corollary 2** If $\neg\text{CSP}(\Gamma)$ is definable in symmetric Datalog then $\text{CSP}(\Gamma) \in L$.

Let $\tau$ be a vocabulary consisting of relational symbols. $SNP$ is the class of sentences of the form

$$\exists S_1, \ldots, S_l \forall v_1, \ldots, v_m \varphi(v_1, \ldots, v_m)$$

where $S_1, \ldots, S_l$ are second-order variables and $\varphi$ is a quantifier-free first-order formula

\footnote{Note again that the variables occurring in $\tau, \overline{y}, \overline{z}$ are not necessarily distinct.}
over the vocabulary $\tau \cup \{S_1, \ldots, S_l\}$ with variables among $u_1, \ldots, u_m$. We assume that $\varphi$ is in CNF. In monotone SNP, every occurrence of a relation symbol from $\tau$ is negated. In Krom SNP, every clause of the quantifier-free first-order part $\varphi$ has at most one possible occurrence of a second-order variable and at most one negative occurrence of a second-order variable. Symmetric restricted Krom monotone SNP is the subset of restricted Krom monotone SNP formulae that contain with every clause of the form $\psi \lor S_i \lor \neg S_j$ also the clause $\psi \lor \neg S_i \lor S_j$ (where $\psi$ contains no second-order variables).

We denote as symmetric Datalog($\neg$) the extension of symmetric Datalog programs in which the negation of EDB predicates is allowed. In the appendix, we show how Theorem 5 of [7] can be adapted to obtain the following.

**Theorem 3** Let $C$ be a collection of $\tau$-structures. Then 1 is equivalent to 2, and 3 is equivalent to 4.

1. $C$ is definable in symmetric Datalog;
2. $\neg C$ is definable in symmetric restricted Krom monotone SNP;
3. $C$ is definable in symmetric Datalog($\neg$);
4. $\neg C$ is definable in symmetric restricted Krom SNP.

It can be shown that the expressive power of symmetric Datalog, or equivalently symmetric restricted Krom monotone SNP is quite limited. In particular, there exists no symmetric Datalog program that computes the transitive closure of a binary relation [8].

A finite successor structure is a structure whose domain is $\{0, 1, \ldots, n - 1\}$ (for some $n \in \mathbb{N}$) and whose vocabulary contains the two constant symbols $\text{min}$ and $\text{max}$ and the binary predicate $S$ whose interpretations are the constants 0, $n - 1$ and the successor relation $S = \{(x, x + 1)|x < n - 1\}$. $\mathcal{O}$ denotes the set of all finite successor structures.

A logic captures the complexity class $C$ if for every problem $P \subseteq \mathcal{O}$, $P$ is in $C$ if and only if there exists a formula $\psi$ in the logic such that $P = \{R \in \mathcal{O}|R \models \psi\}$. Using the results of [11], one can show:

**Theorem 4** Over the set of finite successor structures, symmetric Datalog($\neg$) captures $L$.

We give a proof in the appendix. Similarly, Datalog($\neg$) and linear Datalog($\neg$) capture $P$ and NL respectively (see [16, 7, 11] and comments following the proof of Theorem 4).

**4 Applications**

We consider in this section a number of specific classes of constraint languages $\Gamma$ for which $\neg \text{CSP}(\Gamma)$ is expressible in symmetric Datalog. This includes, to the best of our knowledge, all families of CSP known to lie in logspace, thus providing preliminary evidence that symmetric Datalog is a unifying explanation for logspace computable CSP.

For a constraint language $\Gamma$, we denote as $\langle \Gamma \rangle$ the relational clone generated by $\Gamma$ (also known as the primitive positive closure of $\Gamma$) i.e. the set of relations which can be defined by a primitive positive formula over $\Gamma$ and the equality relation. For any finite $\Lambda \subseteq \langle \Gamma \rangle$ there exists a polynomial time reduction from CSP($\Lambda$) to CSP($\langle \Gamma \rangle$) and this fairly straightforward observation is crucial when using algebraic tools to study the complexity of CSP [5].

In some particularly simple cases $\neg \text{CSP}(\Gamma)$ is in fact definable by a Datalog program without any recursive rule. These FO-definable CSP have been completely characterized [2, 14]. All such problems can be solved in $\text{CO-NLOGTIME}$, a class provably strictly contained in logspace (see [1] for a thorough discussion). It has been noted that over any non-trivial domain there exist constraint languages $\Gamma, \Lambda$ such that $\langle \Gamma \rangle = \langle \Lambda \rangle$ but CSP($\Gamma$) is FO-definable while CSP($\Lambda$) is not. Our first result
however guarantees that \( \neg \text{CSP}(\Lambda) \) is still definable in symmetric Datalog.

**Theorem 5** If \( \Gamma \) is a constraint language such that \( \neg \text{CSP}(\Gamma) \) is definable in symmetric Datalog and \( \Lambda \) is a finite subset of \( \Gamma \) then \( \neg \text{CSP}(\Lambda) \) is definable in symmetric Datalog. In particular, if \( \text{CSP}(\Gamma) \) is first-order definable then \( \neg \text{CSP}(\Lambda) \) is definable in symmetric Datalog, and furthermore, \( \text{CSP}(\Lambda) \) is either first-order definable itself or \( \text{L-complete} \).

