Fast Algorithms and Lower Bounds for Temporal Reasoning

Manuel Bodirsky* Jan Kára†

Abstract

We introduce two new tractable temporal constraint languages, which both strictly contain the class Ord-Horn of Bürkert and Nebel. The presented algorithms decide whether a given set of constraints from these languages is consistent in time that is quadratic in the input size; this also yields a new algorithm for Ord-Horn constraints, where the best known algorithms also have quadratic running time. We also prove that the two languages are *maximally tractable*, i.e., if we add a new temporal relation to one of these constraint languages, the corresponding constraint satisfaction problem becomes NP-complete. Our proof applies the so-called product Ramsey theorem, which we believe will be useful in similar contexts of constraint satisfaction complexity classification. Finally, we prove that (unlike Ord-Horn) the two languages cannot be solved by Datalog or by establishing local consistency.

1 Introduction

One of the most fundamental and well-known temporal constraint languages is the so-called *point algebra*. This language contains relation symbols for $\leq, <$, and $\neq$, interpreted over a dense linear order. Vilain, Kautz and van Beek showed that consistency of a given set of constraints over this language can be decided in polynomial time by local consistency techniques [23]. Later,

*Humboldt-Universität zu Berlin, Germany
†Charles University, Prague, Czech Republic

van Beek described an algorithm that runs in $O(n^2)$, where n is the number of variables [29].

A considerably larger tractable temporal constraint language was introduced by Bürkert and Nebel [9]. Their language, called *Ord-Horn*, strictly contains the point algebra. Bürkert and Nebel used resolution to show that consistency of a set of Ord-Horn constraints can be decided in $O(s^3)$, where s is the size of the input. They also showed that establishing path-consistency can be used to decide whether a given set of Ord-Horn constraints has a solution. Koubarakis [25] later presented an algorithm with a running time in $O(s^2)$. Our results lead to a new algorithm for Ord-Horn constraints, which also has a quadratic running time.

Ord-Horn is motivated by temporal reasoning tasks for constraints on *time intervals*. The study of constraints on intervals (which can be used to model events in time) was initiated by Allen [1], who introduced an algebra of binary constraint relations on intervals. Allen's interval algebra is studied intensively in Artificial Intelligence, where it is one of the benchmark applications of constraint satisfaction in general [10], but also in graph theory [17], database theory [30], and the theory of relation algebras [26]. The complexity to decide the consistency of a given set of constraints from Allen's algebra is in general NP-complete [1]. However, several fragments of Allen's interval algebra are tractable. All tractable fragments have been classified recently [12, 22].

It is well-known that every constraint on *intervals* can be translated into a constraint on *time points*. Bürkert and Nebel used this translation to identify one of the tractable fragments of Allen's interval algebra, namely the set of all interval constraints that translate to Ord-Horn constraints on points. Our work only concerns constraints on time points. However, using the translation from constraints on intervals to constraints on time points, our algorithms gives new results for reasoning on intervals as well.

There are temporal constraint languages for time points where one can not expect a polynomial time algorithm. A well-known temporal constraint language with an NP-complete consistency problem consists of a single relation symbol for the *betweenness relation*, which is the ternary relation $\{(x, y, z) \mid x{<}y{<}z \vee z{<}y{<}x\}$; another example of such an NP-complete language consists of the *cyclic ordering relation*, which is the ternary relation $\{(x, y, z) \mid x{<}y{<}z \vee y{<}z{<}x \vee z{<}x{<}y\}$. The constraint satisfaction problems for these two languages are listed as NP-complete in the book of Garey and Johnson [16]. A general classification of tractable and NP-complete temporal

constraint languages is not known.

We present two new *tractable* temporal languages that both strictly contain all Ord-Horn relations. These languages are defined by universal-algebraic closure properties. We will from now on call the constraints in the first of these languages *ll-closed*, and constraints in the second language *dual ll-closed*. Examples of ll-closed relations are the following four relations, defined by the formulas $x \leq y$, $x \neq y$, $(x{=}y \vee u{=}v) \rightarrow a{=}b$, and $x{>}y \vee x{>}z$. The presented algorithms for ll-closed and for dual ll-closed constraints have a running time that is quadratic in the input size.

Both languages are provably *maximally tractable*: we show that every language that strictly contains one of our two languages has an NP-complete constraint satisfaction problem. Our proof has three main ingredients: First, we apply the so-called (universal-) algebraic approach, which was previously mainly applied to *finite* domain constraint satisfaction [8]. Second, we need some fundamental concepts from model theory to make the algebraic approach work for temporal constraint languages, building on work in [2–6]. Third, we use the so-called *product Ramsey theorem (PRT)*, which becomes a particularly powerful tool for complexity classification of *infinite* domain constraint satisfaction problems.

Traditionally, one of the main algorithmic tools in constraint satisfaction, and in particular in temporal reasoning, are *local consistency techniques* [1, 9, 12, 17, 23], for instance algorithms based on establishing path-consistency. Consistency based algorithms can be formulated conveniently as Datalog programs [4,14,24]. Roughly speaking, Datalog is Prolog without function symbols, and comes from Database theory [13]. We show that, unlike Ord-Horn [9], ll-closed and dual ll-closed constraints can *not* be solved by a Datalog program. This shows in particular that there are path-consistent instances with ll-closed constraints that do not have a solution. In our proof we apply a pebble-game argument that was originally introduced for finite domains [14, 24], but has been shown to generalize to a wide range of infinite domain constraint languages, including temporal languages [4]. This is also interesting from a theoretical point of view: for constraint satisfaction problems of languages over a finite domains, all known algorithms are essentially based on group-theoretic algorithms or Datalog [14]. However, the algorithms we present for temporal reasoning are neither group-theoretic nor based on Datalog.

2 Temporal Constraint Languages

A *(qualitative) temporal relation* is a relation that is first-order definable in an unbounded countable dense linear order. All such linear orders are isomorphic [20,27], and we therefore use $(\mathbb{Q}, <)$ to denote this structure. An example of a temporal relation is the ternary *Betweenness* relation $\{(x, y, z) \in \mathbb{Q}^3 \mid x{<}y \wedge y{<}z \vee z{<}y \wedge y{<}x\}$ mentioned in the introduction. It is well-known that every temporal relation also has a *quantifier-free* definition [20, 27], i.e., we can define every temporal relation with a formula that is a Boolean combination of literals of the form $x < y$ (as shown above in the case of the Betweenness relation).

A *temporal constraint language* is an (at most countable) set of relation symbols $R_1, R_2, \ldots$, where each relation symbol R_i is associated with an arity $k_i \geq 2$, and is interpreted by a k_i-ary temporal relation. As an example, consider the set $\Gamma_0 := \{\neq, \leq, <, =\}$, with the obvious interpretation over $(\mathbb{Q}, <)$. For simplicity, we use the same symbol for the relation symbol and the corresponding temporal relation.

In this paper, we study the complexity of the validity problem for first-order sentences of the form

$$\exists x_1, \ldots, x_n.\ \phi_1 \wedge \cdots \wedge \phi_m\ ,$$

where $\Phi := \{\phi_1, \ldots, \phi_m\}$ is a set of atomic formulas with variables from $x_1, \ldots, x_n$ and relation symbols from a fixed constraint language Γ. This problem is also called the *constraint satisfaction problem CSP*(Γ) of Γ. The set Φ is called the *instance* of the CSP, and the atomic formulas $\phi_1, \ldots, \phi_m$ are called the *constraints* of the instance. Let $\phi = R(x_1, \ldots, x_k)$ be a constraint. We say that ϕ has *arity* $ar(\phi) = k$. Let x_i be from $\{x_1, \ldots, x_k\}$. We then say that ϕ is *imposed* on x_i. A tuple $(a_1, \ldots, a_n) \in \mathbb{Q}^n$ is called a *solution* for Φ, if the assignment $x_i := a_i$ satisfies all formulas in Φ.

