

# The Complexity of Temporal Constraint Satisfaction Problems

Manuel Bodirsky and Jan Kára

## Abstract

A *temporal constraint language* is a set of relations that has a first-order definition in  $(\mathbb{Q}; <)$ , the dense linear order of the rational numbers. We present a complete complexity classification of the constraint satisfaction problem (CSP) for temporal constraint languages: if the constraint language is contained in one out of nine temporal constraint languages, then the CSP can be solved in polynomial time; otherwise, the CSP is NP-complete. Our proof combines model-theoretic concepts with techniques from universal algebra, and also applies the so-called product Ramsey theorem, which we believe will be useful in similar contexts of constraint satisfaction complexity classification.

An extended abstract of this paper appeared in the proceedings of STOC'08.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Preliminaries</b>	<b>2</b>
2.1	Model-theoretic preliminaries . . . . .	2
2.2	Camerons theorem . . . . .	3
2.3	The constraint satisfaction problem . . . . .	4
2.4	Universal-algebraic preliminaries . . . . .	5
2.5	Polymorphism clones of temporal constraint languages . . . . .	6
2.6	Hard temporal CSPs . . . . .	7
<b>3</b>	<b>Endomorphisms</b>	<b>8</b>
<b>4</b>	<b>Shuffle closed languages</b>	<b>11</b>
4.1	Shuffle closure . . . . .	11
4.2	Operations providing min-union closure . . . . .	13
4.3	Operations providing min-intersection closure . . . . .	14
4.4	Operations providing min-xor closure . . . . .	16
4.5	Operations generating $min$ , $mi$ , $mx$ . . . . .	18
<b>5</b>	<b>Algorithms for shuffle-closed languages</b>	<b>21</b>
5.1	An algorithm for languages preserved by $min$ . . . . .	23
5.2	An algorithm for languages preserved by $mi$ . . . . .	24
5.3	An algorithm for languages preserved by $mx$ . . . . .	25
<b>6</b>	<b>The complete classification</b>	<b>25</b>
6.1	The operations $lex$ and $ll$ . . . . .	25
6.2	Operations generating $ll$ , $dual-ll$ , or $lex$ . . . . .	27
6.3	The product Ramsey theorem . . . . .	28
6.4	Proof of the main result . . . . .	30
<b>7</b>	<b>Concluding remarks</b>	<b>33</b>

# 1 Introduction

A constraint satisfaction problem is a computational problem where the task is, informally, to decide for a given set of variables and constraints on the variables whether there exists an assignment of values to the variables that satisfies all constraints. Such problems appear naturally and frequently in most areas of computer science, for example in Artificial Intelligence, Scheduling, Computational Linguistics, Computational Biology, Combinatorial Optimization, Operations Research, Computer Algebra, and Programming Languages.

We study the computational complexity of a large class of constraint satisfaction problems where the variables denote *time points*. Formally, by a *temporal constraint language*  $\Gamma$  we mean a relational structure  $(\mathbb{Q}; R_1, R_2, \dots)$  where each  $R_i$  has a first-order definition in  $(\mathbb{Q}; <)$ , the rational numbers with the dense linear order. A temporal constraint language is *finite* if it contains finitely many relations. The *constraint satisfaction problem (CSP)* for a finite temporal constraint language  $\Gamma$  is the computational problem to decide for a first-order sentence  $\Phi$  of the form

$$\exists x_1, \dots, x_n. \psi_1 \wedge \dots \wedge \psi_p,$$

where  $\psi_1, \dots, \psi_p$  are atomic formulas of the form  $R(x_{i_1}, \dots, x_{i_k})$ , whether  $\Phi$  is true or false in  $\Gamma$ .

Several famous polynomial-time solvable and NP-complete problems can be formulated as temporal CSPs. For example, if  $\Gamma$  equals  $(\mathbb{Q}; Betw)$  where *Betw* is the ternary relation  $Betw = \{(x, y, z) \in \mathbb{Q}^3 \mid (x < y < z) \vee (z < y < x)\}$ , then the corresponding CSP becomes the Betweenness problem listed in the book of Garey and Johnson [24]. Similarly, the Cyclic Ordering problem [24] can be formulated as the CSP for  $(\mathbb{Q}; \{\mathbb{Q}^3 \mid (x < y < z) \vee (y < z < x) \vee (z < x < y)\})$ .

One of the most fundamental temporal constraint languages is  $(\mathbb{Q}; \leq, <, \neq, =)$ . The CSP for this constraint language is known as the *network consistency problem of the Point Algebra* in Artificial Intelligence, and is easily seen to be tractable [41]. Even simpler is the CSP for  $(\mathbb{Q}; <)$  itself, which can be viewed as the graph acyclicity problem for directed graphs. A considerably larger temporal constraint language is the set of Ord-Horn relations, introduced by Nebel and Bürkert [37]. The CSP for Ord-Horn relations can be solved by resolution in polynomial time. Another temporal constraint language that is known to be tractable is the class of AND/OR precedence constraints in scheduling [36]; also see [5]. It has been asked in [27] whether a subclass of temporal CSPs called *ordering CSPs* can be classified with respect to their computational complexity. Satisfiability thresholds for *random instances* of ordering CSPs have been studied in [26].

The class of temporal constraint languages is of fundamental importance for infinite domain constraint satisfaction, since CSPs for such languages appear as important special cases in several other classes of CSPs that have been studied, e.g., constraint languages about branching time, partially ordered time, spatial reasoning, and set constraints [9, 22, 32]. Moreover, several polynomial-time solvable classes of constraint languages on *time intervals* [21, 34, 37] can be solved by translation into polynomial-time solvable temporal constraint languages.

In this paper we present a complete classification of the computational complexity of the constraint satisfaction problem for temporal constraint languages. The CSP can be solved in polynomial time if the temporal constraint language  $\Gamma$  is contained in at least one out of nine temporal constraint languages; otherwise, at least one out of six specific temporal constraint satisfaction problems can be expressed by  $\Gamma$ , and the problem is NP-complete. Two of the polynomial-time solvable temporal languages properly contain all Ord-Horn relations [5].

A similar classification result was obtained by Schaefer [39] for Boolean constraint languages, i.e., relational structures over a two element set. Schaefer showed that the CSP for a Boolean constraint language is tractable if the language is contained in one out of six Boolean constraint languages; otherwise the CSP is NP-complete.

The question whether such a complexity dichotomy holds for all constraint languages over a *finite* domain [23] is one of the major open research problems in constraint satisfaction complexity. In the last decade, a strong connection of this problem to central and deep questions in universal algebra has stimulated further activity [10–14, 20, 29, 30]; the starting point of this connection is

the observation that the complexity of the CSP is fully described by the so-called polymorphisms of the constraint language.

The techniques that we apply to study temporal CSPs take their impetus from this algebraic approach. In order to use polymorphisms for constraint languages over  $\mathbb{Q}$ , we need fundamental concepts from model theory, as in [1,3,6]. Another important ingredient is Cameron’s classification of temporal constraint languages: up to first-order interdefinability, there are exactly five different temporal constraint languages. Two structures that are first-order interdefinable might have CSPs of different computational complexity (this is why we need polymorphisms and universal algebra), but still Cameron’s result turns out to be useful in our proof of the complexity dichotomy. Finally, in the combinatorial part of the proofs we also apply the so-called product Ramsey theorem; we believe that Ramsey theory can be applied in a similar way for complexity classifications of other classes of constraint languages over infinite domains.

## 2 Preliminaries

### 2.1 Model-theoretic preliminaries

A temporal constraint language  $\Gamma = (\mathbb{Q}; R_1, R_2, \dots)$  is a structure with a first-order definition in  $(\mathbb{Q}; <)$ , the dense linear order of the rational numbers. That is, for every relation  $R_i$  of  $\Gamma$  of arity  $k_i$  there is a first-order formula  $\phi(x_1, \dots, x_{k_i})$  with  $k_i$  free variables  $x_1, \dots, x_{k_i}$  that defines  $R_i$  over  $(\mathbb{Q}; <)$  in the usual way. Relations with a first-order definition in  $(\mathbb{Q}; <)$  will also be called *temporal relations*.

Temporal constraint languages have remarkable model-theoretical properties, which are important in this paper. The *first-order theory* of a relational structure is the set of all first-order sentences that are true in  $\Gamma$ .

**Definition 1.** *A relational structure  $\Gamma$  over a countable domain is called  $\omega$ -categorical if all countable models of the first order theory of  $\Gamma$  are isomorphic to  $\Gamma$ .*

The structure  $(\mathbb{Q}; <)$  is  $\omega$ -categorical (see e.g. [16,28]); this is due to Cantor [17].

**Lemma 1** (see e.g. [28]). *If  $\Gamma$  is  $\omega$ -categorical, and  $\Delta$  has a first-order definition in  $\Gamma$ , then  $\Delta$  is also  $\omega$ -categorical.*

As a consequence, all temporal constraint languages are  $\omega$ -categorical. There is also an algebraic characterization of  $\omega$ -categoricity. An *automorphism* of a relational structure  $\Gamma$  is an isomorphism between  $\Gamma$  and  $\Gamma$ . The set of all automorphisms of  $\Gamma$  forms a permutation group  $\text{Aut}(\Gamma)$  on the domain  $D$  of  $\Gamma$ . The *orbit of a  $k$ -tuple*  $(t_1, \dots, t_k)$  over  $D$  is the set  $\{(\alpha(t_1), \dots, \alpha(t_k)) \mid \alpha \in \text{Aut}(\Gamma)\}$ . A permutation group is called *oligomorphic* if for every  $k \geq 1$  there is only a finite number of distinct orbits of  $k$ -tuples over  $D$ . The following was independently shown by Engeler, Svenonius, and Ryll-Nardzewski, and is of fundamental importance for the universal-algebraic approach to constraint satisfaction.

**Theorem 2** (see e.g. [16,28,35]). *Let  $\Gamma$  be a relational structure. Then the following are equivalent:*

1.  $\Gamma$  is  $\omega$ -categorical.
2. The automorphism group of  $\Gamma$  is oligomorphic.
3. For all  $k$ , there are only finitely many inequivalent first-order formulas with  $k$  free variables.
4. Every  $k$ -ary first-order definable relation in  $\Gamma$  is the union of a finite number of orbits of  $k$ -tuples of the automorphism group of  $\Gamma$ .
5. A relation is first-order definable over  $\Gamma$  if and only if it is preserved by the automorphisms of  $\Gamma$ .

**Corollary 3.** *A relation  $R \subseteq \mathbb{Q}^k$  is temporal if and only if it is preserved by  $\text{Aut}(\mathbb{Q}; <)$ .*

*Proof.* Since  $(\mathbb{Q}; <)$  is  $\omega$ -categorical, this follows directly from Item 5 of Theorem 2.  $\square$

Another important model-theoretic concept is *homogeneity*: a relational structure  $\Gamma$  is called *homogeneous*<sup>1</sup> if every isomorphism between induced substructures of  $\Gamma$  can be extended to an automorphism of  $\Gamma$ . It is well-known that the structure  $(\mathbb{Q}; <)$  is homogeneous [28].