**Proof:** The theorem relies on the work of two of the authors. Since \( \Lambda \) is a finite subset of \( \Gamma \) then it can be obtained from the set \( \Gamma \) by a finite sequence of applications of six basic constructions, five of which are shown in [15] to preserve expressibility in symmetric Datalog. It remains to show that if \( \neg \text{CSP}(\Gamma) \) is expressible in symmetric Datalog then so is \( \neg \text{CSP}(\Gamma \cup \{=\}) \). We present this argument as Lemma 11 in the appendix.

If \( \text{CSP}(\Gamma) \) is first-order definable then \( \neg \text{CSP}(\Gamma) \) is certainly expressible in symmetric Datalog, hence so is \( \neg \text{CSP}(\Lambda) \). The second part also follows from a result of [15]: if a CSP is not FO expressible then it is Logspace-hard (under FO reductions).

Dalmau showed in [7] that constraint satisfaction problems defined by *implicational constraints* [13] (known in [6] as *0/1/all constraints*) are definable in linear Datalog. A binary relation \( R \subseteq D^2 \) is said to be implicational if it is of one of three forms:

1. \( R = B \times C \) for some \( B, C \subseteq D \);
2. \( R = \{(b, f(b)) : b \in B\} \) where \( B \subseteq D \) and \( f \) is an injective function;
3. \( R = \{b\} \times C \cup B \times \{c\} \) for some \( B, C \subseteq D \) with \( b \in B \) and \( c \in C \).

Note that the relation \( \leq \) over the two-element domain is implicational since it is \( \{0\} \times \{0, 1\} \cup \{0, 1\} \times \{1\} \). Furthermore \( \{0\} \times \{1\} \) is also implicational and it is easy to see that \( \text{CSP}(\{\leq, \{0\} \times \{1\}\}) \) is \( \text{NL-complete} \) and in fact \( \neg \text{CSP}(\{\leq, \{0\} \times \{1\}\}) \) is not expressible in symmetric Datalog [8]. The difficulty in fact stems from implicational constraints of the third form and we use the following lemma to show that for any \( \Gamma \) consisting solely of implicational constraints of the first two forms, \( \neg \text{CSP}(\Gamma) \) is definable in symmetric Datalog.

**Theorem 6** Let \( \Gamma \) be a constraint language over the domain \( D \) such that for some \( d \in D \), every relation \( R \) in \( \Gamma \) is either

1. a unary relation \( S \subseteq D \);
2. a binary relation \( R_{a} \) such that \( R_{\pi} = \{(a, \pi(a)) : a \in D\} \) for some permutation \( \pi \) with \( \pi(d) = d \);
3. a \( k \)-ary relation \( R_{k} \) with \( k \geq 2 \) and \( R_{k} = \{(a_1, \ldots, a_k) : \exists i \ a_i = d\} \).

Then \( \neg \text{CSP}(\Gamma) \) is definable in symmetric Datalog. The result also holds if the permutations have no fixed point \( d \) but \( \Gamma \) does not contain any relation of the third form above.

**Proof:** Let us assume that the permutations have a fixed point \( d \) (see comment following the proof) and that \( \Gamma \) contains relations of the form \( R_{k} = \{(a_1, \ldots, a_k) : \exists i \ a_i = d\} \). Before describing the symmetric Datalog program, we make a few basic observations and recast the problem in a more graph-theoretic fashion. First note that if \( j \leq k \), then the relation \( R_j \{(a_1, \ldots, a_j) : \exists i \ a_i = d\} \) is simply the set of tuples such that \( (a_1, \ldots, a_j, \ldots, a_j) \in R_k \). By Theorem 5, we can thus assume without loss of generality that \( \Gamma \) contains a single relation \( R_k \). Next, we can also assume that \( \Gamma \) contains all unary relations over \( D \), i.e. for any \( S \subseteq D \), \( \Gamma \) contains the unary relation \( U_S = S \). If \( A \) is an input \( \tau \)-structure and \( \phi : A \rightarrow \Gamma \) is a homomorphism, then for each \( x \in A \) we must have \( \phi(x) \in \bigcap_{x \in U_S} S \). For convenience, we denote by \( U_x \) this intersection which corresponds to the subset of values for \( x \) in \( D \) which respect the unary constraints imposed on \( x \).
For a $\tau$-structure $A$, we construct an edge-labeled and vertex-labeled directed graph $G = (V, E)$ as follows. Vertices are the elements of the universe of $A$. For any $x, y$ such that $(x, y) \in R_\pi$ we add an edge $(x, y)$ labeled with $\pi$ and an edge $(y, x)$ labeled with $\pi^{-1}$. Finally we color the vertex $x$ with $U_x$. Any path $p$ between vertices $x$ and $y$ can be thought of as labeled by the permutation $\pi_p$ which is the product of the labels of individual edges on that path.

Let $x$ be a vertex of the graph with $U_x = \{a_1, \ldots, a_t\}$. We say that a set of $t$ paths $p_1, \ldots, p_t$ from $x$ to vertices $y_1, \ldots, y_t$ in $G$ is permutation-blocking for the vertex $x$ if for each $1 \leq i \leq t$ we have either $\pi_{p_i}(a_i) \not\in U_{y_i}$ or $x = y_i$ and $\pi_{p_i}(a_i) \not= a_i$. By design, if $G$ contains a permutation-blocking pattern then there can be no homomorphism $\phi$ from $A$ to $\Gamma$. Indeed, for any such $\phi$, we require $\phi(x) \in U_x$. However, if $\phi(x) = a_i$ then we must also have $\phi(y_i) = \pi_{p_i}(a_i) \not\in U_{y_i}$.