Example. Let R be the 4-ary temporal relation defined by $(x{=}y \wedge y{<}u \wedge u{=}v) \vee (x{<}y \wedge y{<}u \wedge u{<}v)$. Then $\Phi_1 := \{R(x_1, x_2, y_1, y_2),\ R(x_1, x_2, y_2, y_3),\ R(x_1, x_2, y_3, y_1)\}$ is an instance of CSP($\{R\}$). It is easy to see that the sentence $\exists x_1, x_2, y_1, y_2, y_3 \bigwedge_{\phi \in \Phi_1} \phi$ is true, and a solution to Φ_1 is $(0, 0, 1, 1, 1)$.

It is straightforward to verify that whether or not an n-tuple t is a solution to an instance only depends on the weak linear order $tp(t)$ defined on

$\{1, \ldots, n\}$ by $(i, j) \in tp(t)$ iff $x_i \leq x_j$. We also say that t *satisfies* $tp(t)$ [1]. This observation leads to a natural way to represent temporal relations. If R is a k-ary temporal relation, R can be represented by a set $\mathcal{R}$ of weak linear orders on $\{1, \ldots, k\}$ as follows. For every k-tuple $t \in R$, the weak linear order $tp(t)$ is contained in $\mathcal{R}$. Conversely, for every weak linear order w in $\mathcal{R}$ there is a k-tuple $t \in R$ such that $w = tp(t)$. As an example, the relation R in the example above can be characterized as the set of all tuples that satisfies either $tp((0, 0, 1, 1))$ or $tp((0, 1, 2, 3))$.

We call a finite constraint language Γ *tractable*, if CSP(Γ) can be solved in polynomial time. The constraint language Γ_0 mentioned at the beginning of this section, for instance, corresponds to the well-studied point-algebra that we mentioned in the introduction, and is tractable. Classically, an *infinite* constraint language Γ is called tractable if every finite subset of the constraint language is tractable. However, the algorithmic results established for the tractable languages in the present paper are stronger. We therefore also introduce a stronger notion of tractability. An (finite or infinite) constraint language Γ is called *globally tractable*, if CSP(Γ) can be solved in polynomial time in the input size, where the relation symbols in the instance are represented as sets of weak linear orders as described above[2]. In this case, we measure the input size of a given instance Φ by the number n of variables in Φ and the *weighted* number $m = \sum_{R(x_1,\ldots,x_k)\in\Phi} k|\mathcal{R}|$ of constraints. This is, m denotes the size of the representations of all the constraints in the instance. Note that the input size grows with the same order as $n + m$.

If Γ is the set of all temporal relations, then CSP(Γ) is well-known to be NP-complete: in fact, already the constraint language that only contains a relation symbol for the Betweenness relation is NP-complete [16]. For containment in NP, note that one can verify in polynomial time whether a given vector $\overline{a} \in Q^n$ is a solution for a given instance Φ. We can therefore nondeterministically verify in polynomial time whether there exists a weak linear order on n elements and an (arbitrary) n-tuple t satisfying this weak linear order such that t is a solution to Φ.

To study the computational complexity of the CSP, reductions from one

[1] The notation *tp* is motivated by the concept of *(complete) types* in model theory; see e.g. [20, 27].

[2] Also for constraint satisfaction with finite domains, there is a potential difference between the notion of tractability and notions of global tractability. It is an open problem whether these notions coincide; in fact, for finite domain CSPs it has been conjectured that tractability and global tractability are equivalent [8].

constraint language to another can be described conveniently using the notion of *primitive positive definability* from logic (see e.g. [20, 27]). A formula is called *primitive positive*, if it has the form $\exists x_1, \ldots, x_l.\ \psi_1 \wedge \cdots \wedge \psi_l$, where ψ_i is atomic (it might be of the form $x = y$, i.e., we always include equality in our language). The atomic formulas might contain free variables and existentially quantified variables from $x_1, \ldots, x_l$. As usual, every formula with k free variables defines on a structure Γ a k-ary relation. Primitive positive definability of relations is an important concept in constraint satisfaction, because primitive positive definable relations can be 'simulated' by the constraint satisfaction problem. The following is frequently used in hardness proofs for CSPs [8].

LEMMA 2.1. *Let Γ be a constraint language, and let R be a relation that has a primitive positive definition in Γ. Then CSP(Γ) is NP-complete if and only if CSP($\Gamma \cup \{R\}$) is NP-complete.*

Primitive positive definability can be characterized by preservation under so-called *polymorphisms* – this is the starting point of the so-called *(universal-) algebraic approach* to constraint satisfaction (see e.g. [7, 8]). This approach brought together several research areas and proved to be extremely productive for constraint satisfaction with finite domains. We introduce the fundamentals in the next section.

3 The Algebraic Approach

We first introduce the fundamental concepts from model theory and universal algebra; they are standard, see e.g. [20, 28].

We say that a k-ary function (also called *operation*) $f : \mathbb{Q}^k \to \mathbb{Q}$ *preserves* an m-ary relation $R \subseteq \mathbb{Q}^m$ if whenever $R(x_1^i, \ldots, x_m^i)$ holds for all $1 \leq i \leq k$ in Γ, then $R\big(f(x_1^1, \ldots, x_1^k), \ldots, f(x_m^1, \ldots, x_m^k)\big)$ holds in $\mathbb{Q}$. If f preserves all relations of a temporal constraint language Γ, we say that f is a polymorphism of Γ. The unary bijective polymorphisms are called the *automorphisms* of Γ; the set of all automorphisms of Γ is denoted by $\mathrm{Aut}(\Gamma)$.

The set of all polymorphisms $\mathrm{Pol}(\Gamma)$ of a temporal constraint language forms an algebraic object called *clone* [28], which is a set of operations defined on a set D that is closed under composition and that contains all projections. Moreover, $\mathrm{Pol}(\Gamma)$ is also closed under interpolation: We say that a k-ary

operation f is *interpolated* by a set of k-ary operations F if for every finite subset A of $\mathbb{Q}$ there is some operation $g \in F$ such that $f(\overline{x}) = g(\overline{x})$ for every $\overline{x} \in A^k$. The set of operations that are interpolated by F is called the *local closure* of F; if F equals its local closure, we say that F is *locally closed*. We say that F *locally generates* an operation g if g is in the smallest locally closed clone containing all operations in F.

An operation $f : \mathbb{Q}^k \to \mathbb{Q}$ *depends on its i-th argument* if there exist tuples $(d_1, \ldots, d_k), (d'_1, \ldots, d'_k) \in D^k$ such that $f(d_1, \ldots, d_k) \neq f(d'_1, \ldots, d'_k)$, $d_j = d'_j$ for all $j \neq i$, and $d_i \neq d'_i$. In other words, there is a k-tuple such that changing the i-th coordinate of the k-tuple changes the value of the tuple under f.

The universal-algebraic approach to constraint satisfaction for temporal constraint languages rests on the following fact, which follows directly from a result in [6] (since temporal constraint languages are first-order definable in $(\mathbb{Q}, <)$, and therefore *ω-categorical* [20, 27]).

THEOREM 3.1. (FOLLOWS FROM [6]) *Let Γ be a temporal constraint language. Then a temporal relation R has a primitive positive definition in Γ if and only if R is preserved by all polymorphisms of Γ.*

For a temporal constraint language Γ, every automorphisms of $(\mathbb{Q}, <)$ is a polymorphism of Γ. Therefore, for this article we make the convention to say that an operation f *locally generates* another operation g if $\{f\} \cup \mathrm{Aut}((\mathbb{Q}, <))$ locally generates g. By the above remark this should cause no confusion.

Also observe that a surjective k-ary polymorphism can be represented by a weak linear order on $\mathbb{Q}^k$: we let $\overline{x} \leq \overline{y}$ iff $f(\overline{x}) \leq f(\overline{y})$. It is easy to see that if two surjective operations f and g define the same weak linear order on $\mathbb{Q}^k$, then there is an automorphism α of $(\mathbb{Q}, <)$ such that $f = \alpha(g)$. Hence, f is a polymorphism of a temporal constraint language Γ if and only if g is a polymorphism of Γ.