## 2.2 Camerons theorem

In this subsection we recall the classical result of Cameron [15] that describes temporal constraint languages *up to first-order interdefinability*. We say that two structures  $\Gamma$  and  $\Delta$  are *first-order interdefinable* if  $\Gamma$  has a first-order definition in  $\Delta$ , and  $\Delta$  has a first-order definition in  $\Gamma$ .

**Theorem 4** (Relational version of Camerons theorem; see e.g. [33]). *Let  $\Gamma$  be a temporal constraint language. Then  $\Gamma$  is first-order interdefinable with exactly one out of the following five homogeneous structures.*

- The dense linear order  $(\mathbb{Q}; <)$  itself,
- The structure  $(\mathbb{Q}; \text{Betw})$ , where *Betw* is the ternary relation  $\{(x, y, z) \in \mathbb{Q}^3 \mid (x < y < z) \vee (z < y < x)\}$
- The structure  $(\mathbb{Q}; \text{Cycl})$ , where *Cycl* is the ternary relation  $\{(x, y, z) \mid (x < y < z) \vee (y < z < x) \vee (z < x < y)\}$ ,
- The structure  $(\mathbb{Q}; \text{Sep})$ , where *Sep* is the 4-ary relation  $\{(x_1, y_1, x_2, y_2) \mid (x_1 < x_2 < y_1 < y_2) \vee (x_1 < y_2 < y_1 < x_2) \vee (y_1 < x_2 < x_1 < y_2) \vee (y_1 < y_2 < x_1 < x_2) \vee (x_2 < x_1 < y_2 < y_1) \vee (x_2 < y_1 < y_2 < x_1) \vee (y_2 < x_1 < x_2 < y_1) \vee (y_2 < y_1 < x_2 < x_1)\}$ ,
- The structure  $(\mathbb{Q}; =)$ .

The relation *Sep* is the so-called *separation relation*; note that  $\text{Sep}(x_1, y_1, x_2, y_2)$  holds for elements  $x_1, y_1, x_2, y_2 \in \mathbb{Q}$  iff all four points  $x_1, y_1, x_2, y_2$  are distinct and the smallest interval over  $\mathbb{Q}$  containing  $x_1, y_1$  properly overlaps with the smallest interval containing  $x_2, y_2$  (where properly overlaps means that the two intervals have a non-empty intersection, but one interval does not contain the other).

Temporal constraint languages naturally arise in the theory of infinite permutation groups, because these structures are precisely the structures with a largest possible degree of symmetry on its subsets, which is formalized with the following definition. A subset of  $D$  is called a *k-subset* if it has cardinality  $k$ ; the *orbit of a k-set S* is the set  $\{\alpha(S) \mid \alpha \in \text{Aut}(\Gamma)\}$  where  $\alpha(S)$  is the image of the set  $S$  under  $\alpha$ .

**Definition 2.** *A structure  $\Gamma$  is called highly set-transitive if for all  $k \geq 1$  the structure  $\Gamma$  has precisely one orbit of k-sets.*

The next theorem was also shown by Cameron [15], and was his original motivation for the investigation of structures with a first-order definition in  $(\mathbb{Q}; <)$ . It will not be used in the results we present; however, we would like to state it here because it provides a fundamentally different characterization of the class of temporal constraint languages.

**Theorem 5.** *A relational structure  $\Gamma$  is highly set-transitive if and only if it is a temporal constraint language.*

---

<sup>1</sup>Sometimes also *ultra-homogeneous*, to distinguish it from other notions of homogeneity in model theory.

### 2.3 The constraint satisfaction problem

A first-order formula is called *primitive positive* if it is of the form

$$\exists x_1, \dots, x_n. \psi_1 \wedge \dots \wedge \psi_p,$$

where each formula  $\psi_1, \dots, \psi_p$  is atomic, i.e., of the form  $R(x_{i_1}, \dots, x_{i_k})$ , of the form  $x_{i_1} = x_{i_2}$ , or *false*.

The constraint satisfaction problem for a constraint language  $\Gamma = (\mathbb{Q}; R_1, \dots, R_s)$  with finitely many relations is the following computational problem, denoted by  $\text{CSP}(\Gamma)$ . We are given a primitive positive sentence  $\Phi$  (i.e., a primitive positive formula without free variables) where all relation symbols are relation symbols for the relations in  $\Gamma$ , and the question is whether  $\Phi$  is true in  $\Gamma$ . When  $\Gamma$  is  $(\mathbb{Q}; R)$ , we also write  $\text{CSP}(R)$  instead of  $\text{CSP}((\mathbb{Q}; R))$ .

The conjuncts  $\psi_1, \dots, \psi_p$  of an instance  $\Phi$  of the CSP are also called *constraints*. Hence, in this paper a constraint is a syntactic object (an atomic formula). Note that an instance of the CSP is fully described by its set of constraints. It will later be notationally convenient to treat  $\Phi$  as a set of constraints. The set of variables that appears in a first-order formula  $\Phi$  is denoted by  $V(\Phi)$ . A *solution* for an instance  $\Phi$  of the CSP is a mapping  $s : V(\Phi) \rightarrow \mathbb{Q}$  that satisfies all the constraints of  $\Phi$ .

Note that for finite relational signatures the choice of the way in which the relation symbols are represented in the input does not affect the computational complexity of the problem. This is different when we consider infinite constraint languages  $\Gamma$ . For infinite constraint languages (i.e., for structures with an infinite relational signature), we fix a way how to represent the relation symbols in the input, and then define the constraint satisfaction problem in the same way. However, the computational complexity of  $\text{CSP}(\Gamma)$  might now depend on the choice of this representation. Therefore, we follow a convention from finite domain constraint satisfaction and say that  $\text{CSP}(\Gamma)$  is (*locally*) *tractable* if all reducts of  $\Gamma$  with a finite signature can be solved in polynomial time [14]. Similarly, we say that  $\text{CSP}(\Gamma)$  is in NP if all reducts of  $\Gamma$  with a finite signature are in NP. Clearly, these two concepts do *not* depend on the choice of the representation of the relation symbols in the input.

We would like to remark that there are natural ways to represent temporal relations for infinite constraint languages such that the algorithmic results presented in this paper still hold when we use these representations; these representations are discussed in detail in [5]. In the constraint satisfaction literature, constraint satisfaction problems that can be solved in polynomial time under these representations of the input instances are called *globally tractable*. The necessary modifications of the algorithms to deal with infinite constraint languages are not difficult, but complicate the presentation, and are therefore omitted in this paper.

**Proposition 6.** *For all temporal constraint languages  $\Gamma$  the problem  $\text{CSP}(\Gamma)$  is in NP.*

*Proof.* Let  $\Phi$  be an instance of  $\text{CSP}(\Gamma)$ . Note that whether or not a mapping  $s$  from  $V(\Phi)$  to  $\mathbb{Q}$  is a solution for  $\Phi$  only depends on the weak linear order (i.e., the linear preorder)  $\preceq$  defined on the variables  $V(\Phi)$  by  $(x \preceq y) \Leftrightarrow (s(x) \leq s(y))$ . Clearly, it is possible to verify in deterministic polynomial time whether a given weak linear order on  $V(\Phi)$  is the weak linear order of a solution to  $\Phi$ .  $\square$

The following is an essential tool to establish hardness results for the CSP. A  $k$ -ary relation is called *primitive positive definable* if there exists a primitive positive formula with  $k$  free variables that defines  $R$ . Lemma 7 says that primitive positive definable relations can be ‘simulated’ in constraint satisfaction problems. For finite  $D$ , this can be found in [14], and its (easy) proof also works for infinite  $D$ .

**Lemma 7.** *Let  $\Gamma = (D; R_1, R_2, \dots)$  be a constraint language, and let  $R$  be a relation that has a primitive positive definition in  $\Gamma$ . Then  $\text{CSP}(\Gamma)$  and  $\text{CSP}(D; R, R_1, R_2, \dots)$  are polynomial-time equivalent.*

By  $\langle \Gamma \rangle$  we denote the set of all temporal relations that are primitive positive definable in  $\Gamma$ . Lemma 7 implies that the computational complexity of  $\text{CSP}(\Gamma)$  only depends on  $\langle \Gamma \rangle$  (up to polynomial-time reducibility).

## 2.4 Universal-algebraic preliminaries

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 [14]).

When  $t = (e_1, \dots, e_m) \in D^m$  is an  $m$ -tuple over  $D$  we write  $t[i]$  for the  $i$ -th entry  $e_i$  of  $t$ . Let  $t_1, \dots, t_k$  be  $m$ -tuples over  $D$ , and let  $f$  be a function  $f : D^k \rightarrow D$ . Then we write  $f(t_1, \dots, t_k)$  for the tuple obtained from the tuples  $t_1, \dots, t_k$  by applying  $f$  componentwise, i.e., for the  $m$ -tuple  $(f(t_1[1], \dots, t_k[1]), \dots, f(t_1[m], \dots, t_k[m]))$ .

We say that a  $k$ -ary function (also called *operation*)  $f : D^k \rightarrow D$  *preserves* an  $m$ -ary relation  $R \subseteq D^m$  iff for all  $t_1, \dots, t_k \in R$  the tuple  $f(t_1, \dots, t_k)$  is also contained in  $R$ . If an operation  $f$  does not preserve a relation  $R$ , we say that  $f$  *violates*  $R$ . If  $f$  preserves all relations of a constraint language  $\Gamma$ , we say that  $f$  is a *polymorphism* of  $\Gamma$  (it is also common to say that  $\Gamma$  is *closed under*  $f$ ). Note that the automorphisms of  $\Gamma$  are bijective unary polymorphisms that preserve all relations and their complements. A unary polymorphism of  $\Gamma$  is also called an *endomorphism* of  $\Gamma$ .

The set of all polymorphisms  $\text{Pol}(\Gamma)$  of a relational structure forms an algebraic object called a *clone* [40], which is a set of operations defined on a set  $D$  that is closed under composition and that contains all projections. Moreover,  $\text{Pol}(\Gamma)$  is also closed under interpolation (see Proposition 1.6 in [40]): 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  $D$  there is some operation  $g \in F$  such that  $f(a) = g(a)$  for every  $a \in A^k$ . We say that  $F$  *locally generates* an operation  $g$  if  $g$  is in the smallest clone that is closed under interpolation and contains all operations in  $F$ . For a set  $F$  of operations defined on a set  $D$  the set of all relations over  $D$  that are preserved by all operations in  $F$  is denoted by  $\text{Inv}(F)$ .

**Proposition 8** (Corollary 1.9 in [40]).  *$F$  locally generates  $g$  if and only if  $g$  preserves all relations in  $\text{Inv}(F)$ .*

Polymorphism clones can be used to characterize primitive positive definability over a finite structure, by a result of [8] and [25]. In general, this is not true for infinite structures [6]. However, the result remains true if the relational structure is  $\omega$ -categorical.

**Theorem 9** (of [6]). *Let  $\Gamma$  be an  $\omega$ -categorical constraint language. Then the relations preserved by the polymorphisms of  $\Gamma$  are precisely those that have a primitive positive definition in  $\Gamma$ , in symbols,  $\text{Inv}(\text{Pol}(\Gamma)) = \langle \Gamma \rangle$ .*

The following lemma holds for arbitrary relational structures  $\Gamma$ .