Similarly, for any $(x_1, \ldots, x_k) \in R^A_k$ we say that the $k$-tuple $(y_1, \ldots, y_k)$ is a $k$-blocking pattern in $G$ if for each $1 \leq i \leq k$ there is a path from $x_i$ to $y_i$ and $d \not\in U_{y_i}$. Again, if a $k$-blocking pattern exists for some $(x_1, \ldots, x_k) \in R^A_k$ then there is no homomorphism from $A$ to $\Gamma$. Since none of the $y_i$ can be mapped to $d$ and $d$ is a fixed point of all permutations $\pi$, it follows that none of the $x_i$ can be mapped to $d$ and so $(\phi(x_1), \ldots, \phi(x_k)) \not\in R_k$.

We claim that $A \in \text{CSP}(\Gamma)$ if and only if $G$ contains no permutation-blocking or $k$-blocking patterns. Note that we have already established the left to right implication. Let us suppose that $G$ contains no blocking patterns and explicitly construct a homomorphism from $A$ to $\Gamma$. Consider an arbitrary element $x$ of $A$. If $d \in U_y$ for all $y$ in the connected component (in $G$) of $x$, then we set $\phi(x) = d$. Otherwise, since $G$ contains no permutation-blocking pattern, there exists some $a \in U_x$ such that for each path $p$ from $x$ to $y$ either $x = y$ and $\pi_p(a) = a$ or $x \not= y$ and $\pi_p(a) \not\in U_y$. We set $\phi(x) = a$ and $\phi(y) = \pi_p(a)$ for each vertex $y$ reachable from $x$ by a path $p$. Note that this assignment is well defined: if there are two (or more) distinct paths $p, p'$ from $x$ to a given $y$ then $\pi_{pp'}^{-1}(a) = a$ which means that $\pi_{p}(a) = \pi_{p'}(a)$.

This stage defines $\phi(y)$ for each $y$ in the connected component of $x$. We can repeat the argument to similarly fix the values of $\phi$ in each other connected component of $G$. We need to prove that $\phi$ is indeed a homomorphism. By construction, we have a homomorphism $\phi(x) \in U_x$ for each $(x, y) \in R_\pi$. It remains to show that if $(x_1, \ldots, x_k) \in R^A_k$ then $(\phi(x_1), \ldots, \phi(x_k)) \in R_k$ which, by definition of $R_k$ is equivalent to the requirement that $\phi(x_j) = d$ for some $j$. Since $G$ contains no $k$-blocking pattern, there exists some $x_j$ such that all $y$ in the connected component of $x_j$ satisfy $d \in U_{y}$ and so $\phi(x_j) = d$.

Our symmetric Datalog program contains rules of one of eight types summarized in Figure 1. The program is a simple reflection of the above graph-theoretic construction. For each $|D|$-tuple $(\pi_1, \ldots, \pi_{|D|})$ of permutations of $D$, we create a $|D| + 1$-ary IDB $I_{\pi_1,\ldots,\pi_{|D|}}(x, y_1, \ldots, y_{|D|+1})$ which is intended to represent the fact for each $1 \leq j \leq |D|$ there is a path $p$ labeled with $\pi_j$ from $x$ to $y_j$. We include non-recursive initialization rules stating that if $A$ includes for each $j$ the constraint $(x, y_j) \in R_{\pi_j}$ then the tuple $(x, y_1, \ldots, y_{|D|+1})$ lies in the IDB $I_{\pi_1,\ldots,\pi_{|D|}}$.

We next include recursive rules for these IDBs as follows. Consider two IDBs $I_{\pi_1,\ldots,\rho_{\pi_j},\ldots,\pi_{|D|}}$ and $I_{\pi_1,\ldots,\pi_j,\ldots,\pi_{|D|}}$ whose index differs only in the $j$th permutation. If $x, y_1, \ldots, y_{|D|}$ are such that there exist for each $i$ an $x \leadsto y_i$ path labeled by $\pi_i$ and if further $A$ contains a constraint $(y_j, z) \in R_{\pi_j}$ then there is a path $x \leadsto z$ labeled $\rho_{\pi_j}$. But symmetrically, if there is a $x \leadsto z$ labeled by $\rho_{\pi_j}$, we have a path $x \leadsto y_j$ labeled by $\rho^{-1}\rho_{\pi_j} = \pi_j$. This observation justifies the rules of the form (2) and (3) in our program and it is clear that these IDBs behave as intended.

We use an IDB $J(x_1, \ldots, x_k, y_1, \ldots, y_k)$ of arity $2k$ to represent the fact that for each $i$
there is a path (regardless of labels) from $x_i$ to $y_i$. Clearly, we can initialize this IDB with the non-recursive rule (4). Note that we have one such rule for any choice of $R_\pi$, since we do not care about the actual label of the paths. Similarly, we have recursive rules of the form (5) and (6) for each $1 \leq j \leq k$ and every $R_\pi$.

Finally our program contains a unary goal predicate $G$ which is hit whenever the program has detected a permutation-blocking or $k$-blocking pattern. There exists a $k$-blocking pattern iff there exists a tuple $(x_1, \ldots, x_k, y_1, \ldots, y_k)$ in $J$ and unary relations $U_1, \ldots, U_k$ such that $d \not\in U_j$ and $(x_1, \ldots, x_k) \in R_k^A$ (rules of type (8)). Note that the program contains one such rule for any choice of the unary relations $U_1, \ldots, U_k$ that do not contain $d$.