4 ll-closed Constraints

Let lex be a binary operation on $\mathbb{Q}$ such that $\mathrm{lex}(a, b) < \mathrm{lex}(a', b')$ if either $a < a'$, or $a = a'$ and $b < b'$. Note that every operation l satisfying these conditions is by definition injective. By the concluding paragraph in the previous section it is easy to see that all such operations locally generate the same clone.

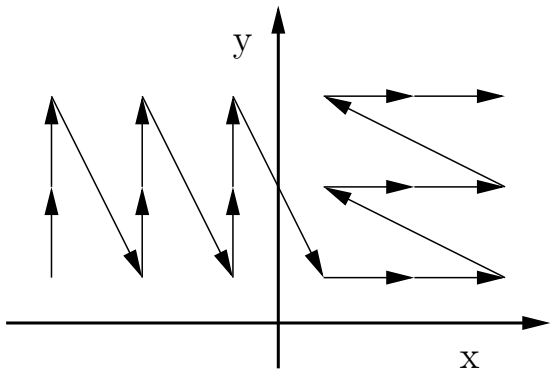

Figure 1: A visualization of the ll operation.

Let ll be a binary operation on $\mathbb{Q}$ such that $\mathrm{ll}(a,b) < \mathrm{ll}(a',b')$ if one of the following cases applies

- $a \leq 0$ and $a < a'$
- $a \leq 0$ and $a = a'$ and $b < b'$
- $a > 0$ and $b < b'$
- $a > 0$ and $b = b'$ and $a < a'$

For an illustration, see Figure 1. In diagrams like this one we draw a directed edge from (a,b) to (a',b') if $\mathrm{ll}(a,b) < \mathrm{ll}(a',b')$.

Again, all operations satisfying these conditions are by definition injective, and locally generate the same clone. It is also easy to see that ll locally generates lex. In the following we study equivalent characterizations of the relations that are preserved by ll.

Definition 1. *We say that a relation R is* ll-closed *if for every two weak orders o_1 and o_2 in R and every index $e \leq k$ the weak order o_3 is also in R, where o_3 is defined as follows: $(i,j) \in o_3$ iff one of the following holds*

- $(i,j) \in o_1$ *and* $(i,j) \in o_2$,
- $(i,j) \in o_1$, $(j,i) \notin o_1$, *and* $(i,e) \in o_1$, *or*
- $(i,j) \in o_2$, $(j,i) \notin o_2$, *and* $(e,j) \in o_1$

The following is not hard to show.

PROPOSITION 4.1. *A temporal relation R is preserved by the operation ll if and only if R is ll-closed.*

What is the complexity to decide whether a given constraint language is a tractable constraint language? This is known as the *meta-problem* for constraint satisfaction problems. We show that at least we can efficiently decide whether a given temporal constraint language is ll-closed.

THEOREM 4.1. *Given a constraint language where all relations are represented as lists of weak linear orders, one can decide in polynomial time in the input size whether the constraint language is ll-closed.*

Proof. (Sketch) We test for each relation R in the constraint language separately, whether it is ll-closed. Let R be k-ary. For all pairs (o_1, o_2) of weak linear orders on $\{1, \ldots, k\}$ in the representation of R, and for each index $e \leq k$, we can verify in linear time in k whether the weak linear order o_3 as described in Definition 1 is also contained in the representation of R. □

Similarly to the ll operation we can define a dual ll operation, as depicted in Figure 2. It is straightforward to dualize Definition 1, Proposition 4.1, Theorem 4.1, and their proofs accordingly. To show that the ll operation and the dual ll operation locally generate distinct clones, we define the following two relations, that will be also of importance in later arguments.

DEFINITION 2. *We define R^{min} to be the ternary relation $\{(x, y, z) \mid x{>}y \vee x{>}z\}$, and R^{max} to be $\{(x, y, z) \mid x{<}y \vee x{<}z\}$.*

Observe that $R^{min}(x, y, z)$ holds if and only if x is larger than the minimum of y and z. Similarly, $R^{max}(x, y, z)$ holds if and only if x is smaller than the maximum of y and z.

PROPOSITION 4.2. *The ll operation does not locally generate the dual ll operation and vice versa.*

Proof. To show that the operation ll does not locally generate the dual ll operation, it suffices to show that there is a temporal relation that is preserved by ll but not by dual ll (see Theorem 3.1). We claim that the relation R^{min} is preserved by the ll operation: Let (x_1, y_1, z_1) and (x_2, y_2, z_2) be triples that both satisfy the relation R^{min}. Without loss of generality, $x_1 < y_1$ (note that

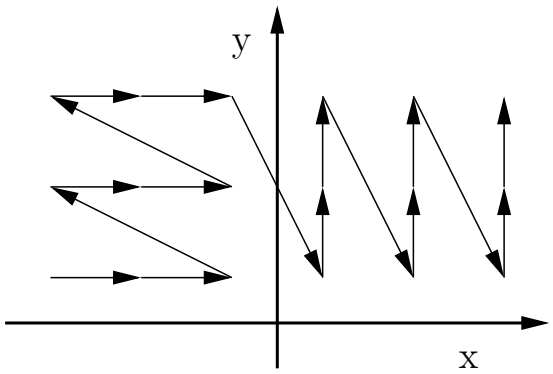

Figure 2: A visualization of the dual ll operation.

the relation is symmetric in the second and third argument). If in this case $x_2 \leq y_2$, then, because ll preserves $\leq$, we have that $\mathrm{ll}(x_1, x_2) \leq \mathrm{ll}(y_1, y_2)$, and because ll is injective, we have that $\mathrm{ll}(x_1, x_2) < \mathrm{ll}(y_1, y_2)$. Therefore $(\mathrm{ll}(x_1, x_2), \mathrm{ll}(y_1, y_2), \mathrm{ll}(z_1, z_2))$ satisfies R^{min}, and we are done. So let us assume that $x_2 > y_2$, and therefore $x_2 < z_2$. We can then show that $(\mathrm{ll}(x_1, x_2), \mathrm{ll}(y_1, y_2), \mathrm{ll}(z_1, z_2))$ satisfies R^{min} unless $x_1 > z_1$. So let us assume that $x_1 > z_1$. Now, in the cases where $y_1 \geq 0$ or $z_1 < 0$ the operation ll preserves R^{min}, since in this case ll acts like a lexicographic order on the two triples. Otherwise, $y_1 < 0$ and $z_1 \geq 0$. It is easy to check that then $\mathrm{ll}(y_1, y_2) < \mathrm{ll}(x_1, x_2)$.

However, R^{min} is not preserved by the dual ll operation: consider the tuples $t_1 := (-1, 1, -2)$ and $t_2 := (-1, -2, 1)$ that both satisfy R^{min}. If we apply the dual ll operation to these two tuples, we obtain dual-ll$(-1, -1) <$ dual-ll$(-2, 1) <$ dual-ll$(1, -2)$, and hence the tuple dual-ll(t_1, t_2) does not satisfy the relation R^{min}.

This shows that the ll operation does not locally generate the dual ll operation. Analogously, we can use the relation R^{max} to show that the dual ll operation does not locally generate the ll operation. □

5 Ord-Horn Constraints are ll-closed

The class of Ord-Horn constraints was introduced by Bürckert and Nebel [9] to identify a tractable class of interval constraints. It is always possible to translate interval constraints into temporal constraints [23]. If the translation of an interval constraint language falls into a tractable temporal constraint

language, the interval constraint language is tractable as well. Bürckert and Nebel showed that the class of *interval* constraints having a translation into Ord-Horn temporal constraints is a maximally tractable fragment of Allen's interval algebra. Note that this does not imply that the class of Ord-Horn constraints is a maximally tractable temporal constraint language on time points. Indeed, this is not the case, as we show in this section. Proposition 5.1 below shows that the class of Ord-Horn constraints is ll-closed. Since the relation R^{min} defined in Section 4 is ll-closed but not Ord-Horn, the class of ll-closed constraints is *strictly* larger than Ord-Horn. Finally, we prove in Section 6 that ll-closed constraints are tractable. Therefore, Ord-Horn is not a maximally tractable class of temporal constraints.