**Lemma 10.** *Let  $\Gamma$  be a relational structure and let  $R$  be a  $k$ -ary relation that is a union of  $l$  orbits of  $k$ -tuples of  $\text{Aut}(\Gamma)$ . If  $R$  is violated by a polymorphism  $g$  of  $\Gamma$  of arity  $m \geq l$ , then  $R$  is also violated by an  $l$ -ary polymorphism of  $\Gamma$ .*

*Proof.* Since  $R$  is violated by  $g$  there are  $k$ -tuples  $t_1, \dots, t_m$  from  $R$  such that  $g(t_1, \dots, t_m)$  is not in  $R$ . Since  $R$  is the union of  $l$  orbits of  $k$ -tuples of  $\text{Aut}(\Gamma)$  and  $l < m$ , there are tuples in  $t_1, \dots, t_m$  that are from the same orbit. In particular, we can without loss of generality assume that for all tuples  $t_{l+1}, \dots, t_m$  there is a tuple from  $t_1, \dots, t_l$  that is in the same orbit. For  $l + 1 \leq j \leq m$ , let  $i_j \leq l$  be an index such that  $t_{i_j}$  and  $t_j$  are in the same orbit. Then there are automorphisms  $\alpha_{l+1}, \dots, \alpha_m$  of  $\Gamma$  such that  $\alpha_j(t_{i_j}) = t_j$ . Therefore the  $l$ -ary operation  $f$  defined as  $f(x_1, \dots, x_l) := g(x_1, \dots, x_l, \alpha_{l+1}(x_{i_{l+1}}), \dots, \alpha_m(x_{i_m}))$  is a polymorphism of  $\Gamma$  and also violates  $R$ .  $\square$

We state an easy corollary of Theorem 9 and Lemma 10.

**Corollary 11.** *Suppose there is no primitive positive definition of  $<$  in a temporal constraint language  $\Gamma$ . Then  $\Gamma$  has a endomorphism that violates  $<$ .*

## 2.5 Polymorphism clones of temporal constraint languages

In this paper, we always deal with polymorphism clones of temporal constraint languages. Thus it is convenient to make the following convention. We say that a set of operations  $F$  generates an operation  $g$  if  $F$  together with all automorphisms of  $(\mathbb{Q}; <)$  locally generates  $g$ . In case that  $F$  contains just one operation  $f$ , we also say that  $f$  generates  $g$ .

**Lemma 12.** *An operation  $f$  generates  $g$  if and only if every temporal relation that is preserved by  $f$  is also preserved by  $g$ .*

*Proof.* By definition,  $f$  generates  $g$  if and only if  $F = \{f\} \cup \text{Aut}(\mathbb{Q}; <)$  locally generates  $g$ . Proposition 8 shows that this is the case if and only if  $g$  preserves all relations in  $\text{Inv}(F)$ . Since a relation is preserved by  $\text{Aut}(\mathbb{Q}; <)$  if and only if it is a temporal relation, we find that  $g$  preserves all relations in  $\text{Inv}(F)$  if and only if  $g$  preserves all temporal relations preserved by  $f$ , which is what we had to show.  $\square$

We now present an equivalent description of the polymorphism clone of a temporal constraint language that is generated by a single operation. A  $k$ -ary operation  $f$  on  $\mathbb{Q}$  defines a weak linear order  $\preceq$  on  $\mathbb{Q}^k$ , as follows: for  $x, y \in \mathbb{Q}^k$ , let  $x \preceq y$  iff  $f(x) \leq f(y)$ . The following observation easily follows from the properties of  $\text{Aut}(\mathbb{Q}; <)$ .

**Observation 1.** *Let  $f$  and  $g$  be two  $k$ -ary operations that define the same weak linear order on  $\mathbb{Q}^k$ . Then  $f$  generates  $g$  and  $g$  generates  $f$ .*

We now define fundamental operations on  $\mathbb{Q}$ . The unary operation  $-$  is defined as  $-(x) := -x$  in the usual sense. Let  $c$  be any irrational number, and let  $e$  be any order-preserving bijection between  $(-\infty, c)$  and  $(c, \infty)$ . Then the operation  $cyc$  is defined by  $e(x)$  for  $x < c$  and by  $e^{-1}(x)$  for  $x > c$ . With these operations and the notion of generation, Cameron's theorem can be rephrased as follows.

**Theorem 13** (Operational version of Camerons theorem; see e.g. [33]). *Let  $\Gamma$  be a temporal constraint language. Then exactly one of the following holds.*

- *The automorphisms of  $\Gamma$  are the permutations generated by  $(\mathbb{Q}; <)$ ;*
- *The automorphisms of  $\Gamma$  are the permutations generated by  $-$ ;*
- *The automorphisms of  $\Gamma$  are the permutations generated by  $cyc$ ;*
- *The automorphisms of  $\Gamma$  are the permutations generated by  $-$  and  $cyc$ ;*
- *All permutations of  $\mathbb{Q}$  are automorphisms of  $\Gamma$ .*

If  $f$  is a  $k$ -ary operation on  $\mathbb{Q}$ , then the operation  $-f(-x_1, \dots, -x_k)$  is called the *dual* of  $f$ . Note that if  $f$  preserves an  $m$ -ary relation  $R$ , then the dual of  $f$  preserves the relation  $-R$ , which is defined to be the relation  $\{(-t_1, \dots, -t_m) \mid (t_1, \dots, t_m) \in R\}$ . Clearly, if  $\text{CSP}(\mathbb{Q}; R_1, R_2, \dots)$  is tractable (is NP-complete), then also  $\text{CSP}(\mathbb{Q}; -R_1, -R_2, \dots)$  is tractable (is NP-complete, respectively).

## 2.6 Hard temporal CSPs

In this subsection we discuss various important NP-complete temporal constraint satisfaction problems. We have already mentioned in the introduction that the Betweenness and the Cyclic Ordering problem in [24] can be formulated as temporal CSPs, and that these problems are NP-complete. The corresponding relations *Betw* and *Cycl* re-appeared in Cameron's theorem (Theorem 4). Another relation that appeared in Theorem 4 is the separation relation *Sep*. The corresponding CSP is again NP-complete.

**Proposition 14.** *Let  $\Gamma$  be any constraint language such that *Sep* has a primitive positive definition in  $\Gamma$ . Then  $\text{CSP}(\Gamma)$  is NP-hard.*

*Proof.* By Lemma 7, it is enough to prove NP-hardness of  $\text{CSP}(\text{Sep})$ , which we do by reduction from the problem  $\text{CSP}(\text{Betw})$  listed in [24] as an NP-complete problem. So assume we are given an instance  $\Phi$  of  $\text{CSP}(\text{Betw})$ . We create an instance  $\Psi$  of  $\text{CSP}(\text{Sep})$  as follows. The set of variables is  $V(\Phi) \cup \{z\}$ , where  $z$  is a new variable. For each constraint  $\phi \in \Phi$  imposed on the variables  $x_1, x_2, x_3 \in V(\Phi)$  we include the constraint  $\text{Sep}(z, x_2, x_1, x_3)$  to  $\Psi$ . It is obvious that the transformation can be performed in polynomial time.

We have to verify that  $\Psi$  has a solution if and only if  $\Phi$  has a solution. If  $\Phi$  has a solution  $t$ , then for any constraint  $\phi \in \Phi$  imposed on the variables  $x_1, x_2, x_3$  either  $t(x_1) < t(x_2) < t(x_3)$  or  $t(x_3) < t(x_2) < t(x_1)$ . Therefore, if we extend  $t$  by mapping  $z$  to a value smaller than all the values of  $t$ , all constraints in  $\Psi$  are satisfied.

For the other implication, suppose that  $\Psi$  has a solution. Because  $\text{Sep}$  is preserved by  $\text{cyc}$ , it also has a solution  $t$  in which  $z$  gets the minimal value among all variables of  $\Psi$ . Now, consider a constraint  $\psi \in \Phi$  imposed on the variables  $z, x_2, x_1, x_3$ . Because  $z$  has the minimal value in  $t$  either  $t(x_1) < t(x_2) < t(x_3)$  or  $t(x_3) < t(x_2) < t(x_1)$ , and therefore the corresponding constraint in  $\Phi$  is satisfied. Hence,  $t$  restricted to  $V(\Phi)$  is a solution of  $\Phi$ .  $\square$

An important relation for our classification is the relation  $S$ , defined as follows.

**Definition 3.** Let  $S$  be the ternary relation  $\{(x, y, z) \in \mathbb{Q}^3 \mid x = y < z \vee x = z < y\}$ .

This relation has an NP-complete constraint satisfaction problem.

**Proposition 15.** Let  $\Gamma$  be any constraint language such that  $S$  has a primitive positive definition in  $\Gamma$ . Then  $\text{CSP}(\Gamma)$  is NP-hard.

*Proof.* By Lemma 7, it is enough to show that  $\text{CSP}(S)$  is NP-hard. We reduce the problem *positive 1-IN-3-3SAT* [24] to the problem  $\text{CSP}(S)$ . Let  $F$  be an input formula for 1-in-3-3SAT with variables  $x_1, \dots, x_n$ . We create the following instance  $\Phi$  of  $\text{CSP}(S)$ :

- We introduce a new variable  $a$ .
- For each variable  $x_i$  of  $F$ , we introduce two variables  $v_i^1, v_i^2$  and a constraint  $S(a, v_i^1, v_i^2)$ .
- For each clause  $C$  of  $F$  with variables  $x_i, x_j, x_k$ , we add a new variable  $v_C$  and introduce the constraints  $S(v_C, v_i^1, v_j^1)$  and  $S(a, v_C, v_k^1)$ .

We show that  $\Phi$  has a solution if and only if  $F$  is 1-in-3 satisfiable. For the implication from right to left suppose we have some 1-in-3 satisfying truth assignment  $s$ . We assign the value 0 to  $a$  and for each variable  $x_i$  of  $F$ , we assign 0 to  $v_i^1$  and 1 to  $v_i^2$  if  $x_i$  is true in  $s$ , and  $i + 1$  to  $v_i^1$  and 0 to  $v_i^2$  if  $x_i$  is false in  $s$ . Clearly, the constraints  $S(a, v_i^1, v_i^2)$  are satisfied for all  $i \in \{1, \dots, n\}$ . For each clause  $C$  containing variables  $x_i, x_j, x_k$ , exactly one of variables is true in  $s$ . If it is  $x_k$ , we assign  $\min(i + 1, j + 1)$  to  $v_C$  and see that both constraints  $S(v_C, v_i^1, v_j^1)$  and  $S(a, v_C, v_k^1)$  are satisfied (as  $v_k^1$  is assigned 0). If it is  $x_i$  or  $x_j$ , we assign 0 to  $v_C$  and again see that both constraints are satisfied.