Permutation blocking patterns are identified using rules of type (7). We consider a set of paths from some $x$ to some $y_1, \ldots, y_k$ with labels $\pi_1, \ldots, \pi_{|D|}$. Hence the body of the rule contains $I_{\pi_1, \ldots, \pi_{|D|}}(x, y_1, \ldots, y_k)$. The fact that these form a permutation-blocking pattern now depends solely on the set of unary relations that bound the variables $x, y_1, \ldots, y_{|D|}$. Our program creates a separate rule of type (7) to handle each combination of unary relations imposed on $x, y_1, \ldots, y_{|D|}$ which result in a blocking pattern. Note that there is a fixed bound on the number of ways in which these $|D| + 1$ variables can be constrained by unary relations. For succinctness, we represented these rules in Figure 1 by using the symbols $U_x$, $U_{y_1}$, etc. to indicate that the body contains a conjunction of unary EDBs constraining these variables in a way that creates a blocking pattern.

We assumed in our proof that $d$ was a fixed point of the permutations. However, it is clear from the argument that this requirement is only needed in the presence of the $R_k$ relations. If $\Gamma$ consists solely of permutations and unary relations, $\neg\text{CSP}(\Gamma)$ can be defined in symmetric Datalog. However, if $\Gamma$ contains a relation $R_k$ but contains a permutation of which $d$ is not a fixed point expressibility in symmetric Datalog cannot be guaranteed. Indeed, over the two-element domain, the relation $R_2$ is the binary OR relation and the non-trivial permutation $\pi$ is disequality. The problem CSP($O_2$, $\neq$) is NL-hard and $\neg\text{CSP}(O_2, \neq)$ does not lie in symmetric Datalog [8].

This theorem immediately provides the following result for implicational constraints.

**Corollary 7** Let $\Gamma$ be a finite set of implicational constraints of the form 1 and 2. Then $\neg\text{CSP}(\Gamma)$ is expressible in symmetric Datalog.

**Proof:** Obviously, a binary relation of the form $B \times C$ can be expressed as the conjunction of the unary relations $U_B$ and $U_C$. Similarly, if $B \subseteq D$ and $f : B \rightarrow D$ is injective then $f$ can be extended to a permutation $\pi$ of $D$ such that $\pi|_B = f$. The implicational relation $R = \{(b, f(b)) : b \in B\}$ can then be expressed as a conjunction of the relation $R_\pi$ and the unary relation $B$. Thus, $\Gamma \subseteq (A)$ for some $A$ of the
form given in Theorem 6 and the result follows by Theorem 5.

Over the two-element domain, there is a very tight correspondence between CSP in logspace and symmetric Datalog.

**Theorem 8** Let $\Gamma$ be a constraint language over the two-element domain. Then $\neg \text{CSP}(\Gamma)$ is definable in symmetric Datalog or $\text{CSP}(\Gamma)$ is hard for NL or $\oplus L$ under logspace reductions.

**Proof:** By [1], we know that CSP$(\Gamma)$ is hard for one of NL or $\oplus L$ unless $\Gamma$ is contained in the relational clone generated by the unary relations, the equality relation and the disequality relation. By Theorem 6, we have $\neg \text{CSP}\{\{0\},\{1\},=,\neq\}$ is definable in symmetric Datalog and this expressibility results extends to all $\Gamma$ in the relational clone by Theorem 5.

If one accepts the hypothesis that NL $\neq L$ and $\oplus L \neq L$, we thus have, over the two-element domain, that CSP$(\Gamma)$ is in logarithmic space iff $\neg \text{CSP}(\Gamma)$ is in symmetric Datalog.

We conclude with a brief discussion on the relationship between our results and the algebraic approach to CSP mentioned earlier.

To each set of relations $\Gamma$ on a set $A$ is associated an algebra $\mathcal{A}(\Gamma)$ with universe $A$ whose basic operations are the operations that preserve all relations in $\Gamma$, i.e. the functions $f : A^k \rightarrow A$ such that for any $t$-ary $R \in \Gamma$, and any $k$ $t$-tuples $\pi_1, \ldots, \pi_k \in R$ it holds that $f(\pi_1, \ldots, \pi_k) \in R$. It is known that whether the problem CSP$(\Gamma)$ is polynomially solvable or NP-hard is determined by the equational properties of this algebra (see e.g. [5]). In [15], this approach is refined, and general hardness results for CSPs are presented for various complexity classes such as L, NL, P and Mod$_pL$ and some necessary algebraic conditions are described for expressibility in various restrictions of Datalog. In particular, if $\neg \text{CSP}(\Gamma)$ is expressible in symmetric Datalog, then the minimal non-trivial factors of the associated algebra must be of a very specific form (in universal algebra lingo, this algebra must generate a variety admitting only the Boolean type). As a special case of Theorem 6, we obtain that all CSPs whose associated algebra is one of these so-called strictly simple algebras have their complement expressible in symmetric Datalog.

**Theorem 9** Let $\Gamma$ be a finite set of relations on the set $A$ such that the algebra $\mathcal{A}(\Gamma)$ is an idempotent, strictly simple algebra of Boolean type. Then $\neg \text{CSP}(\Gamma)$ is expressible in symmetric Datalog.