Definition 3. *A temporal relation is contained in the temporal constraint language* Ord-Horn *iff it can be defined by a conjunction of formulas of the form*

$$(x_1 = y_1 \wedge \cdots \wedge x_{k-1} = y_{k-1}) \rightarrow x_k \; C \; y_k \; ,$$

where $C \in \{<, \leq, =\}$.

Proposition 5.1. *All relations in Ord-Horn are preserved by ll and by dual ll.*

Proof. We will give the argument for the ll operation only; the argument for the dual ll operation is analogous. It suffices to show that every relation that can be defined by a formula Φ of the form $(x_1 = y_1 \wedge \cdots \wedge x_{k-1} = y_{k-1}) \rightarrow x_k \; C \; y_k$ is preserved by ll, where $C \in \{<, \leq, =\}$. Let t_1 and t_2 be two $2k$-tuples that satisfy Φ. Consider a $2k$-tuple k_3 obtained by applying ll componentwise to t_1 and t_2. We distinguish two cases: either there is an $i \leq k-1$ such that in one of the tuples $x_i = y_i$ is not satisfied – in this case $x_i = y_i$ is not satisfied in t_3 as well by injectivity of ll, and therefore the tuple t_3 satisfies Φ. Or $x_i = y_i$ holds for all $i \leq k-1$ in both tuples t_1 and t_2. But then, as t_1 and t_2 satisfy Φ, the literal $x_k C y_k$ holds in both t_1 and t_2. Since ll preserves all relations in $\{<, \leq, =\}$, the literal $x_k C y_k$ holds in t_3, and therefore t_3 satisfies Φ as well. □

6 An Algorithm for ll-closed Constraints

In this section we present an algorithm for ll-closed constraints. It is straightforward to dualize all arguments and the algorithm, and we will therefore also

obtain a distinct algorithm for dual ll-closed constraints.

One of the underlying ideas of the algorithm is to try to find a variable in the given instance such that there is a solution $\overline{s}$ where this variable denotes the smallest value. For this task, we present a procedure called Spec. If Spec fails to find such a solution $\overline{s}$, it returns a set of at least two variables that have to denote the same value in all solutions. Surprizingly, if there are no such variables that denote the same value in all solutions, then Spec does not fail, but produces a solution $\overline{s}$ for the instance.

To formally introduce our algorithm, the definitions below will be useful. Let $\phi = R(x_1, \ldots, x_k)$ be an atomic formula where R is a temporal relation R that is preserved by an operation f. Clearly, for all x_i from $x_1, \ldots, x_k$ the temporal relation defined by $\exists x_i.\phi$ is preserved by f as well. Therefore, if Φ is an instance of the CSP with constraints that are preserved by f, and $\overline{y}$ is a sequence of some of the variables of Φ, then $\Phi' := \{\exists \overline{y}.\phi \mid \phi \in \Phi\}$ can also be viewed as an instance of the CSP with constraints preserved by f. We call Φ' the *projection* of Φ to $X \setminus \overline{y}$. Note that if Φ' is inconsistent, then Φ is inconsistent as well.

The i-th entry in a k-tuple t is called *minimal* if $t[i] \leq t[j]$ for every $j \in [k]$. It is called *strictly minimal* if $t[i] < t[j]$ for every $j \in [k] \setminus \{i\}$.

Definition 4. *Let R be a k-ary relation. A set of entries $S \subseteq [k]$ is called* free for the i-th entry in R *if there exists a tuple $t \in R$ such that the i-th entry is minimal in t, and for every $j \in S$ it holds that $t[i] = t[j]$. We also say that t* defines a free set S.

Let R be a k-ary relation that is preserved by lex (recall that ll-closed constraints are preserved by lex as well). If $S_1, \ldots, S_l$ are all free sets for the i-th entry in R, we consider the corresponding tuples $t_1, \ldots, t_l$ and the tuple $t := \mathrm{lex}(t_1, \mathrm{lex}(t_2, \ldots \mathrm{lex}(t_{l-1}, t_l)))$. Since i is minimal in every tuple $t_1, \ldots, t_l$ and lex preserves both $<$ and $\leq$, it is also minimal in t. Because lex is injective, we have that $t[i] = t[j]$ if and only if these two entries are equal in every tuple $t_1, \ldots, t_l$. Hence, the free set for the i-th entry in R defined by the tuple t is a subset of every free set $S_1, \ldots, S_l$. We call this free set the *minimal free set* for the i-th entry in R.

Lemma 6.1. *Let R be a k-ary relation preserved by lex, let $i \in [k]$ and S be the minimal free set for the i-th entry in R. Then for every $t \in R$ it holds that either $t[i] = t[j]$ for every $j \in S$, or there is a $j \in S$ such that $t[j] < t[i]$.*

Proof. Let $t' \in R$ be the tuple that defines the minimal free set S. Suppose there is a tuple $t \in R$ such that not all entries in S are equal (in particular, $|S| > 1$). Consider the tuple $t'' := lex(t', t)$. By the properties of lex it holds that $t''[i] < t''[j]$ for every $j \in [k] \setminus S$. Furthermore, $t''[i] \leq t''[j]$ for $j \in S$ if and only if $t[i] \leq t[j]$. So unless t'' defines a smaller free set for i in R (which would be a contradiction) it must hold that $t''[i] > t''[j]$ for some $j \in S$. □

Let Φ be an instance of the CSP where each constraint relation is preserved by lex. We create the following (directed) graph $G_\Phi = (X, E)$ for this instance. The vertices of the directed graph are the variables from the instance Φ that can be minimal (i.e., in all the constraints from Φ the corresponding entries can be minimal). We now add edges for each constraint $\phi \in \Phi$ as follows. Suppose the constraint ϕ is of the form $R(x_1, \ldots, x_k)$. For each $1 \leq i \leq k$, let S_i be the minimal free set for the i-th entry in R. We then add for each $y \in S_i \setminus \{x_i\}$ an edge (x_i, y) to E. We call the graph G_Φ the *constraint graph* of Φ.

Example. We return to the example from Section 2. The constraint graph G_{Φ_1} for the instance in this example has the two vertices x_1 and x_2, and an edge from x_2 to x_1. The projection Φ'_1 of Φ_1 to $\{y_1, y_2, y_3\}$ has a constraint graph $G_{\Phi'_1}$ with the three vertices y_1, y_2, y_3, and edges from y_2 to y_1, from y_3 to y_2, and from y_1 to y_3.

A strongly connected component K (see [11]) of a directed graph G such that no edge leaves K is called a *sink component* of G. A vertex of G that belongs to a sink component of size one is called a *sink*.

LEMMA 6.2. *Let K be a sink component of the graph G_Φ for Φ. Then all variables from K must have equal values in all solutions of Φ.*

Proof. We assume that K has at least two vertices (otherwise the lemma is trivial). Consider some solution of Φ and let $M \subseteq K$ be the set of vertices assigned the minimal value among the variables of the sink component K. If $M = K$, we are done. Otherwise, because K is strongly connected, there is an edge from some vertex $v \in M$ to $v' \in K \setminus M$. But then there is a minimal free set of a constraint imposed on v such that neither all variables of the free set are equal (the value of v' is different) nor there is some variable of the free set with value smaller than the value of v (because v has the minimal value among the variables in K). This contradicts Lemma 6.1. □

Lemma 6.2 immediately implies that we can add constraints of the type $x = y$ for all variables x, y from the same sink component. Equivalently, we can consider the CSP instance Φ' where all the variables in sink components are *contracted*. We formalize this idea as follows. Let S be a subset of the variables of Φ. We define Φ' to be the instance where all variables from S are replaced by the same variable.

In some cases, a solution to a projected instance with ll-closed constraints can be used to construct a solution to the original constraint. We say that a tuple (in particular, a solution of an instance) $\overline{x}$ is *injective* if $x_i \neq x_j$ for all $i \neq j$.