For the implication from left to right, let  $s$  be a solution to  $\Phi$ . Without loss of generality we can assume that  $s(a) = 0$  (otherwise we can apply an appropriate automorphism of  $(\mathbb{Q}, <)$  to  $s$ ). Because of the constraints  $S(a, v_i^1, v_i^2)$ , exactly one of  $v_i^1, v_i^2$  is 0 and the other variable is greater than 0 for all  $i \in \{1, \dots, n\}$ . We set variable  $x_i$  to true if  $v_i^1$  is 0, otherwise we set  $x_i$  to false. Now, we have to check that there is exactly one variable set to true in each clause. Let  $C$  be some clause of  $F$  with variables  $x_i, x_j, x_k$ . First, we check that at most one variable is true in each clause. If  $x_k$  is true, then  $v_k^1$  is 0 and hence  $v_C$  must be greater than 0. Consequently, both  $v_i^1$  and  $v_j^1$  must be greater than 0 and so  $x_i$  and  $x_j$  are false. If  $x_i$  or  $x_j$  is true, then  $v_i^1$  or  $v_j^1$  must be 0, respectively. From the properties of  $S$  it follows that at most one of these variables is 0 (and thus at most one of  $x_i, x_j$  is true) and  $v_C$  must be 0 too. Hence,  $v_k^1$  must be greater than 0 and so  $x_k$  is false. What remains to be checked is that at least one variable is set to true in each clause. If all variables  $x_i, x_j, x_k$  are false, it means that  $v_i^1, v_j^1, v_k^1$  are all greater than zero. Therefore  $v_C$  is greater than zero and the constraint  $S(a, v_C, v_k^1)$  cannot be satisfied; a contradiction. So there

is exactly one variable set to true in each clause and so the specified truth assignment is 1-in-3 satisfying.  $\square$

Our results (see Corollary 51) will show that if no relation among *Betw*, *Cycl*, *Sep*, *S*,  $\neg S$ , or the relation  $\{(x, y, z) \in \mathbb{Q}^3 \mid x = y \neq z \vee x \neq y = z\}$  is primitive positive definable in a temporal constraint language  $\Gamma$ , then  $\text{CSP}(\Gamma)$  is tractable.

### 3 Endomorphisms

In this section we study the endomorphisms of temporal constraint languages. As an application, we obtain a reduction of the complexity classification for temporal constraint satisfaction problems to the classification for those languages that admit a primitive positive definition of the binary relation  $<$ .

A *self-embedding* of a relational structure  $\Gamma$  with domain  $D$  is an injective mapping  $f : D \rightarrow D$  such that  $f$  preserves the relations in  $\Gamma$  and their complements. We can also think of self-embeddings of  $\Gamma$  as isomorphisms between  $\Gamma$  and induced substructures of  $\Gamma$ . We will need the following result of [7].

**Theorem 16** (Theorem 5 in [7]). *A formula is equivalent to an existential positive (existential) formula over an  $\omega$ -categorical structure  $\Gamma$  if and only if the formula is preserved by all endomorphisms (self-embeddings) of  $\Gamma$ .*

To use Camerons result about automorphism groups of temporal constraint languages, we first clarify when the self-embeddings are determined by the automorphisms<sup>2</sup>.

**Proposition 17.** *Let  $\Gamma$  be a structure such that for every self-embedding  $e$  of  $\Gamma$  and every finite tuple  $\bar{a}$  there is a self-embedding  $f$  such that  $f(e(\bar{a})) = \bar{a}$ . Then the self-embeddings are generated by the automorphisms of  $\Gamma$ .*

*Proof.* Suppose for contradiction that the self-embeddings are not generated by the automorphisms of  $\Gamma$ . Then by Proposition 8 there must be a relation  $R$  that is preserved by all automorphisms but not preserved by the self-embeddings. Since  $\Gamma$  is  $\omega$ -categorical, Theorem 2 shows that  $R$  is first-order definable, and Theorem 16 shows that  $R$  is not existentially definable over  $\Gamma$ . Let  $\phi$  be a first-order definition of  $R$  in prenex normal form with a minimal number of quantifier blocks, and let  $\phi_0$  be the quantifier-free part of  $\phi$ .

We claim that then there is an existential formula  $\phi'$  that is not equivalent to a universal formula. If the innermost quantifier block of  $\phi$  is existential with variables  $x_1, \dots, x_k$ , then  $\phi' = \exists x_1, \dots, x_k. \phi_0$  cannot be equivalent to a universal formula  $\psi$  over  $\Gamma$ , otherwise we could replace the subformula  $\phi'$  in  $\phi$  by  $\psi$  and would obtain a formula that is equivalent to  $\phi$  but has fewer quantifier blocks. If the innermost quantifier block is universal with variables  $x_1, \dots, x_k$ , then either  $\phi' = \exists x_1, \dots, x_k. \neg \phi_0$  is an existential formula that is not equivalent to a universal formula, and the claim in the beginning of this paragraph is proved, or there exists a formula  $\psi$  equivalent to  $\phi'$  of the form  $\forall y_1, \dots, y_l. \psi_0$  where  $\psi_0$  is quantifier-free. If we then replace the subformula  $\forall x_1, \dots, x_k. \phi_0$  of  $\phi$  by  $\exists y_1, \dots, y_l. \neg \psi_0$  we obtain a formula that is equivalent to  $\phi$  but has fewer quantifier blocks. This shows the claim.

Since  $\neg \phi'$  is not equivalent to an existential formula, by Theorem 16 there must be a self-embedding  $e$  and a tuple  $\bar{a}$  such that  $\bar{a}$  satisfies  $\neg \phi'$  and  $e(\bar{a})$  satisfies  $\phi'$ . By assumption there exists a self-embedding  $f$  of  $\Gamma$  such that  $f(e(\bar{a})) = \bar{a}$ . Since  $f$  preserves the existential formula  $\phi'$ , we have that  $\bar{a}$  satisfies  $\phi'$ , contradiction.  $\square$

Note that all temporal constraint languages have only one orbit of 2-sets (Theorem 5). For structures with this property we can show that all endomorphisms are injective unless they have a constant endomorphism.

<sup>2</sup>We thank Markus Junker for the proof idea of this result.

**Lemma 18.** *Let  $\Gamma$  be such that  $\text{Aut}(\Gamma)$  has only one orbit of 2-sets. If  $\Gamma$  has a non-injective endomorphism  $f$ , then  $\Gamma$  also has a constant endomorphism.*

*Proof.* Let  $D$  be the domain of  $\Gamma$ , and let  $f$  be an endomorphism of  $\Gamma$  such that  $f(b) = f(b')$  for two distinct values  $b, b' \in D$ . Let  $a_1, a_2, \dots$  be an enumeration of  $D$ . We construct an infinite sequence of endomorphisms  $e_1, e_2, \dots$ , where  $e_i$  is an endomorphism that maps all of the values  $a_1, \dots, a_i$  to  $a_1$ . This suffices, since by local closure the mapping defined by  $e(x) := a_1$  for all  $x$  is an endomorphism of  $\Gamma$ .

For  $e_1$ , we take the identity map, which clearly is an endomorphism with the desired properties. To define  $e_i$  for  $i \geq 2$ , let  $\alpha$  be an automorphism of  $\Gamma$  that maps  $\{a_1, e_{i-1}(a_i)\}$  to  $\{b, b'\}$  (such an automorphism exists because of the assumption on  $\text{Aut}(\Gamma)$ ). Then the endomorphism  $f(\alpha(e_{i-1}(x)))$  is constant on  $a_1, \dots, a_i$  (recall that  $a_1 = e_{i-1}(a_1) = \dots = e_{i-1}(a_{i-1})$ ). It is known that  $\text{Aut}(\Gamma)$  has one orbit of 1-sets (the number of orbits of  $n$ -sets is not smaller than the number of orbits of  $(n-1)$ -sets; this is 3.1 in [16]), and hence there is also an automorphism  $\alpha'$  that maps  $f(b)$  to  $a_1$ . Then  $e_i(x) := \alpha'(f(\alpha(e_{i-1}(x))))$  is an endomorphism with the desired properties.  $\square$

**Proposition 19.** *Let  $\Gamma$  be a temporal constraint language. Then exactly one of the following cases applies.*

1.  $\Gamma$  has a constant endomorphism;
2. All endomorphisms of  $\Gamma$  preserve  $<$ ;
3. The set of endomorphisms of  $\Gamma$  equals the set of unary operations generated by  $-$ ;
4. The set of endomorphisms of  $\Gamma$  equals the set of unary operations generated by  $\text{cyc}$ ;
5. The set of endomorphisms of  $\Gamma$  equals the set of unary operations generated by  $-$  and  $\text{cyc}$ ;
6. The set of endomorphisms of  $\Gamma$  equals the set of all injective unary operations.

*Proof.* First note that all the cases are indeed disjoint: A constant endomorphism violates  $<$ , and cannot be generated by a set of injective unary operations; this shows that the first case is distinct from all others. Disjointness of the remaining cases follows from Theorem 13.

If  $\Gamma$  has a non-injective endomorphism, then Lemma 18 shows that there is also a constant endomorphism. Otherwise all endomorphisms of  $\Gamma$  are injective. We show that then all endomorphisms are self-embeddings. Suppose for contradiction that this is not the case, i.e., there is an endomorphism  $e$  and an atomic formula that is true on  $(e(a_1), \dots, e(a_l))$  but not true on  $(a_1, \dots, a_l)$  in  $\Gamma$ . Because  $e$  is injective, there is an  $\alpha \in \text{Aut}(\mathbb{Q}; <)$  such that  $\alpha(e(\{a_1, \dots, a_l\})) = \{a_1, \dots, a_l\}$ . For simplicity of notation, we write  $\alpha e$  for the function obtained as a composition of  $\alpha$  and  $e$ . Then  $(\alpha e)^l$ , i.e., the composition of  $(\alpha e) \dots (\alpha e)$  with  $l$ -factorial many terms of the form  $(\alpha e)$ , maps  $a_i$  to itself for all  $1 \leq i \leq l$ . But since the operation  $(\alpha e)^{l-1} \alpha$  is an endomorphism, we have that  $((\alpha e)^{l-1} \alpha)(e(a_1), \dots, e(a_l)) = (\alpha e)^l(a_1, \dots, a_l) = (a_1, \dots, a_l)$  satisfies the atomic formula as well, a contradiction.

In fact, the argument also shows that for any  $a_1, \dots, a_l \in \mathbb{Q}$  there exists a self-embedding  $f$  of  $\Gamma$  into  $\Gamma$  such that  $f(e(a_i)) = a_i$  for all  $i \in \{1, \dots, l\}$ . Since this holds for all endomorphisms  $e$  and in particular for all self-embeddings of  $\Gamma$ , Proposition 17 shows that the self-embeddings and hence the endomorphisms are generated by the automorphisms of  $\Gamma$ . Now the claim of the statement follows directly from Theorem 13.  $\square$

The following theorem shows that we can focus on constraint languages where  $<$  is primitive positive definable.

**Theorem 20.** *Let  $\Gamma$  be a temporal constraint language. Then it satisfies at least one of the following:*

- (a) *There is a primitive positive definition of  $\text{Cycl}$ ,  $\text{Betw}$ , or  $\text{Sep}$  in  $\Gamma$ .*
- (b)  *$\text{Pol}(\Gamma)$  contains a constant operation.*

(c)  $\text{Aut}(\Gamma)$  contains all permutations of  $\mathbb{Q}$ .