**Proof:** We invoke a classification of idempotent, strictly simple algebras of Boolean type (see Theorem 6.1 of [20]) to get a precise description of the algebra $\mathcal{A}(\Gamma)$: either it is quasiprimal, and hence it admits the discriminator $t$ as a basic operation, where

$$t(x, y, z) = \begin{cases} z & \text{if } x = y, \\ x & \text{else}, \end{cases}$$

or there exists some $d \in A$, and $G$ a group of permutations of $A$ such that $d$ is the unique fixed point of every non-identity element in $G$, and such that the set of basic operations of $\mathcal{A}(\Gamma)$ is equal to $F_k$ for some $2 \leq k \leq \omega$, where $F_k$ consists of all idempotent operations that preserve the relations in $G^0 = \{(b, f(b)) : \pi \in G\}$ and the relations $R_k$ defined in the statement of Theorem 6. $F_{\omega}$ in the intersection of all the $F_k$.

Accordingly we split the proof in two cases:

**Case 1.** Suppose that $\mathcal{A}(\Gamma)$ is quasiprimal, and thus $t$ preserves every relation in $\Gamma$. It follows from standard results in universal algebra that since every relation in $\Gamma$ is invariant under the discriminator $t$, each is determined by its projections on at most two factors. In other words, we obtain that $\Gamma \subseteq \langle \Gamma' \rangle$ where $\Gamma'$ consists of all projections on at most two factors of all the relations in $\Gamma$. By Theorem 5, it now suffices to prove that $\neg \text{CSP}(\Gamma')$ is in symmetric Datalog. By Theorem 4.2 of [19], every binary

---

$^3$Mod$_pL$ is the class of problems which are logspace reducible to solving systems of linear equations over $\mathbb{Z}_p$. 

---

9
relation invariant under the discriminator is either a product of two unary relations or is the form $\pi^o = \{ (x, \pi(x) : x \in B_\pi) \}$ where $B_\pi$ is some non-empty subset of $A$ and $\pi$ is some injective map from $B_\pi$ into $A$. This case is precisely covered by Corollary 7.

Case 2. Suppose now that the set of basic operations of $\mathcal{A}(\Gamma)$ is $F_k$. For any $k$ we have $\Gamma \subseteq \{ (G^o, R_2, \ldots, R_j, \ldots) \}$; however, since $\Gamma$ is finite and since each relation in it is produced from finitely many of the $R_j$, it follows that there exists some finite $\ell$ such that $\Gamma \subseteq (G^o R_\ell)$ and by Theorem 6, $\neg \text{CSP}(G^o, R_\ell)$ is expressible in symmetric Datalog.

In light of this result and Theorem 8, it is tempting to conjecture an analog of Theorem 8 for non-Boolean domains. By results in [15, 8], it remains to show that if the variety generated by $\mathcal{A}(\Gamma)$ admits only the Boolean type then $\neg \text{CSP}(\Gamma)$ is definable in symmetric Datalog.

Acknowledgements

We wish to thank Albert Atserias, Victor Dalmau and Heribert Vollmer for useful discussions. Some of the work presented here was developed during the Dagstuhl Seminar on the Complexity of Constraints.

References

Appendix

This appendix includes the proofs of Theorems 3 and 4 as well as a lemma required in our proof of Theorem 5.

**Theorem 3** Let $C$ be a collection of $\tau$-structures. Then 1 is equivalent to 2, and 3 is equivalent to 4.

1. $C$ is definable in symmetric Datalog;
2. $\neg C$ is definable in symmetric restricted Krom monotone SNP;
3. $C$ is definable in symmetric Datalog($\neg$);
4. $\neg C$ is definable in symmetric restricted Krom SNP.

We use the following lemma.

**Lemma 10** Let $Q$ be a Datalog($\neg$) program over $\tau$ with IDBs $I_1, \ldots, I_m$. Let

$$
\psi(I_1, \ldots, I_m, x_1, \ldots, x_n) = \bigwedge_{h \leftarrow (b_1 \land \ldots \land b_q)} h \leftarrow (b_1 \land \ldots \land b_q).
$$

Let $A$ be a $\tau$-structure such that there exist relations $R_1, \ldots, R_m$ such that

$$
A, R_1, \ldots, R_m \models \forall x_1, \ldots, x_n \psi(I_1, \ldots, I_m, x_1, \ldots, x_n).
$$

Then $I^{A_Q}(t) \rightarrow R_i(t)$ for each $i, 1 \leq i \leq m$ and each $t \in A^r$ where $r$ is the arity of $I_i$.

**Notation:** For convenience, we write $I^{A_Q(i)}_i \rightarrow R_i$ to denote that for each tuple $t \in A^r$ where $r$ is the arity of $I_i$, $I^{A_Q(i)}_i(t) \rightarrow R_i$. We write $I^{A_Q} \rightarrow R$ to denote that for each $i, 1 \leq i \leq m$ and for each tuple $t \in A^r$ where $r$ is the arity of $I_i$, $I^{A_Q(i)}_i(t) \rightarrow R_i(t)$.

**Proof:** Consider the sequence $A^{Q[0]}, A^{Q[1]}, \ldots, A^{Q}$ and for the sake of contradiction assume that for some $i$ and for some tuple $t' \in A^r$, $I^{A_Q}(t')$ is true but $R_i(t')$ is false. Then there exists a smallest $j$, an index $k$, and a tuple $t \in A^r$ such that $I^{A_Q[j]}_k(t)$ is true but $R_k(t)$ is false.