LEMMA 6.3. *Let Φ be an instance of the CSP with variables X and ll-closed constraints. Let x be a sink in G_Φ. If the projection Φ' of Φ to $X \setminus \{x\}$ has an injective solution, then Φ has an injective solution as well.*

Proof. Let $\overline{s}$ be an injective solution to Φ'. Consider a constraint $\phi = R(x_1, \ldots, x_k)$ from Φ that is imposed on x. By the definition of Φ' there is a tuple $t \in R$ such that t agrees with $\overline{s}$ on $\{x_1, \ldots, x_k\} \setminus \{x\}$. Because x is a sink, there is tuple $t' \in R$ such that the entry corresponding to x is strictly minimal. It is now easy to check that there are automorphisms α, β such that the tuple $t'' = \alpha(ll(\beta(t'), t))$ agrees with $\overline{s}$ on $X \setminus \{x\}$, and that the entry corresponding to x is strictly minimal. As R is ll-closed, $t'' \in R$. Hence we see that for each constraint there is a tuple where the entry corresponding to x is strictly minimal and the rest of the tuple agrees with $\overline{s}$ on $X \setminus \{x\}$. So we can extend the solution by assigning to x a value smaller than any value used in $\overline{s}$ and the lemma readily follows. □

Now we are ready to state our algorithm for instances with ll-closed constraints.

ALGORITHM 6.1.

```
Spec(Φ) {
// Input: Φ constraints with variables X
// Output: If Φ has no solution, then return false
// If Φ has an injective solution, then return true
// Otherwise return S ⊆ X, |S| ≥ 2, s.t. for all
// x, y ∈ S we have x = y in all solutions of Φ
    G := ConstructGraph(Φ)
```

```
    Y := ∅, Φ' := Φ
    While G contains a sink s
        Y := Y ∪ {s}
        Φ' := projection of Φ' to X \ Y
        G := ReconstructGraph(Φ')
    If Y = X then return true
    else if V_G = ∅ then return false
    else return a sink component S from G
    end if }

Solve(Φ): {
// Input: instance Φ with variables X
// Output: true or false
    S := Spec(Φ)
    If S = false then return false
    else if S = true then return true
    else
        Let Φ' be contraction of S in Φ
        return Solve(Φ')
    end if }
```

THEOREM 6.1. *The procedure Solve*(Φ) *in Algorithm 6.1 decides whether a given set of ll-closed constraints* Φ *has a solution. There is an implementation of the algorithm that runs in time* $O(nm)$*, where* n *is the number of variables of* Φ *and* m *is the weighted number of constraints in* Φ*.*

Proof. The correctness of the procedure Spec immediately implies the correctness of the procedure Solve. In the procedure Spec, after iterated deletion of sinks in G', we have to distinguish three cases.

In the first case $Y = X$, and $V_G = \emptyset$. In this case we construct by induction an injective solution of Φ as follows. Let $x_1, \ldots, x_n$ be the elements from Y in the reverse order in which they were taken into Y. For $0 \leq i \leq n$, let Φ_i be the instance Φ projected to $X \setminus \{x_1, \ldots, x_i\}$. Note that $\Phi_0 = \Phi$, and that $\Phi_n = \Phi'$ is the projection of Φ to the empty set, which trivially has an injective solution. We inductively assume that Φ_i, for $i \leq n$, has an injective solution. Then Lemma 6.3 applied to x_i, the instance Φ_{i-1}, and the injective solution to Φ_i implies that also Φ_{i-1} has an injective solution. By induction,

Φ_i has an injective solution for all $0 \leq i \leq n$, and in particular $\Phi_0 = \Phi$ has an injective solution. Therefore, the output `true` of Spec is correct.

In the second case, $Y \neq X$ and $V_G = \emptyset$. Note that in every solution to Φ' some variable must denote the minimal value. However, since $V_G = \emptyset$, no variable can denote the minimal element, and therefore Φ' has no solution. Because Φ' is a projection of Φ to $X \setminus Y$, the instance Φ is inconsistent as well.

In the third case, $V_G \neq \emptyset$, and therefore G must contain a sink component S. Because V_G does not contain sinks, $|S| \geq 2$. We claim that for all variables $x, y \in S$ we have $x = y$ in all solutions to Φ. Lemma 6.2 applied to the projection of Φ to $X \setminus Y$ implies that whenever some variables are in the same sink component, they must have the same value in every solution, and hence the output is correct in this case as well.

Since in each recursive call of Solve the instance in the argument has at least one variable less, Solve is executed at most n times. Since the projection Φ' and the constraint graph G can be constructed in linear time in the input size, the total running time is easily seen to be cubic in the input size. However, we will now describe an implementation of the sub-procedures such that the total running time is in $O(nm)$.

First, note that we can assume that n is smaller than m. Otherwise, the constraint is not connected (we use the notion of connectivity for instances of the CSP as e.g. in [19]). We can in this case use the same implementation, analyse the running time for each of the connected components separately, and will get the same result.

In our implementation, if s is a sink of G at some iteration of the while-loop, we first compute the projection of Φ' to $X \setminus Y$ by updating only the constraints imposed on s in Φ'. The total number number of operations we have to perform in all iterations of the while-loop is then bounded by m.

The constraint graph G is now updated as follows. Note that since the new instance resulted from the previous instance by projection, some edges in G need to be removed. Moreover, we might have to add some new variables and edges incident to these new variables. For the edge deletions, it again suffices to process only constraints that are imposed on s, and the total cost for this step during all iterations of the while-loop is bounded by m. To see which variables have to be added to G, we maintain a data structure that stores for each variable u the list of constraints in which u occurs at an entry that can not be minimal in the constraint. When computing Φ', we can maintain this data structure without increasing our asymptotic running

time. As soon as some of these lists becomes empty, we add the corresponding variable to G, and compute the outgoing edges in G for this variable. Again, the total costs for these operations are in $O(m)$. Finally, since we can construct the graph G, we can clearly also maintain a list of sinks and decide in constant time whether G has a sink or not. This shows that the algorithm can be implemented in time $O(nm)$. □

7 ll-closed Constraints are Maximally Tractable

Suppose Δ is a constraint language that strictly contains all ll-closed constraints. In this section we want to show that CSP(Δ) is NP-complete. Unless P=NP, this shows that ll-closed constraints are maximally tractable in the sense that every larger constraint language does not have a polynomial time algorithm.

Since Δ contains all ll-closed constraints, the polymorphism clone of Δ is contained in the clone locally generated by ll and $\mathrm{Aut}(\mathbb{Q}, <)$. In particular, all polymorphisms of Δ are injective and preserve the relations $\leq$ and R^{min}, because the same holds for ll, for all automorphisms of $(\mathbb{Q}, <)$, and also for all operations that are locally generated by these operations.

If all polymorphisms of Δ preserve the Betweenness relation, then Theorem 3.1 shows that the Betweenness relation is primitive positive definable in Δ. Since Betweenness is NP-complete [16], Lemma 2.1 shows that in this case CSP(Δ) is NP-complete. So suppose that Δ has a polymorphism that violates the betweenness relation. One of the starting observations is that Δ has in this case also a *binary* polymorphism that violates the Betweenness relation. This follows from the following more general lemma.

LEMMA 7.1. *Let R be a k-ary temporal relation whose representation consists of m weak linear orders on $\{1, \ldots, k\}$. If f is an operation that violates R, then f locally generates an m-ary operation that violates R.*

Proof. Let f' be an operation of smallest arity l that is locally generated by f and violates R. Then there are k-tuples $t_1, \ldots, t_l$ in R such that $f'(t_1, \ldots, t_l) \notin R$. For $l > m$ there are two tuples t_i and t_j that satisfy the same weak linear order, and therefore there is an automorphism α of $(\mathbb{Q}, <)$ such that $\alpha(t_i) = t_j$. Then the $l-1$-ary operation g defined as

$g(t_1, \ldots, t_{j-1}, t_{j+1}, \ldots, t_l) := f(t_1, \ldots, t_{i-1}, \alpha(t_j), t_{i+1}, \ldots, t_l)$ also violates R, a contradiction. Hence, $l \leq m$. In case that $l = m$, we are done. In case that $l \leq m$, the result also follows, because we can then always obtain an m-ary operation that violates R by composing f' with projections. □

We now want to prove that every binary injective operation that preserves $\leq$ and R^{min} but violates the Betweenness relation already locally generates the operation ll. This contradicts our assumption that Δ *strictly* contains all ll-closed constraints, and we have shown that ll-closed constraints are maximally tractable.