(d) There is a primitive positive definition of  $<$  in  $\Gamma$ . Moreover,  $\Gamma$  contains a binary operation that violates *Betw*.

*Proof.* If there is a primitive positive definition of *Betw* in  $\Gamma$  we are in case (a). Otherwise, since *Betw* consists of two orbits of triples of the automorphism group of  $(\mathbb{Q}; <)$ , Lemma 10 shows that there is a binary polymorphism of  $\Gamma$  that violates *Betw*. If there is a primitive positive definition of  $<$  in  $\Gamma$ , we are in case (d). Otherwise, again by Lemma 10, there is a unary polymorphism of  $\Gamma$  that violates  $<$ . Proposition 19 shows that  $\Gamma$  is preserved by a constant,  $-$ , or *cyc*. For each of these three operations we show the claim of the statement separately in the following three paragraphs.

If  $\Gamma$  is preserved by a constant we are in case (b), so we assume in the following that  $\Gamma$  is not preserved by a constant.

If  $\Gamma$  is preserved by  $-$ , the relation *Betw* consists of only one orbit of triples, and Lemma 10 shows that there is an endomorphism that violates *Betw*. Proposition 19 then implies that  $\Gamma$  is also preserved by *cyc*. Thus, the relation *Sep* consists of only one orbit of 4-tuples. Again, either *Sep* has a primitive positive definition, and we are in case (a), or there is an endomorphism that violates *Sep*. Proposition 19 now shows that  $\Gamma$  is preserved by all injective unary operations and we are in case (c).

If  $\Gamma$  is preserved by *cyc*, then the relation *Cycl* consists of only one orbit of triples. If *Cycl* has a primitive positive definition in  $\Gamma$ , we are in case (a). Otherwise, Lemma 10 shows that there is an endomorphism that violates *Cycl*. Proposition 19 then shows that  $\Gamma$  is also preserved by  $-$ . But the statement of the lemma has already been shown in the case that  $\Gamma$  is preserved by both  $-$  and *cyc* in the previous paragraph, so we are done.  $\square$

In case (a), the problem  $\text{CSP}(\Gamma)$  is NP-hard, as we have seen in Section 2.6. In case (b) it is easy to see that  $\text{CSP}(\Gamma)$  is trivial. In case (c) the complexity of  $\text{CSP}(\Gamma)$  has been classified in [4]. In the following, we therefore study only those temporal constraint languages where  $<$  is primitive positive definable.

## 4 Shuffle closed languages

One important subclass of temporal constraint languages are *shuffle closed* constraint languages. As we will see, there are NP-complete shuffle-closed constraint languages. However, in this section we present three additional restrictions for shuffle-closed constraint languages, and in Section 5 we present polynomial time algorithms that solve the corresponding CSPs.

From now on, we denote the set  $\{1, \dots, n\}$  also by  $[n]$ .

### 4.1 Shuffle closure

We define shuffle closure, and show how this property of temporal relations can also be described by preservation under a certain binary operation on  $\mathbb{Q}$ .

**Definition 4.** A  $k$ -ary relation  $R$  is called *shuffle closed* iff for all tuples  $t_1, t_2 \in R$  and all indices  $l \in [k]$  there is a tuple  $t_3 \in R$  such that for all  $i, j \in [k]$  we have  $t_3[i] \leq t_3[j]$  iff

- $t_1[i] \leq t_1[l]$  and  $t_1[i] \leq t_1[j]$ , or
- $t_1[l] < t_1[i]$ ,  $t_1[l] < t_1[j]$ , and  $t_2[i] \leq t_2[j]$ .

Let  $pp$  be an arbitrary binary operation on  $\mathbb{Q}$  such that  $pp(a, b) \leq pp(a', b')$  iff one of the following cases applies:

- $a \leq 0$  and  $a \leq a'$
- $0 < a$ ,  $0 < a'$ , and  $b \leq b'$ .

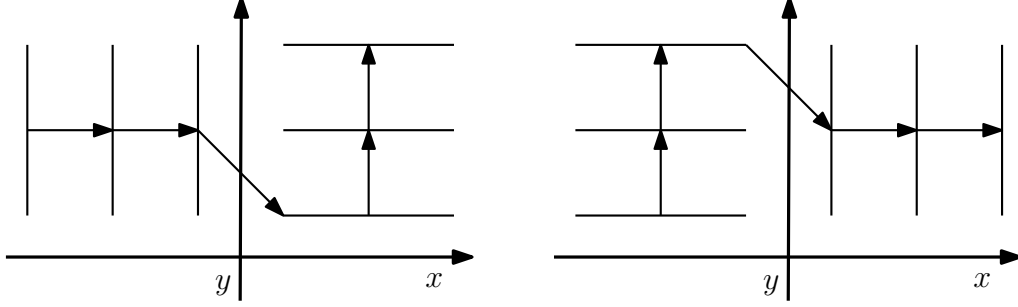


Figure 1: A visualization of  $pp$  (left) and  $dual\text{-}pp$  (right).

Clearly, such an operation exists. For an illustration, see the left diagram in Figure 1. In diagrams for binary operations  $f$  as in Figure 1, we draw a directed edge from  $(a, b)$  to  $(a', b')$  if  $f(a, b) < f(a', b')$ . Unoriented lines in rows and columns of picture for an operation  $f$  relate pairs of values that get the same value under  $f$ . The right diagram of Figure 1 is an illustration of the  $dual\text{-}pp$  operation. The name of the operation  $pp$  is derived from the word ‘*projection-projection*’, since the operation behaves as a projection to the first argument for negative first argument, and a projection to the second argument for positive first argument.

**Proposition 21.** *A temporal relation is shuffle-closed if and only if it is preserved by  $pp$ .*

*Proof.* Let  $R$  be a shuffle-closed relation, and let  $t_1$  and  $t_2$  be tuples from  $R$ . We want to show that  $t_3 = pp(t_1, t_2) \in R$ . If  $t_1$  only contains positive values, then there clearly exists an  $\alpha \in \text{Aut}(\mathbb{Q}; <)$  such that  $t_3 = \alpha(t_2)$ , and since  $R$  is preserved by the automorphisms of  $(\mathbb{Q}; <)$ , we are done. Otherwise, let  $l \in [k]$  be an index such that  $t_1[l]$  is the largest entry in  $t_1$  that is not positive. Because  $R$  is shuffle-closed, we know that there exists a tuple  $t'_3 \in R$  such that  $t'_3[i] \leq t'_3[j]$  iff  $(t_1[i] \leq t_1[l]$  and  $t_1[i] \leq t_1[j])$  or  $(t_1[l] < t_1[i], t_1[l] < t_1[j],$  and  $t_2[i] \leq t_2[j])$  for all  $i, j \in [k]$ . By the definition of  $pp$ , and the choice of  $l$ , the tuple  $t_3$  satisfies the same property, and therefore there exists  $\beta \in \text{Aut}(\mathbb{Q}; <)$  such that  $t_3 = \beta(t'_3)$ , and hence  $t_3 \in R$ .

For the opposite direction, we assume that  $R$  is preserved by  $pp$ , and have to show shuffle closure of  $R$ . Let  $t_1, t_2$  be tuples in  $R$ , and let  $l \in [k]$ . Choose  $\gamma \in \text{Aut}(\mathbb{Q}; <)$  such that  $\gamma$  maps  $t_1[l]$  to 0. Then  $t_3 = pp(\gamma(t_1), t_2)$  is a tuple that satisfies the conditions specified in the definition of shuffle-closure.  $\square$

Due to Proposition 21, we use for constraint languages  $\Gamma$  the phrases ‘ $\Gamma$  is shuffle-closed’ and ‘ $\Gamma$  is preserved by  $pp$ ’ interchangeably. The following lemma states an important property of shuffle-closed languages that will be used several times in the next subsections.

**Lemma 22.** *Let  $t_1, \dots, t_l$  be tuples from a  $k$ -ary relation  $R$  closed under  $pp$ , and let  $M_1, \dots, M_l \subset [k]$  be disjoint sets of indices such that  $\bigcup_{i=1}^l M_i = [k]$  and such that for all  $i, j \in [l]$  with  $i < j$  and for all  $i' \in M_i, j' \in M_j$  it holds that  $t_i[i'] < t_j[j']$ . Then there is a tuple  $t \in R$  such that*

- for all  $i, j \in [l]$  with  $i < j$  and for all  $i' \in M_i, j' \in M_j$  it holds that  $t[i'] < t[j']$ ;
- for all  $i \in [l]$  and all  $i', i'' \in M_i$  it holds that  $t[i'] \leq t[i'']$  iff  $t_i[i'] \leq t_i[i'']$ .

*Proof.* Let  $\beta_1, \dots, \beta_{l-1} \in \text{Aut}(\mathbb{Q}; <)$  be such that  $\beta_i$  maps  $\max\{t_i[i'] \mid i' \in M_i\}$  to 0. We set

$$t := pp(\beta_1(t_1), pp(\beta_2(t_2), \dots, pp(\beta_{l-1}(t_{l-1}), t_l) \dots)) .$$

The tuple  $t$  clearly belongs to  $R$ .

We prove by induction on  $l$  that  $t$  satisfies the other conditions of the lemma. Observe that  $\beta_1$  maps all the entries of  $t_1$  at  $M_1$  to non-positive values. Thus for  $l = 2$ , it is easy to check from the

properties of  $pp$  that for each  $i \in M_1$  and  $i' \in M_2$  we have  $t[i] < t[i']$  as required by the statement of the lemma. Also the second condition is immediate. For  $l > 2$  let  $t'$  be defined by

$$t' := pp(\beta_2(t_2), pp(\beta_3(t_3), \dots, pp(\beta_{l-1}(t_{l-1}), t_l) \dots)).$$

Then we have  $t = pp(\beta_1(t_1), t')$ . Now we apply the same argument as for  $l = 2$ . Because the order on  $[k] \setminus M_1$  is preserved by the application of  $pp$ , we know that the conditions are satisfied for the sets  $M_2, \dots, M_l$ . The argument also shows that the entries at  $M_1$  are smaller than the entries at  $[k] \setminus M_1$  and that their order is the same as in  $t_1$ .  $\square$

The following lemma is a simple criterion for showing that certain operations generate  $pp$ .

**Lemma 23.** *Let  $f$  be a binary operation preserving  $<$  such that for some automorphisms  $\alpha, \beta$  of  $(\mathbb{Q}; <)$  we have  $f(x, y) = \alpha(x)$  for all  $x \leq -1$ ,  $0 < y < 1$ , and  $f(x, y) = \beta(y)$  for all  $x > 1$ ,  $0 < y < 1$ . Then  $f$  generates  $pp$ .*

*Proof.* It suffices to show that every relation preserved by  $f$  is also preserved by  $pp$ . Let  $R$  be preserved by  $f$ , and let  $t_1, t_2$  be two tuples from  $R$ . Let  $\gamma \in \text{Aut}(\mathbb{Q}; <)$  be such that  $\gamma(x) = x + 1$  for all positive entries  $x$  of  $t_2$  and  $\gamma(x) = x - 1$  for all other entries  $x$  of  $t_2$ . Let  $\delta \in \text{Aut}(\mathbb{Q}; <)$  be such that all entries of  $\delta(t_2)$  are larger than 0 and smaller than 1. Then  $f(\gamma(t_1), \delta(t_2))$  is in the same orbit as  $pp(t_1, t_2)$ , which is what we wanted, to show.  $\square$

It is easy to verify that the relation  $S$ , defined in Subsection 2.6, is shuffle-closed. Proposition 15 shows that  $\text{CSP}(S)$  is NP-complete. Hence, the property of shuffle-closure is not strong enough to guarantee tractability.