Let $I_k(x_{k_1}, \ldots, x_{k_r}) \leftarrow b_1; \ldots; b_q$ be the rule which added $t$ to $I^{A_Q[j]}_k$ using the interpretation $f : x_1, \ldots, x_n \rightarrow A$. Notice that $t = (f(x_{k_1}), \ldots, f(x_{k_r}))$. By the choice of $j$, $I^{A_Q[j-1]} \rightarrow R$. Therefore $A, R_1, \ldots, R_m, f(x_1), \ldots, f(x_n) \models b_1 \land \ldots \land b_q$ and $b_1 \land \ldots \land b_q \rightarrow R_k(f(x_{k_1}), \ldots, f(x_{k_r}))$ in $\psi$. This implies that $R_k(t)$ is true which is a contradiction.

---

**Proof:** [Proof of Theorem 3] We prove the equivalence of 1 and 2. The equivalence of 3 and 4 is an obvious modification of the proof.

1 $\rightarrow$ 2: Let $Q$ be a symmetric Datalog program with IDB predicates $I_1, \ldots, I_m$ one of which is the goal predicate $I_g$. Let $\psi$ be a CNF formula defined as:

$$
\psi(I_1, \ldots, I_m, x_1, \ldots, x_n) = \\
\neg I_g \land \bigwedge_{h \leftarrow (b_1 \land \ldots \land b_q)} h \leftarrow (b_1 \land \ldots \land b_q) \\
\neg I_g \land \bigwedge_{h \leftarrow (b_1 \land \ldots \land b_q)} h \lor \neg b_1 \lor \ldots \lor \neg b_q.
$$

Let $\phi$ be the following symmetric restricted Krom monotone SNP sentence

$$
\phi = \exists R_1, \ldots, R_m \forall x_1, \ldots, x_n \psi(I_1, \ldots, I_m, x_1, \ldots, x_n).
$$

We show that $\phi$ is satisfied exactly by those structures that are rejected by $Q$. Assume that $Q$ rejects $A$. Then clearly, $A, I^{A_Q[1]}_1, \ldots, I^{A_Q}_m \models \forall x_1, \ldots, x_n \psi(I_1, \ldots, I_m)$ so $A \models \phi$.

Conversely, assume that $A$ is a structure such that $A \models \phi$. Then there exist relations $R_1, \ldots, R_m$ such that $A, R_1, \ldots, R_m \models \forall x_1, \ldots, x_n \psi(I_1, \ldots, I_m, x_1, \ldots, x_n)$. By Lemma 10, $I^{A_Q} \rightarrow R$ and in particular $I^{A_Q}_g \rightarrow R_g$. Therefore because $R_g$ is false $I^{A_Q}_g$ is also false, i.e. $Q$ rejects $A$.  

---

11
2 → 1: Let \( \phi = \exists I_1, \ldots, I_m \forall x_1, \ldots, x_n \psi(I_1, \ldots, I_m, x_1, \ldots, x_n) \) be an arbitrary sentence in symmetric restricted Krom monotone SNP. We rewrite \( \phi \) in an equivalent "implicational" form which we call \( \phi' \) and the modified \( \psi \) becomes \( \psi' \). Parallel to this, we construct a symmetric Datalog program \( Q \) as follows:

1. Add a new second order variable \( I_{m+1} \) to the existential quantifier block of \( \phi \) and a new clause to \( \psi \), \( (I_{m+1} = \text{False}) \). The IDBs of \( Q \) are \( I_1, \ldots, I_{m+1} \) and \( I_{m+1} \) is the goal predicate. Let \( C \) be a clause of \( \psi \) (but not the newly added clause);

2. If \( C = h \lor b_1 \lor \ldots \lor b_q \) where \( h, b_1, \ldots, b_q \) are literals and \( C \) contains a non-negated second order variable which we denoted by \( h \) then rewrite the \( C \) as \( h \leftarrow (\neg b_1 \land \ldots \land \neg b_q) \). Add the rule \( h \leftarrow b_1; \ldots; b_q \) to \( Q \) in which the IDBs are the second order variables of \( C \);

3. If \( C = b_1 \lor \ldots \lor b_q \) where \( C \) does not contain a non-negated second order variable then rewrite \( C \) as \( I_{m+1} \leftarrow (\neg b_1 \land \ldots \land \neg b_q) \). Add \( I_{m+1} \leftarrow -b_1; \ldots; -b_q \) to \( Q \).

Observe that \( Q \) is a symmetric. We show that \( Q \) accepts exactly the same set of structures which falsify \( \phi' \). Assume that \( Q \) rejects \( A \). Then clearly, \( A, I_1^{AQ}, \ldots, I_{m+1}^{AQ} \models \forall x_1, \ldots, x_n \psi'(I_1, \ldots, I_m, x_1, \ldots, x_n) \), so \( A \models \phi' \).

Conversely, assume that \( A \) is a structure such that \( A \models \phi' \). Then there exist relations \( R_1, \ldots, R_{m+1} \) such that \( A, R_1, \ldots, R_{m+1} \models \forall x_1, \ldots, x_n \psi'(I_1, \ldots, I_m, x_1, \ldots, x_n) \). By Lemma 10, \( I_i^{AQ} \to R_i \), \( 1 \leq i \leq m+1 \). In particular \( I_{m+1}^{AQ} \to R_{m+1} \) where \( R_{m+1} \) is forced to be false. Therefore \( I_{m+1}^{AQ} \) is false, i.e. \( Q \) rejects \( A \).