In our argument we make use of the so-called *product Ramsey theorem*, which is given here in a formulation that follows from Theorem 4.1 in [15]; see [18] for a general introduction to Ramsey theory. If $S_1, \ldots, S_d$ are sets, we call a set of the form $S_1 \times \cdots \times S_d$ a *grid*, and also write S^d for a product of the form $S \times \cdots \times S$ with d factors. A $[k]^d$*-subgrid* of a grid $S_1 \times \cdots \times S_d$ is a subset of $S_1 \times \cdots \times S_d$ of the form $S'_1 \times \cdots \times S'_d$ where S'_i is a k-element subset of S_i.

THEOREM 7.1. (FOLLOWS FROM [15]) *Let k, r, and d be positive integers, and $S_1, \ldots, S_d$ be infinite sets. If $S_1 \times \cdots \times S_d$ is linearly ordered, and the $[k]^d$-subgrids of $S_1 \times \cdots \times S_d$ are colored with r colors such that $[k]^d$-subgrids inducing isomorphic linear orders get the same color, then there exist infinite sets $S'_i \subset S_i$ such that all $[k]^d$-subgrids of $S'_1 \times \cdots \times S'_d$ have the same color.*

We use this theorem for $d = k = r = 2$. To illustrate the way in which we use the theorem, let us first show the following result.

PROPOSITION 7.1. *Every binary injective operation f that preserves $\leq$ locally generates lex.*

Proof. Let (x_1, y_1), (x_2, y_1), (x_1, y_2), (y_2, y_2) be four points from $\mathbb{Q}^2$. Since f preserves $\leq$ and is injective, either $f(x_1, y_1) < f(x_2, y_1) < f(x_1, y_2) < f(x_2, y_2)$ or $f(x_1, y_1) < f(x_1, y_2) < f(x_2, y_1) < f(x_2, y_2)$. Color the $[2]^2$ subgrids of $\mathbb{Q}^2$ with two colors according to these two cases. The product Ramsey theorem implies that we find infinite sets $X, Y \subset \mathbb{Q}$ such that all $[2]^2$-subgrids of $X \times Y$ are monochromatically colored. Then it is easy to verify that the operation f together with the automorphisms of $(\mathbb{Q}, <)$ interpolates lex. □

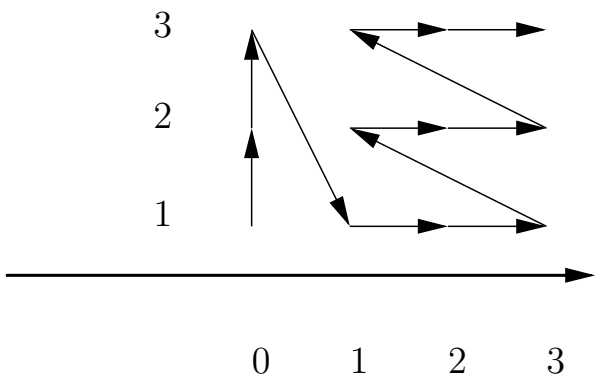

Figure 3: An illustration of the operation g from Lemma 7.2 for $k = 3$.

Note that preservation by lex alone does not give rise to a tractable constraint language, since lex preserves the Betweenness relation. For an illustration of the next lemma, see Figure 3.

LEMMA 7.2. *Let f be a binary injective operation that preserves $\leq$ and R^{min}, but does not preserve Betweenness. Then for every $k \geq 0$ there are automorphisms α and β of $(\mathbb{Q}, <)$ such that the operation g defined as $g(x, y) := f(\alpha(x), \beta(y))$ or $g(x, y) := f(\alpha(y), \beta(x))$ satisfies for natural numbers i, j, p, q with $0 \leq i \leq p \leq k$ and $1 \leq j, q \leq k$ that $g(i, j) < g(p, q)$ if and only if*

- *$i = 0$ and $p \geq 1$,*
- *$i = p$ and $j < q$, or*
- *$0 < i < p$ and $j \leq q$.*

Note that the operation g of Lemma 7.2 is injective and preserves $\leq$. Also note that the ll operation satisfies the conditions formulated for g.

Proof. Since f does not preserve the Betweenness relation, and is injective, there are elements a_1, b_1, c_1 and a_2, b_2, c_2 such that $a_1 < b_1 < c_1$, $a_2 > b_2 > c_2$, and one of the following four cases applies:

1. $f(a_1, a_2) < f(c_1, c_2) < f(b_1, b_2)$
2. $f(c_1, c_2) < f(a_1, a_2) < f(b_1, b_2)$

3. $f(b_1, b_2) < f(c_1, c_2) < f(a_1, a_2)$

4. $f(b_1, b_2) < f(a_1, a_2) < f(c_1, c_2)$.

We can exclude cases 3 and 4, because (b_1, a_1, c_1) and (b_2, a_2, c_2) satisfy the relation R^{min}, but $f(b_1, b_2), f(a_1, a_2), f(c_1, c_2)$ does not.

In case 1, define $x := b_1$, $x' := c_1$, $y := c_2$, $y' := b_2$, and in the second case define $x := a_1$, $x' := b_1$, $y := b_2$, $y' := a_2$. Now choose an infinite set $X_1 \subset \mathbb{Q}$ of elements between x and x', and an infinite set $Y_1 \subset \mathbb{Q}$ of elements between y and y'.

As in the proof of Proposition 7.1, we apply the product Ramsey theorem to $X_1 \times Y_1$, and find infinite sets $X_1' \subset X_1$, $Y_1' \subset Y_1$ such that all $[2]^2$-subgrids of $X_1' \times Y_1'$ are monochromatically colored.

Next, select an infinite set X_2 of elements from $\mathbb{Q}$ that are larger than c_1, and an infinite set Y_2 of elements from $\mathbb{Q}$ that are larger than a_2. We apply the product Ramsey theorem two more times: this time we color the $[2]^2$ subgrids of $X_1' \times Y_2$ and the $[2]^2$ subgrids of $X_2 \times Y_1'$ and get infinite sets $X_1'' \subset X_1', Y_2' \subset Y_2, X_2' \subset X_2, Y_1'' \subset Y_1$ such that the $[2]^2$-subgrids of $X_1'' \times Y_2'$ and of $X_2' \times Y_1''$ are monochromatically colored.

Now suppose that two of these infinite grids are colored differently. In this case it is easy to see that we get ll or dual ll by local interpolation. In the case that we get the ll operation, we are done. It is impossible that g locally generates the dual ll operation, since g is generated by the ll operation, but we have seen in Proposition 4.2 that the clone locally generated by ll and the clone locally generated by the dual ll operation are incomparable.

So, assume that all three grids are oriented in the same direction. Up to reflection of arguments of g, we can assume without loss of generality that for all four points (x_1, y_1), (x_2, y_1),(x_1, y_2), (y_1, y_2) from one of these grids we have $f(x_1, y_1) < f(x_2, y_1) < f(x_1, y_2) < f(x_2, y_2)$. See Figure 4.

Now, let p be the minimal element of X_2', and let r be the minimal and s the maximal element of Y_1''. Suppose that we are in the case $f(a_1, a_2) < f(c_1, c_2) < f(b_1, b_2)$. Since f preserves $\leq$ and is injective, we have that $f(a_1, s) < f(a_1, a_2)$. We know that $f(a_1, a_2) < f(c_1, c_2)$. Again, since f preserves $\leq$ and is injective, we have that $f(c_1, c_2) < f(p, r)$. By transitivity, $f(a_1, s) < f(p, r)$. It is now straightforward to construct the automorphisms α and β as required in the statement of the Lemma (α maps a_1 to 0). The case $f(c_1, c_2) < f(a_1, a_2) < f(b_1, b_2)$ is analogous. □

LEMMA 7.3. *Let g be an operation that satisfies the conditions of Lemma 7.2. Then every relation that is preserved by g is ll-closed.*

Proof. Let R be a k-ary relation that is preserved by f. We show that R is ll-closed. Let t_1 and t_2 be two k-tuples from R. We have to show that $t_3 := \mathrm{ll}(t_1, t_2)$ is in R as well. Without loss of generality we can assume that $t_i < t_j$ if $i < j$ (otherwise, rename arguments of R). If all entries of t_1 are larger than 0, then we choose automorphisms $\alpha, \beta \in \mathrm{Aut}(\mathbb{Q}, <)$ such that the entries of $\alpha(t_1)$ and $\beta(t_2)$ are from $\{1, \ldots, k\}$. Clearly we can then also choose an automorphism $\gamma \in \mathrm{Aut}(\mathbb{Q}, <)$ that maps $g(\alpha(t_1), \beta(t_2))$ to t_3, and we are done.