## 4.2 Operations providing min-union closure

This section introduces and studies a stronger property than shuffle-closure, namely preservation under the binary operation  $min$  that maps two values  $x$  and  $y$  to the smaller of the two values; see Figure 2 for an illustration of the operation  $min$ . We also present a sufficient condition that implies that a temporal constraint language is preserved by  $min$ .

For constraint languages over a finite domain,  $min$ - and  $max$ -closed relations were studied in [31]. An equivalent clausal description of such constraints is known; however, the equivalence only holds for *finite* domains. The tractability of the CSP where the constraint language has such a clausal description has also been shown for infinite domains [18]. But the algorithm presented in [18] cannot be applied to all  $min$ -closed constraint languages over an infinite domain; it is already not clear how to adapt this approach to deal with the relation  $\{(x, y, z) \mid x > y \vee x > z\}$ , which is clearly  $min$ -closed. In Subsection 5.1 we describe an algorithm that efficiently solves the CSP for temporal constraint languages that are preserved by  $min$ .

**Definition 5.** *Let  $t$  be from  $\mathbb{Q}^k$ . The set of indices  $\{i \in [k] \mid t[i] \leq t[j] \text{ for all } j \in [k]\}$  is called the min-set of  $t$ , and denoted by  $M(t)$ .*

**Definition 6.** *A relation is called min-union closed if for all tuples  $t_1, t_2$  in  $R$  there exists a tuple  $t_3$  in  $R$  such that  $M(t_3) = M(t_1) \cup M(t_2)$ .*

We now want to link min-union closure of the relations in the constraint language to the existence of certain polymorphisms.

**Definition 7.** *Let  $f$  be a binary operation preserving  $<$ . We say that  $f$  provides min-union closure if  $f(0, 0) = f(0, x) = f(x, 0)$  for all integers  $x > 0$ .*

The operation  $min$  mentioned above is an example of an operation providing min-union closure. The following lemma connects Definition 6 and Definition 7.

**Lemma 24.** *Let  $R$  be a temporal relation preserved by an operation  $f$  providing min-union closure. Then  $R$  is min-union closed.*

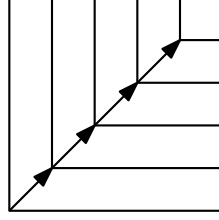


Figure 2: Illustration of the operation  $min$ .

*Proof.* Let  $t_1$  and  $t_2$  be tuples in  $R$ , and let  $a_1$  and  $a_2$  be the minimal values among the entries of  $t_1$  and  $t_2$ , respectively. Then there are  $\alpha_1, \alpha_2 \in \text{Aut}(\mathbb{Q}; <)$  such that  $\alpha_1(a_1) = \alpha_2(a_2) = 0$ , and such that  $\alpha_1$  and  $\alpha_2$  map all other entries of  $t_1$  and  $t_2$  to integers. Observe that all entries at  $M(t_1) \cup M(t_2)$  in the tuple  $t_3 = f(\alpha_1(t_1), \alpha_2(t_2))$  have the same value. Because  $f$  preserves  $<$ , this value is strictly smaller than the values at all other entries in  $t_3$ . Hence,  $M(t_3) = M(t_1) \cup M(t_2)$ .  $\square$

The following proposition implies that  $\{f, pp\}$  generates  $min$  for every operation  $f$  that provides min-union closure.

**Proposition 25.** *A temporal relation  $R$  is preserved by  $pp$  and an operation providing min-union closure if and only if  $R$  is preserved by  $min$ .*

*Proof.* Clearly,  $min$  provides min-union closure. Also observe that the operation  $min$  satisfies the conditions of Lemma 23, and hence  $min$  generates  $pp$ .

For the opposite direction, suppose that  $R$  is  $k$ -ary and preserved by  $pp$  and an operation  $f$  providing min-union closure. We show that for any two tuples  $t_1, t_2 \in R$  the tuple  $t_3 = min(t_1, t_2)$  is in  $R$  as well. Let  $l$  be the number of distinct values in  $t_3$  and  $v_1 < v_2 < \dots < v_l$  be these values. We define  $M_i$ ,  $i \in [l]$ , to be the set of indices of  $t_3$  with the  $i$ -th lowest value, i.e.,  $M_i = \{j \in [k] \mid t_3[j] = v_i\}$ .

Now let  $\alpha_1, \dots, \alpha_l \in \text{Aut}(\mathbb{Q}; <)$  be such that  $\alpha_i(v_i) = 0$  and such that the entries of  $\alpha_i(t_1)$  and  $\alpha_i(t_2)$  are integers. Using these automorphisms we define the tuples  $s_1, \dots, s_l$  by  $s_i = f(\alpha_i(t_1), \alpha_i(t_2))$ . Clearly, these tuples belong to  $R$ . It also holds that  $s_i$  is constant at  $M_i$  because for each  $j \in M_i$  at least one of the entries  $t_1[j], t_2[j]$  is equal to  $v_i$  (the other one can be only greater) which is subsequently mapped to 0 by  $\alpha_i$  and  $f$  maps all such pairs to the same value. Furthermore, for each  $j' \in M_{i'}$  for  $i < i' \leq l$  we have that  $s_i[j']$  is greater than the value of  $s_i$  at  $M_i$ , because  $min(t_1[j'], t_2[j']) = v_{i'}$  is greater than  $v_i$  and  $f$  preserves  $<$ .

Now we can apply Lemma 22 to the obtained tuples  $s_1, \dots, s_l$  and the corresponding sets  $M_1, \dots, M_l$ . The lemma gives us some tuple  $t'_3$  from  $R$  which is constant at each set  $M_i$ ,  $i \leq [l]$ , and such that for each  $i < j \leq l$  the value of  $t'_3$  at  $M_i$  is lower than the value of  $t'_3$  at  $M_j$ . Thus  $t'_3$  has the same order of entries as  $t_3$  which shows that  $t_3$  is in  $R$  as well.  $\square$

### 4.3 Operations providing min-intersection closure

In this section, we study a different restriction of shuffle-closed constraint languages.

**Definition 8.** *A relation  $R$  is called min-intersection closed if for all tuples  $t_1, t_2$  in  $R$ , if  $M(t_1) \cap M(t_2) \neq \emptyset$ , then there exists a tuple  $t_3$  in  $R$  such that  $M(t_3) = M(t_1) \cap M(t_2)$ .*

**Definition 9.** *Let  $f$  be a binary operation preserving  $<$ . We say that  $f$  provides min-intersection closure if  $f(0, 0) < f(0, x)$  and  $f(0, 0) < f(x, 0)$  for all integers  $x > 0$ .*

**Lemma 26.** *Let  $R$  be a temporal relation that is preserved by an operation  $f$  that provides min-intersection closure. Then  $R$  is min-intersection closed.*

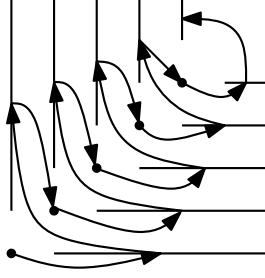


Figure 3: Illustration of the operation  $mi$ .

*Proof.* Let  $t_1$  and  $t_2$  be two tuples in  $R$  such that  $M(t_1) \cap M(t_2)$  is non-empty, that is, it contains an index  $i$ . Choose  $\alpha_1, \alpha_2 \in \text{Aut}(\mathbb{Q}; <)$  such that  $\alpha_1(t_1[i]) = \alpha_2(t_2[i]) = 0$ , and such that  $\alpha_1$  and  $\alpha_2$  map all other entries of  $t_1$  and  $t_2$  to integers. Consider the tuple  $t_3 = f(\alpha_1(t_1), \alpha_2(t_2))$ . Because at the entries from  $M(t_1)$  (from  $M(t_2)$ ) the tuple  $\alpha_1(t_1)$  ( $\alpha_2(t_2)$ ) equals 0, and because  $f(0, 0) < f(0, x)$  and  $f(0, 0) < f(x, 0)$  for all positive integers  $x$ , it follows that in  $t_3$  all entries at  $M(t_1) \cap M(t_2)$  have a strictly smaller value than all values at the symmetric difference  $M(t_1) \Delta M(t_2)$ . Because  $f$  preserves  $<$ , it also follows that all entries at  $M(t_1) \cap M(t_2)$  have a strictly smaller value than the entries not at  $M(t_1) \cup M(t_2)$ . We conclude that  $M(t_3) = M(t_1) \cap M(t_2)$ .  $\square$

An example of an operation that provides min-intersection closure is the operation  $mi$ , defined by

$$mi(x, y) := \begin{cases} \alpha(x) & \text{if } x < y \\ \beta(x) & \text{if } x = y \\ \gamma(y) & \text{if } x > y \end{cases}$$

where  $\alpha, \beta, \gamma$  are unary operations that preserve  $<$  such that

$$\beta(x) < \gamma(x) < \alpha(x) < \beta(x + \varepsilon)$$

for all  $x \in \mathbb{Q}$  and all  $0 < \varepsilon \in \mathbb{Q}$  (see Figure 3). Operations  $\alpha, \beta, \gamma$  with these properties can be constructed as follows. Let  $a_1, a_2, \dots$  be an enumeration of  $\mathbb{Q}$ . Inductively assume that we have already defined  $\alpha, \beta, \gamma$  on  $\{a_1, \dots, a_n\}$  such that  $\beta(a_i) < \gamma(a_i) < \alpha(a_i) < \beta(a_j)$  whenever  $a_i < a_j$ , for  $i, j \in [n]$ . Clearly, this is possible for  $n = 1$ . If  $a_{n+1} > a_i$  for all  $i \in [n]$ , let  $a_j$  be the maximum of  $\{a_1, \dots, a_n\}$ , and define  $\alpha(a_j) < \beta(a_{n+1}) < \gamma(a_{n+1}) < \alpha(a_{n+1})$ . In the case that  $a_{n+1} < a_i$  for all  $i \in [n]$  we proceed analogously. Otherwise, let  $i, j \in [n]$  such that  $a_i$  is the largest possible and  $a_j$  is smallest possible such that  $a_i < a_{n+1} < a_j$ . In this case, define  $\alpha(a_i) < \beta(a_{n+1}) < \gamma(a_{n+1}) < \alpha(a_{n+1}) < \beta(a_j)$ . In this way we define unary operations  $\alpha, \beta, \gamma$  on all of  $\mathbb{Q}$  with the desired properties.

In fact, the operation  $mi$  will be of special importance, because the following proposition shows that  $pp$  together with any operation providing min-intersection closure generates the operation  $mi$ .