**Theorem 4** Over the set of finite successor structures symmetric Datalog(\( \neg \)) captures \( L \).

**Proof**: It follows from [12] and [17] that over the set of finite successor structures \( \text{STC}_{\bar{x}, \bar{y}}(\text{\bar{x}}, \bar{y}) = (\text{min}, \text{max}) \) captures \( L \). Here \( \psi \) is a quantifier-free first-order formula and \( \text{STC}_{\bar{x}, \bar{y}}(\bar{x}, \bar{y}) \) denotes the reflexive, symmetric and transitive closure of the binary relation defined by \( \psi \). Now we use an idea from Theorem 6.4 in [11].

If \( P \) is a problem in \( L \) then the complement of \( P \) can be defined by a formula \( \phi = \neg \text{STC}_{\bar{x}, \bar{y}}(\bar{x}, \bar{y}) = (\text{min}, \text{max}) \) where \( \psi \) is quantifier-free. Let \( \forall \psi_i \) be the disjunctive normal form of \( \psi \) and build the formula:

\[
\exists R \forall \bar{x}, \bar{y}, \bar{z}, R(\bar{x}, \bar{y}) \land \\
\bigwedge_i (\neg \psi_i(\bar{y}, \bar{z}) \lor R(\bar{x}, \bar{y}) \leftrightarrow R(\bar{x}, \bar{z})) \land \\
\neg R(\text{min}, \text{max})
\]

This formula is equivalent to \( \phi \) and it can be rewritten as a symmetric restricted Krom SNP formula:

\[
\exists R \forall \bar{x}, \bar{y}, \bar{z}, R(\bar{x}, \bar{y}) \land \\
\bigwedge_i (\neg \psi_i(\bar{y}, \bar{z}) \lor R(\bar{x}, \bar{y}) \lor R(\bar{x}, \bar{z})) \land \\
\neg R(\text{min}, \text{max})
\]

Now we use Theorem 3 to define \( P \) in symmetric Datalog(\( \neg \)).

Conversely, a symmetric Datalog(\( \neg \)) can be evaluated in \( L \) by a simple extension of Theorem 1.

**Comments**: A similar argument can be used using transitive closure instead of symmetric transitive closure to show that linear Datalog(\( \neg \)) over the set of finite successor structures captures \( NL \).

Finally, the following lemma is needed to complete the proof of Theorem 5. We state a slighter more general result.

**Lemma 11** Let \( \Gamma \) be a finite set of relations such that \( \neg \text{CSP}(\Gamma) \) is expressible in (linear,
symmetric) Datalog. Then \( \neg\text{CSP}(\Gamma \cup \{=\}) \) is also expressible in (linear, symmetric) Datalog respectively.

**Proof:** We proceed by modifying a program for the first problem to obtain a program for the second. Let \( P \) be a (linear, symmetric) Datalog program for \( \neg\text{CSP}(\Gamma) \) with goal predicate \( G \). Note that we may modify the program \( P \) as follows without loss of generality: for every rule \( \rho \) of \( P \) of the form

\[
G \leftarrow R_1, \ldots, R_n
\]

where each \( R_i \) is an EDB, add the following rules to \( P \):

\[
\text{I}_\rho \leftarrow R_1, \ldots, R_n
\]

\[
G \leftarrow \text{I}_\rho
\]

where \( I_\rho \) is a new IDB whose variables are exactly those appearing in the \( R_i \). For example, if \( P \) contains the following rule \( \rho \)

\[
G \leftarrow R_1(x, y, z);
R_2(x, w, t); R_3(x, z, t)
\]

then add to \( P \) the rules

\[
\text{I}_\rho(x, y, z, w, t) \leftarrow R_1(x, y, z);
R_2(x, w, t); R_3(x, z, t)
\]

\[
G \leftarrow \text{I}_\rho(x, y, z, w, t).
\]

Finally, remove rule \( \rho \) from \( P \). It is clear that the set of structures accepted by the new program remains unchanged.

We define a program \( Q \) as follows:

(i) For every rule of \( P \) with IDB’s \( I, J_1, \ldots, J_s \) and EDB’s \( R_1, \ldots, R_n \)

\[
I \leftarrow J_1, \ldots, J_s, R_1, \ldots, R_n
\]

\( Q \) will have the rule

\[
\widehat{I} \leftarrow \widehat{J}_1, \ldots, \widehat{J}_s, R_1, \ldots, R_n;
\]

(ii) Let \( E \) denote the relational symbol (i.e. EDB) that corresponds to the equality relation.