We now prove by induction on $s \geq 1$ that there is a tuple r in R that coincides with $\mathrm{ll}(t_1, t_2)$ on the first s entries. For the initial case of the induction, let $l_1 \geq 1$ be the largest index such that $t_1[l_1] = t_1[1] \leq 0$. Then, let $\alpha \in \mathrm{Aut}(\mathbb{Q}, <)$ be such that $\alpha(t_1[i]) = 0$ for $i \leq l_1$ and such that the remaining entries of $\alpha(t_1)$ are from $\{1, \ldots, k\}$. Let $\beta \in \mathrm{Aut}(\mathbb{Q}, <)$ be such that the entries of $\beta(t_2)$ are from $\{1, \ldots, k\}$. It is easy to verify that there is a permutation γ in $\mathrm{Aut}(\mathbb{Q}, <)$ that maps $g(\alpha(t_1[i]), \beta(t_2[i]))$ to $t_3[i]$ for $0 \leq i \leq l_1$. Then we let r be the tuple $\gamma(g(\alpha(t_1), \beta(t_2)))$ from R.

For the induction step, suppose that we have already constructed a tuple r that conincides with t_3 on the first s entries. If the remaining entries of t_3 are all positive, it is easy to find a permutation γ in $\mathrm{Aut}(\mathbb{Q}, <)$ that maps r to t_3. Let $l_s \geq 1$ be the largest index such that $t_1[s + l_s] = t_1[s + 1] \leq 0$. Let $\alpha, \beta \in \mathrm{Aut}(\mathbb{Q}, <)$ be such that $\alpha(t_1[s + 1]) = 0$, $\alpha(t_1[i]) \in \{1, \ldots, k\}$ for $i > s + l_s$, and $\beta(r) \in \{1, \ldots, k\}$. Then it is again easy to see that there is a permutation γ in $\mathrm{Aut}(\mathbb{Q}, <)$ that maps $g(\alpha(t_1[i]), \beta(r[i]))$ to $t_3[i]$ for $1 \leq i \leq s + l_s$: for $1 \leq i \leq s$, this follows from the fact that g preserves $\leq$. For $s + 1 \leq i \leq s + l_s$ this follows from the properties of g derived in Lemma 7.2. □

Lemma 7.2 and 7.3 together yield the following.

THEOREM 7.2. *Let f be a binary injective operation that preserves $\leq$ and R^{min}, but does not preserve Betweenness. Then every relation that is preserved by f is ll-closed.*

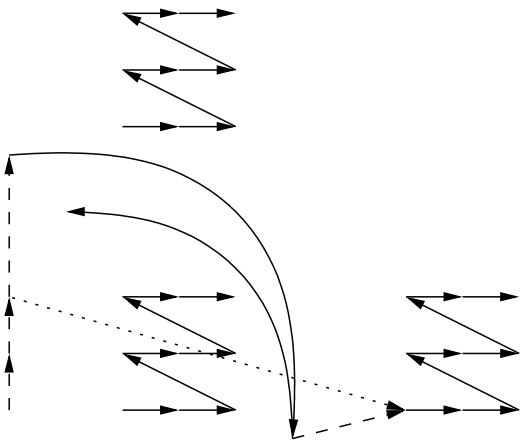

Figure 4: A diagram for Lemma 7.2. The non-straight arcs are present because f violates Betweenness. We also see three $[3]^2$ grids that are oriented in the same direction. The dashed arcs are present, because f preserves $\leq$. The dotted edge follows from transitivity. We finally have the situation of Lemma 7.2 for $k = 3$.

8 ll-closed Constraints and Datalog

In this section, we introduce Datalog, and then prove that the constraint satisfaction problem for ll-closed constraints can not be solved by Datalog programs. This result should not be confused with the weaker fact that establishing k-consistency does not imply global consistency, for any k. This was shown for Ord-Horn in [25]. But recall that Ord-Horn *can* be solved by a Datalog program [9].

All constraint satisfaction problems studied in the literature so far where one can show that they can not be solved by Datalog have the *ability to count* [14]. However, it is easy to verify that the temporal constraint language that only contains the ternary relation $R^{\min}$ does *not* have the ability to count. However, we present a proof that CSP($\{R^{\min}\}$) can not be solved by a Datalog program.

We will now define Datalog. Our definition will be purely operational; for the standard semantical approach to the evaluation of Datalog programs, see [13]. A Datalog program is a finite set of Horn clauses, i.e., clauses of the form $\psi \leftarrow \phi_1, \ldots, \phi_l$, where $l \geq 0$ and where $\psi, \phi_1, \ldots, \phi_l$ are atomic formulas of the form $R(\overline{x})$. The formula ψ is called the *head* of the rule, and $\phi_1, \ldots, \phi_l$ are called the *body*. We assume that all variables in the head also

occur in the body. The relation symbols occurring in the head of some clause are called *intentional*, and all other relation symbols in the clauses are called *extensional*.

If Γ is a finite temporal constraint language, we might use Datalog programs to solve CSP(Γ) as follows. Let Π be a Datalog program whose extensional symbols are from Γ, and let L be the set of intentional relation symbols of Π. We assume that there is one distinguished 0-ary intentional relation symbol `false`. Now, suppose we are given an instance Φ of CSP(Γ). The *evaluation* of Π on Φ proceeds in steps $i = 0, 1, \ldots$. At each step i we maintain a set of literals Φ^i with relation symbols from L; it always holds that $\Phi^i \subset \Phi^{i+1}$. Each clause of Π is understood as a rule that may derive a new literal (with a relation symbol from L) from the literals in Φ^i. Initially, we have $\Phi^0 := \Phi$. Now suppose that $R_1(x_1^1, \ldots, x_{k_1}^1), \ldots, R_l(x_1^l, \ldots, x_{k_l}^l)$ are literals in Φ^i, and $R_0(y_1^0, \ldots, y_{k_0}^0) \leftarrow R_1(y_1^1, \ldots, y_{k_1}^1), \ldots, R_l(y_{k_1}^l, \ldots, y_{k_l}^l)$ is a rule from Π, where $y_j^i = y_{j'}^{i'}$ if and only if $x_j^i = x_{j'}^{i'}$. Then $R_0(x_1^0, \ldots, x_l^0)$ is the newly derived literal in Φ^{i+1}, where $x_j^0 = x_{j'}^i$ if and only if $y_j^0 = y_{j'}^i$. The procedure stops if no new literal can be derived. We say that Π solves CSP(Γ), if for every instance Φ of CSP(Γ) there exists an evaluation of Π on Φ that derives `false`if and only if Φ has no solution.

We want to remark that the so-called method of *establishing path-consistency*, which is very well-known and frequently applied in Artificial Intelligence, can be formulated with Datalog programs where the intentional symbols are at most binary and all rules use at most three variables in the body.

We prove that already for the temporal language that only consists of $R^{\min}$ there is no Datalog program that solves the corresponding constraint satisfaction problem. We use a pebble-game characterization of the expressive power of Datalog, which was originally shown in [14] and [24] for finite domain constraint satisfaction, and which holds for a wide variety of infinite domain constraint languages as well, including qualitative temporal constraint satisfaction (see the journal version of [4]).