**Proposition 27.** *A temporal relation  $R$  is preserved by  $pp$  and an operation  $f$  providing min-intersection closure if and only if  $R$  is preserved by  $mi$ .*

*Proof.* It is clear that  $mi$  provides min-intersection closure, and Lemma 23 shows that  $mi$  generates  $pp$ .

For the opposite direction, suppose  $R$  is  $k$ -ary and preserved by  $pp$  and an operation  $f$  providing min-intersection closure. We show that for any two tuples  $t_1, t_2 \in R$  the tuple  $t_3 = mi(t_1, t_2)$  is in  $R$  as well. Let  $\alpha, \beta, \gamma$  be the mappings from the definition of the operation  $mi$ . Let  $v_1 < \dots < v_l$  be the minimal-length sequence of rational numbers such that for each  $i' \in [k]$  it holds that  $t_3[i'] \in \bigcup_{j \in [l]} \{\alpha(v_j), \beta(v_j), \gamma(v_j)\}$ . Let  $M_i$  be

$$\{i' \in [k] \mid t_3[i'] \in \{\alpha(v_i), \beta(v_i), \gamma(v_i)\}\}.$$

Observe that for each  $i' \in M_i$  at least one of  $t_1[i']$  and  $t_2[i']$  is equal to  $v_i$  and the other value is greater or equal to  $v_i$ . Let  $M_i^\alpha$  be the set of those  $i' \in M_i$  where  $v_i = t_1[i'] < t_2[i']$ ,  $M_i^\beta$  the set of those  $i' \in M_i$  where  $v_i = t_1[i'] = t_2[i']$ , and  $M_i^\gamma$  the set of those  $i' \in M_i$  where  $v_i = t_2[i'] < t_1[i']$ .

Let  $\delta_1, \dots, \delta_l \in \text{Aut}(\mathbb{Q}; <)$  be such that  $\delta_i$  maps  $v_i$  to 0 and such that the entries of  $\delta_i(t_1)$  and  $\delta_i(t_2)$  are integers. Let  $\eta$  be a permutation from  $\text{Aut}(\mathbb{Q}; <)$  that maps  $f(0, 0)$  to 0. For each  $i \in [l]$  we define

$$s_i := pp(\eta(f(\delta_i(t_1), \delta_i(t_2))), pp(\delta_i(t_2), t_1)). \quad (1)$$

We verify that for all  $i \in [l]$  the tuple  $s_i$  is constant on each of the sets  $M_i^\alpha, M_i^\beta, M_i^\gamma$ , the value at  $M_i^\beta$  is lower than the value at  $M_i^\gamma$  which is lower than the value at  $M_i^\alpha$ . Furthermore, for each  $j \in [l], j > i$ , and each  $i' \in M_i, j' \in M_j$ , it holds that  $s_i[i'] < s_i[j']$ . Having this, we can apply Lemma 22 and obtain a tuple from  $R$  with the same ordering of entries as in  $t_3$ , which proves the lemma.

Because  $\delta_i$  maps  $v_i$  to 0, the properties of  $pp$  imply that the tuple  $t'_i = pp(\delta_i(t_2), t_1)$  is constant at  $M_i^\beta \cup M_i^\gamma$  and at  $M_i^\alpha$ , and the value at the first set is smaller than the value at the second set. Because the values of  $t_2$  at  $M_i^\alpha \cup \bigcup_{j=i+1}^l M_j$  are greater than  $v_i$  and the values of  $t_1$  at  $\bigcup_{j=i+1}^l M_j$  are also greater than  $v_i$  (recall that for each  $j \in [l], j' \in M_j$  it holds that  $\min(t_1[j'], t_2[j']) = v_j$ ) we conclude that the values of  $t'_i$  at  $\bigcup_{j=i+1}^l M_j$  are greater than those at  $M_i$ .

The application of  $f$  in (1) yields a tuple which is constant on  $M_i^\beta$  and its value there (which is consequently mapped to 0 by  $\eta$ ) is smaller than the values at  $M_i^\alpha \cup M_i^\gamma \cup \bigcup_{j=i+1}^l M_j$ . Thus it is easy to verify from the properties of  $pp$  that the outer application of  $pp$  in (1) yields a tuple with the desired properties.  $\square$

**Example.** An interesting example of a relation that is preserved by  $mi$  but not by  $min$  is the 4-ary relation  $I$  defined as follows.

$$\begin{aligned} I(a, b, c, d) \equiv & (a = b \wedge b < c \wedge c = d) \\ & \vee (a = b \wedge b > c \wedge c = d) \\ & \vee (a = b \wedge b < c \wedge c < d) \\ & \vee (a > b \wedge b > c \wedge c = d) \end{aligned}$$

To see that  $I$  is preserved by  $mi$ , let  $t_1$  and  $t_2$  be two tuples from  $I$ . We have to show that  $t_3 := mi(t_1, t_2) \in I$ . First note that  $I(a, b, c, d)$  is equivalent to

$$(a \geq b) \wedge (b \neq c) \wedge (c \leq d) \wedge (a = b \vee b > c) \wedge (b < c \wedge c = d),$$

and that  $mi$  preserves  $\leq$  and  $\neq$ .

We distinguish the following cases.

1.  $t_1[2] < t_1[3]$  and  $t_2[2] < t_2[3]$ . Then  $t_1[1] = t_1[2]$  and  $t_2[1] = t_2[2]$ , and hence  $t_3[1] = t_3[2]$ . Since  $mi$  preserves  $<$ , we have  $t_3[2] < t_3[3]$ . Since  $mi$  preserves  $\leq$ , we have that  $t_3[3] \leq t_3[4]$ , and hence  $t_3[1] = t_3[2] < t_3[3] < t_3[4]$  or  $t_3[1] = t_3[2] < t_3[3] = t_3[4]$ , which proves the claim.
2.  $t_1[2] < t_1[3]$  and  $t_2[2] > t_2[3]$ . Then  $t_1[1] = t_1[2]$  and  $t_2[3] = t_2[4]$ . We verify that  $t_3$  satisfies the equivalent characterization of  $I$  given above; since  $mi$  preserves  $\leq$  and  $\neq$ , this amounts to proving that  $t_3$  satisfies the two clauses  $(a = b \vee b > c) \wedge (b < c \wedge c = d)$ .

The first sub-case we consider is  $t_3[2] < t_3[3]$ . Then by the assumptions on  $t_1$  and  $t_2$  and by definition of  $mi$  we have that  $t_1[2] < t_2[2]$ . Therefore,  $t_1[1] = t_1[2] < t_2[2] \leq t_2[1]$  and thus  $t_3[1] = t_3[2]$  again by the properties of  $mi$ ; we see that both clauses are satisfied. The second sub-case is that  $t_3[2] > t_3[3]$ . Then by the assumptions on  $t_1$  and  $t_2$  and by definition of  $mi$  we have that  $t_1[4] \geq t_1[3] > t_2[3] = t_2[4]$ . Thus  $t_3[3] = t_3[4]$  and again both clauses are satisfied.

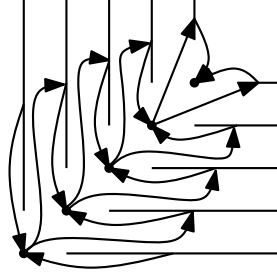


Figure 4: Illustration of the operation  $mx$ .

3.  $t_1[2] > t_1[3]$  and  $t_2[2] > t_2[3]$ . This is analogous to the first case.
4.  $t_1[2] > t_1[3]$  and  $t_2[2] < t_2[3]$ . This is analogous to the second case.

The relation  $I$  is not preserved by  $min$  since  $(0, 0, 1, 2) \in I$  and  $(2, 1, 0, 0) \in I$  but  $min((0, 0, 1, 2), (2, 1, 0, 0)) = (0, 0, 0, 0) \notin I$ .

**Example.** The following ternary temporal relation  $U$  is preserved by  $min$  (we omit the easy proof), but not preserved by  $mi$ .

$$\begin{aligned}
 U(x, y, z) \equiv & (x = y \wedge y < z) \\
 & \vee (x = z \wedge z < y) \\
 & \vee (x = y \wedge y = z)
 \end{aligned}$$

To see that  $U$  is not preserved by  $mi$ , note that  $mi((0, 0, 1), (0, 1, 0))$  has three distinct values and hence is not in  $U$ , but  $(0, 0, 1), (0, 1, 0) \in U$ . An algorithm that solves constraint languages preserved by  $mi$  can be found in Subsection 5.2.

#### 4.4 Operations providing min-xor closure

We now introduce the last of the mentioned closure conditions.

**Definition 10.** A relation is called *min-xor closed* if for all tuples  $t_1, t_2$  in  $R$  where the symmetric difference  $M(t_1) \Delta M(t_2)$  is nonempty there exists a tuple  $t_3$  in  $R$  such that  $M(t_3) = M(t_1) \Delta M(t_2)$ .

**Definition 11.** Let  $f$  be a binary operation preserving  $<$ . We say that  $f$  provides *min-xor closure* if  $f(0, 0) > f(0, x) = f(y, 0)$  for all integers  $x, y > 0$ .

For an example of a binary operation that provides min-xor closure, consider the following binary operation, which we denote by  $mx$ .

$$mx(x, y) := \begin{cases} \alpha(\min(x, y)) & \text{if } x \neq y \\ \beta(x) & \text{if } x = y \end{cases}$$

where  $\alpha$  and  $\beta$  are unary operations that preserve  $<$  such that  $\alpha(x) < \beta(x) < \alpha(x + \varepsilon)$  for all  $x \in \mathbb{Q}$  and all  $0 < \varepsilon \in \mathbb{Q}$  (see Figure 4). Similarly as for the definition of  $mi$ , such operations  $\alpha, \beta$  can be easily constructed. It is easy to see that the operation  $mx$  neither preserves the relation  $I$  nor the relation  $U$  introduced in Subsection 4.3.

**Lemma 28.** Let  $R$  be a temporal relation that is preserved by an operation  $f$  providing min-xor closure. Then  $R$  is min-xor closed.

*Proof.* Let  $t_1$  and  $t_2$  be tuples in  $R$ , and suppose that the symmetric difference  $M(t_1) \triangle M(t_2)$  of  $M(t_1)$  and  $M(t_2)$  is non-empty. Let  $a_1$  and  $a_2$  be the minimal values of the entries of  $t_1$  and of  $t_2$ , respectively. Then there are  $\alpha_1, \alpha_2 \in \text{Aut}(\mathbb{Q}; <)$  such that  $\alpha_1(a_1) = 0$  and  $\alpha_2(a_2) = 0$  and such that  $\alpha_1$  and  $\alpha_2$  map all other entries of  $t_1$  and  $t_2$  to integers. Consider the tuple  $t_3 = f(\alpha_1(t_1), \alpha_2(t_2))$ . Because  $\alpha_1(t_1)$  is 0 for all entries at  $M(t_1)$ ,  $\alpha_2(t_2)$  is 0 for all entries at  $M(t_2)$ , and  $f(0, 0) > f(0, x) = f(y, 0)$  for all  $x, y > 0$ , it follows that in  $t_3$  all entries at  $M(t_1) \cap M(t_2)$  have a strictly larger value than all entries at  $M(t_1) \triangle M(t_2)$ , which all have the same value. Because  $f$  preserves  $<$ , all entries of  $t_3$  at  $M(t_1) \cap M(t_2)$  have a smaller value than all entries not at  $M(t_1) \cup M(t_2)$ . We conclude that the tuple  $t_3 \in R$  satisfies  $M(t_3) = M(t_1) \triangle M(t_2)$ .  $\square$

The following lemma implies that  $\{f, pp\}$  generates  $mx$  for any operation  $f$  that provides min-xor closure.