For every IDB \( I \) of arity \( k \) the program \( P \), and every \( 1 \leq i \leq k \), \( Q \) has the rules

\[
\widehat{I}(x_1, \ldots, x_k) \leftarrow \widehat{I}(x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots, x_k); x_iEy_i
\]

and

\[
\widehat{I}(x_1, \ldots, x_k) \leftarrow \widehat{I}(x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots, x_k); y_iEx_i.
\]

Notice that the program \( Q \) we have obtained is indeed linear (symmetric) if the program \( P \) is. We shall show that \( Q \) decides \( \neg\text{CSP}(\Gamma \cup \{=\}) \) correctly. For this we use a simple reduction of \( \neg\text{CSP}(\Gamma \cup \{=\}) \) to \( \neg\text{CSP}(\Gamma) \). Let \( T \) and \( T' \) denote the “target” structures for these problems, i.e. whose base set is \( A \) and whose basic relations are those of \( \Gamma \) and \( \Gamma \cup \{=\} \) respectively. Given an input \( S' \) for \( \text{CSP}(T') \), construct an input \( S \) for \( \text{CSP}(T) \) as follows: let \( \theta \) denote the partition of the base set of \( S' \) (which is also the base set of \( S \)) into connected components of the relation \( E \): now for any basic relation \( R \) in \( \Gamma \), let \( (x_1, \ldots, x_k) \) be in \( R^S \) whenever there exists \( (x'_1, \ldots, x'_k) \) in \( R^{S'} \) such that \( x_i \) and \( x'_i \) are in the same \( E \)-block, for every \( 1 \leq i \leq k \). We claim that \( S \) admits a homomorphism to \( T \) if and only if \( S' \) admits a homomorphism to \( T' \). One direction is trivial. Now suppose that \( f : S \rightarrow T \) is a homomorphism, and let \( x \) and \( y \) be distinct elements of some \( E \)-block. Let \( f' \) be the function obtained from \( f \) by setting \( f'(x) = f(y) \) and \( f'(t) = f(t) \) for all \( t \neq x \). By construction of \( R^S \), if a tuple of the relation \( R^S \) has an occurrence of \( x \) then the tuple obtained by replacing all occurrences of \( x \) by \( y \) is also in \( R^S \), and hence obviously \( f' \) is also a homomorphism. Hence if there is a homomorphism from \( S \) to \( T \) there is one which is constant on every \( E \)-block; this is obviously a homomorphism from \( S' \) to \( T' \).

For any IDB \( K \) (in any of the two programs), and any input structure \( A \) to the program, recall that we denote as \( K^{A^{Q(t)}} \) the content of this IDB after \( t \) steps of Datalog evaluation and denote as \( K^{A^S} \) the content of this IDB at the end.
of the run of the program, i.e. when the contents stabilize. When the input structure is clear we shall simply write $K^Q[t]$ and $K^Q$.

**Claim.** If $S$ and $S'$ are the above structures, then for every IDB $I$ of the program $P$, $f^{S^Q[I]} = f^{S'^Q}$.

Proving this claim completes the proof of our lemma: indeed, from the claim it follows immediately that $Q$ accepts $S$ if and only if $P$ accepts $S'$, which occurs if and only if there is a homomorphism from $S'$ to $T'$ if and only if there is a homomorphism from $S$ to $T$.

**Proof of Claim.** Let $e$ denote the reflexive, symmetric, transitive closure of the relation $E$ on $S$. We say that a $k$-ary relation $\theta$ on the base set of $S$ is $e$-closed if $(x_1, \ldots, x_k) \in \theta$ whenever there exists some $(y_1, \ldots, y_k) \in \theta$ with $x_i, y_i$ for all $1 \leq i \leq k$. Two tuples $\vec{x}$ and $\vec{y}$ are said to be $e$-equivalent if $x_i, y_i$ for all $i$.

We split the proof in two steps:

**Step 1.** For every $t \geq 0$, $I_t \subseteq \hat{T}^Q$, and hence $I^Q \subseteq \hat{T}^Q$.

We shall use the following fact that follows directly from the construction of the program $Q$:

**Fact.** For every IDB $I$ the relation $\hat{T}^Q$ is $e$-closed.

We prove the inclusion by induction on $t$. For $t = 0$ there is nothing to prove. Assume the inclusion holds for some $t \geq 0$. Consider a rule

$I \leftarrow J, R_1, \ldots, R_n$

(the case of a rule with no IDB in the head is identical), and suppose that a tuple $\vec{r} \in I$ was obtained via this rule at step $t + 1$. Let $\vec{y}$ be a tuple such that $x_i, y_i$ for all $i$; then modify the assignment of values to the variables of the rule by replacing each $x_i$ by $y_i$. Suppose $J^Q[I]$ and $\theta$ are $e$-closed, this assignment of values still satisfies all conditions in the head and so forces the presence of $\vec{y}$ in $I^{Q[t+1]}$.

Now we prove the inclusion by induction on $t$: for $t = 0$ there is nothing to show. Assume the inclusion holds for some $t \geq 0$. Suppose that a tuple $\vec{r} \in \hat{T}$ was obtained at step $t + 1$. First consider a rule of the form

$I \leftarrow J, R_1, \ldots, R_n$

(the case of a rule with no IDB in the head is identical), and suppose that a tuple $\vec{r} \in I$ was obtained via this rule at step $t + 1$. Consider the assignment of values to the variables of the rule that yields $\vector{r}$; by definition of the relations $R^S_i$ and by induction hypothesis, this assignment also satisfies the conditions in the head of the rule

$I \leftarrow J, R_1, \ldots, R_n$

and hence $\vec{r}$ is in $I^{Q[t+1]} \subseteq I^Q$. On the other hand if the tuple $\vec{r}$ was obtained at step $t + 1$ via a rule of the form

$I(x_1, \ldots, x_k) \leftarrow J(x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots, x_k), x_i, y_i$

(the case for the symmetric rule is identical), it means that there exists a tuple $e$-equivalent to $\vec{r}$ which is in $\hat{T}^{Q[t]}$, and hence in $I^Q$ by induction hypothesis. By the fact, we conclude that $\vec{r} \in I^Q$.

\[ \blacksquare \]