Let Γ be a temporal constraint language, and let Φ be an instance of CSP(Γ). Then the *existential k-pebble game on* Φ is the following game between the players *Spoiler* and *Duplicator*. Spoiler has k pebbles $p_1, \ldots, p_k$. He places his pebbles on variables from Φ Initially, no pebbles are placed. In each round of the game Spoiler picks some of this pebbles. If they are already placed on Φ, then Spoiler first removes them from Φ. Spoiler then places these pebbles on variables from Φ, and Duplicator responds by assigning elements

from $\mathbb{Q}$ to these variables. This assignment has to satisfy all the constraints ϕ from Φ where all variables in ϕ are pebbled, otherwise Spoiler wins the game. Duplicator wins, if the game continues *forever*, i.e., if Spoiler can never win the game.

THEOREM 8.1. (FROM [4]) *Let Γ be a temporal constraint language. There is no Datalog program that solves CSP(Γ) if and only if for every k there exists an inconsistent instance of CSP(Γ) such that Duplicator wins the existential k-pebble game on Φ.*

The rest of this section is devoted to the proof of the following theorem.

THEOREM 8.2. *There is no Datalog program that solves CSP($\{R^{\min}\}$).*

Proof. Let k be an arbitrary number. To apply Theorem 8.1 we have to construct an inconsistent instance Φ of CSP($\{R^{\min}\}$) such that Duplicator wins the existential k-pebble game on Φ.

For this, let G be a 4-regular graph of girth $2k+1$, i.e., all cycles in G have more than $2k$ vertices. It is known and easy to see that such graphs exist; it is even known that there are such graphs of size exponential in k [21]. Orient the edges in G such that there are exactly two outgoing and two incoming edges for each vertex in G (since G is 4-regular, there exists an Euler tour for G, which shows that such an orientation exists [11]).

Now we can define our instance Φ of CSP($\{R^{\min}\}$) as follows. The variables of Φ are the vertices from G. The instance Φ contains the constraint $R^{\min}(w,u,v)$ iff uw and vw are the two incoming edges at vertex w. We claim that Φ does not have a solution: if there was a solution, some variable w must denote the minimal value. But for every variable w we find a constraint $R^{\min}(w,u,v)$ in Φ, and this constraint is violated since either u or v must be strictly smaller than w.

Consider a connected non-empty subgraph G' of G having at most $2k$ vertices where only one vertex r has no outgoing edges, and where all vertices have either two or no incoming edges. Since G has girth $2k+1$, G' must be a binary tree with root r. We call G' *dominated*, if all leaves in G' are pebbled.

Duplicator always maintains the property that whenever the root r in a dominated tree is pebbled during the game, then the value assigned to r is strictly larger than the minimum of all the values assigned to the leaves. Clearly, this property is satisfied at the beginning of the game.

Suppose that during the game Spoiler pebbles the variable u. Let $T_1, \ldots, T_s$ be those newly created dominated trees in G that have pebbled roots $r_1, \ldots, r_s$, for $s \geq 0$. If $s > 0$, let r_i be the root which received the minimal value a among all the roots $r_1, \ldots, r_s$. We claim that if u is the root of a dominated tree T, then a is strictly larger than the minimum b of all the values assigned to the leaves of T. Otherwise, the graph $T \cup T_i$ was a dominated tree that violates the invariant even before the variable u has been pebbled, a contradiction. Therefore, in this case Duplicator can choose a value c between b and a for the variable u. Since c is smaller than a, in all the new dominated trees $T_1, \ldots, T_s$ in G the value assigned to $r_1, \ldots, r_s$ is strictly larger than c, and hence the invariant is preserved. Moreover, if $R(w, u, v)$ (or $R(w, v, u)$) is a constraint in Φ where w and v have been pebbled, then this constraint is satisfied by the assignment.

Since c is larger than b, this choice also guarantees that if v, v' are pebbled variables then any constraint of the form $R^{\min}(u, v, v')$ is satisfied, because in this case the variables u, v, v' induce a dominated tree with root u in G.

If there is no dominated tree T where u is the root, then Duplicator assigns a value to u that is smaller than all values assigned to other variables. If $s = 0$, Duplicator plays a value that is larger than all values assigned to other variables. In both cases it is easy to check that Duplicator maintains the invariant, and satisfies all constraints $\phi \in \Phi$ where all variables are pebbled. By induction, we have shown that Duplicator has a winning strategy for the existential k-pebble game on Φ. □

References

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] M. Bodirsky. The core of a countably categorical structure. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS'05)*, LNCS 3404, pages 100–110, Springer-Verlag, 2005.

[3] M. Bodirsky and H. Chen. Oligomorphic clones. To appear in Algebra Universalis, 2006.

[4] M. Bodirsky and V. Dalmau. Datalog and constraint satisfaction with infinite templates. In *Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science (STACS'06)*, LNCS 3884, pages 646–659. A journal version is available from the webpage of the first author, 2006.

[5] M. Bodirsky and J. Kára. The complexity of equality constraint languages. In *Proceedings of the International Computer Science Symposium in Russia (CSR'06)*, 2006.

[6] M. Bodirsky and J. Nešetřil. Constraint satisfaction with countable homogeneous templates. *Journal of Logic and Computation (JLC)*, 16(3):359–373, 2006.

[7] A. Bulatov, P. Jeavons, and A. Krokhin. The complexity of constraint satisfaction: An algebraic approach (a survey paper). *In: Structural Theory of Automata, Semigroups and Universal Algebra (Montreal, 2003), NATO Science Seiries II: Mathematics, Physics, Chemistry*, 207:181–213, 2005.

[8] A. Bulatov, A. Krokhin, and P. G. Jeavons. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34:720–742, 2005.

[9] H.-J. Bürckert and B. Nebel. Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra. *Journal of the ACM*, 42(1):43–66, 1995.

[10] R. Dechter. *Constraint Processing.* Morgan Kaufmann, 2003.

[11] R. Diestel. *Graph Theory.* Springer–Verlag, New York, 1997.

[12] T. Drakengren and P. Jonsson. Twenty-one large tractable subclasses of allen's algebra. *Artif. Intell.*, 93:297–319, 1997.

[13] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, 1999. 2nd edition.

[14] T. Feder and M. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal on Computing*, 28:57–104, 1999.

[15] S. Felsner, P. C. Fishburn, and W. T. Trotter. Finite three dimensional partial orders which are not sphere orders. *Discrete Math.*, 201:101–132, 1999.

[16] Garey and Johnson. *A Guide to NP-completeness.* CSLI Press, Stanford, 1978.

[17] M. C. Golumbic and R. Shamir. Complexity and algorithms for reasoning about time: a graph-theoretic approach. *Journal of the ACM (JACM)*, 40(5):1108 – 1133, 1933.

[18] R. L. Graham, B. L. Rothschild, and J. H. Spencer. *Ramsey theory.* Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., 1990. Second edition.

[19] P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48:92–110, 1990.

[20] W. Hodges. *A shorter model theory.* Cambridge University Press, 1997.

[21] S. Janson, T. Luczak, and A. Rucinski. *Random Graphs.* John Wiley and

Sons, 2000.

[22] P. Jeavons, P. Jonsson, and A. A. Krokhin. Reasoning about temporal relations: The tractable subalgebras of Allen's interval algebra. *Journal of the ACM*, 50(5):591–640, 2003.

[23] H. Kautz, P. van Beek, and M. Vilain. Constraint propagation algorithms: A revised report. *Qualitative Reasoning about Physical Systems*, pages 373–381, 1990.

[24] P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proceedings of PODS'98*, pages 205–213, 1998.

[25] M. Koubarakis. Tractable disjunctions of linear constraints: Basic results and applications to temporal reasoning. *Theoretical Computer Science*, 266:311–339, 2001.

[26] P. B. Ladkin and R. D. Maddux. On binary constraint problems. *Journal of the Association for Computing Machinery*, 41(3):435–469, 1994.

[27] D. Marker. *Model Theory: An Introduction*. Springer, 2002.

[28] A. Szendrei. *Clones in universal Algebra*. Seminaire de mathematiques superieures. Les Presses de L'Universite de Montreal, 1986.

[29] P. van Beek. Reasoning about qualitative temporal information. *Artificial Intelligence*, 58:297–326, 1992.

[30] R. van der Meyden. The complexity of querying indefinite information about linearly ordered domains. *Journal of Computer and Systems Science*, 54(1):113–135, 1997.