**Proposition 29.** *A temporal relation  $R$  is preserved by  $pp$  and an operation  $f$  providing min-xor closure if and only if  $R$  is preserved by  $mx$ .*

*Proof.* Clearly,  $mx$  provides min-xor closure. Lemma 23 shows that  $mx$  generates  $pp$ .

For the opposite direction, suppose that  $R$  is  $k$ -ary and preserved by  $pp$  and an operation  $f$  providing min-xor closure. We show that for any two tuples  $t_1, t_2 \in R$  the tuple  $t_3 = mx(t_1, t_2)$  is in  $R$  as well. Let  $\alpha, \beta$  be the mappings as in the definition of the operation  $mx$ . Let  $v_1 < \dots < v_l$  be minimal set of rational numbers such that  $t_3[i] \in \bigcup_{j \in [l]} \{\alpha(v_j), \beta(v_j)\}$  for all  $i \in [k]$ , and let  $M_i$  be the set of indices  $\{i' \in [k] \mid t_3[i'] \in \{\alpha(v_i), \beta(v_i)\}\}$ . Observe that for each  $i' \in M_i$  at least one of  $t_1[i']$  and  $t_2[i']$  is equal to  $v_i$  and the other value is greater or equal to  $v_i$ . Let  $M_i^\alpha$  be the set of those  $i' \in M_i$  where  $t_1[i'] \neq t_2[i']$  and  $M_i^\beta$  the set of those  $i' \in M_i$  where  $v_i = t_1[i'] = t_2[i']$ .

Let  $\delta_1, \dots, \delta_l \in \text{Aut}(\mathbb{Q}; <)$  be such that  $\delta_i$  maps  $v_i$  to 0 and such that the entries of  $\delta_i(t_1)$  and  $\delta_i(t_2)$  are integers. For each  $i \in [l]$  we define  $s_i := f(\delta_i(t_1), \delta_i(t_2))$ . It is easy to see from the choice of  $\delta_i$  and properties of  $f$  that for each  $i \in [l]$  the tuple  $s_i$  is constant at  $M_i^\alpha, M_i^\beta$ , and that the value at  $M_i^\alpha$  is lower than the value at  $M_i^\beta$ . Furthermore, because  $f$  preserves  $<$ , because the values of  $t_1$  at  $\bigcup_{j=i+1}^l M_j$  are greater than  $v_i$ , and because the values of  $t_2$  at  $\bigcup_{j=i+1}^l M_j$  are greater than  $v_i$ , we see that for each  $j \in [l], j > i$  and each  $i' \in M_i, j' \in M_j$ , it holds that  $s_i[i'] < s_i[j']$ . Having this, we can apply Lemma 22 and obtain a tuple from  $R$  with the same ordering of entries as in  $t_3$ , which proves the lemma.  $\square$

An interesting example of a temporal relation that is preserved by  $mx$  is the ternary relation  $X$  defined as follows.

$$\begin{aligned} X(x, y, z) &\equiv (x = y \wedge y < z) \\ &\quad \vee (x = z \wedge z < y) \\ &\quad \vee (y = z \wedge y < x) \end{aligned}$$

The relation is not preserved by  $min$  and by  $mi$ : the tuples  $t_1 = (0, 0, 1)$ ,  $t_2 = (0, 1, 0)$  are in  $X$ , but  $min(t_1, t_2) = (0, 0, 0) \notin X$ , and  $mi(t_1, t_2)$  has three distinct entries and hence is not in  $X$  as well.

An algorithm that solves constraint languages preserved by  $mx$  can be found in Subsection 5.3.

## 4.5 Operations generating $min$ , $mi$ , $mx$

As we have seen in Proposition 15, if the relation  $S$  has a primitive positive definition in  $\Gamma$ , then  $\text{CSP}(\Gamma)$  is NP-hard. We show that if a temporal constraint language is shuffle-closed and does not admit a primitive positive definition of  $S$ , then it is preserved by  $min$ ,  $mi$ , or  $mx$ .

If the relation  $S$  does not have a primitive positive definition in  $\Gamma$ , then Theorem 9 implies that there is a polymorphism  $f$  of  $\Gamma$  that does not preserve  $S$ . By Theorem 20, it suffices to consider languages  $\Gamma$  such that  $<$  has a primitive positive definition in  $\Gamma$ . We start with a sequence of auxiliary lemmas.

**Lemma 30.** *Let  $f$  be a binary operation preserving  $<$ , and suppose that there is an infinite sequence  $x_1 < x_2 < \dots$  of elements of  $\mathbb{Q}$  and  $y_1 \in \mathbb{Q}$  such that  $f(x_1, y_1) \geq f(x_2, y_1) < f(x_i, y_1)$  for all  $i > 2$ . Then  $f$  generates an operation providing min-intersection closure.*

*Proof.* Because  $f$  preserves  $<$ , we have that for any infinite sequence  $y_1 < y_2 < \dots$  it holds that  $f(x_2, y_i) > f(x_1, y_1)$ . Hence, the binary operation defined by  $f(\alpha(x), \beta(y))$  provides min-intersection closure, where  $\alpha \in \text{Aut}(\mathbb{Q}; <)$  maps  $0, 1, \dots$  to  $x_2, x_3, \dots$  and  $\beta \in \text{Aut}(\mathbb{Q}; <)$  maps  $0, 1, 2, \dots$  to  $y, y_1, y_2, \dots$   $\square$

**Lemma 31.** *Suppose  $f$  preserves  $<$  and generates a sequence of operations  $f_1, f_2, \dots$  such that for each  $f_k$  it holds that  $f_k(0, 0) < f_k(x, 0)$  and  $f_k(0, 0) < f_k(0, x)$  for all integers  $x \in [k]$ . Then  $f$  generates an operation  $g$  providing min-intersection closure.*

*Proof.* Fix an enumeration  $x_1, x_2, \dots$  of  $\mathbb{Q}$ . For each  $k$ , we define an equivalence relation  $\sim$  on the set  $\mathcal{S}_k$  of all restrictions of operations from  $\{f_1, f_2, \dots\}$  to  $\{x_1, \dots, x_k\}^2$ . Define  $f \sim f'$  for two operations  $f, f' \in \mathcal{S}_k$  iff there exists  $\alpha \in \text{Aut}(\mathbb{Q}; <)$  such that  $f(x_i, x_j) = \alpha(f'(x_i, x_j))$  for all  $i, j \in [k]$ . Clearly,  $\sim$  is an equivalence relation, and for every  $k$  and every function  $f \in \mathcal{S}_k$  there are only finitely many weak linear orders of the set  $\{f(x_i, x_j) \mid i, j \in [k]\}$ . Hence,  $\sim$  has only finitely many equivalence classes on  $\mathcal{S}_k$ .

We now define an infinite directed acyclic graph whose vertices are the equivalence classes of  $\sim$  on all sets  $\mathcal{S}_k$  and where  $(f, f')$  is an arc if  $f \in \mathcal{S}_k$ ,  $f' \in \mathcal{S}_{k+1}$ , and  $f'$  restricted to  $\{x_1, \dots, x_k\}^2$  is equivalent to  $f$  under  $\sim$ . We have already observed that this graph must have finite outdegree, and since there are arbitrarily long paths starting at the equivalence class of the mapping  $g_0$  with the empty domain, König's tree lemma implies that the tree contains an infinite path of equivalence classes starting at the equivalence class of  $g_0$ .

Now, we use this infinite path to define  $g(x, y)$  inductively as follows. The restriction of  $g$  to  $\{x_1, \dots, x_k\}^2$  will be an element from the  $k$ -th node of the infinite path. Initially, this is trivially true if  $g$  is restricted to the empty set. Suppose  $g$  is already defined on  $\{x_1, \dots, x_k\}^2$ , for  $k \geq 0$ . By construction of the infinite path, we find representatives  $g_k$  of the  $k$ -th and  $g_{k+1}$  of the  $k+1$ -st element on the path such that  $g_k$  is a restriction of  $g_{k+1}$ . The inductive assumption gives us  $\alpha \in \text{Aut}(\mathbb{Q}; <)$  such that  $\alpha(g_k(x, y)) = g(x, y)$  for all  $x, y \in \{x_1, \dots, x_k\}$ . We set  $g(x_{k+1}, y)$  to be  $\alpha(g_{k+1}(x_{k+1}, y))$  and  $g(y, x_{k+1})$  to be  $\alpha(g_{k+1}(y, x_{k+1}))$  for all  $y \in \{x_1, \dots, x_{k+1}\}$ . The restriction of  $g$  to  $\{x_1, \dots, x_{k+1}\}^2$  will therefore be a member of the  $k+1$ -st element of the infinite path. The operation  $g$  defined in this way is indeed generated by  $\{f_1, f_2, \dots\}$ . By assumptions on  $\{f_1, f_2, \dots\}$  it also follows that  $g$  preserves  $<$  and  $g(0, 0) < g(0, x)$  and  $g(0, 0) < g(x, 0)$  for all integers  $x > 0$ , and so  $g$  provides min-intersection closure.  $\square$

**Lemma 32.** *Let  $f$  be a binary operation preserving  $<$  such that there is an infinite sequence  $x_1 < x_2 < \dots$  and  $y_1 \in \mathbb{Q}$  satisfying  $f(x_i, y) > f(x_j, y_1)$  for all  $1 \leq i < j$ . Then  $\{f, pp\}$  generates an operation providing min-intersection closure.*

*Proof.* By Lemma 31, it suffices to show that there is a sequence of operations  $f_1, f_2, \dots$ , generated by  $\{f, pp\}$  such that  $f_k(0, 0) < f_k(x, 0)$  and  $f_k(0, 0) < f_k(0, x)$  for all  $k \geq 1$  and all  $x \in [k]$ .

So let  $k \geq 0$  be a fixed integer, and  $y_1 < y_2 < \dots$  be an arbitrary infinite sequence. Let  $\alpha_k$  be from  $\text{Aut}(\mathbb{Q}; <)$  mapping  $\{f(x_i, y_i) \mid 1 \leq i \leq k\} \cup \{f(x_i, y_1) \mid 1 \leq i \leq k\}$  into  $\{x_2, \dots, x_{2k}\}$  and  $\beta_1, \beta_2 \in \text{Aut}(\mathbb{Q}; <)$  such that  $\beta_1$  maps  $0, 1, 2, \dots$  to  $x_1, x_2, x_3, \dots$  and  $\beta_2$  maps  $0, 1, 2, \dots$  to  $y_1, y_2, y_3, \dots$ . We define

$$f_k(x, y) := f(\alpha_k(f(\beta_1(x), \beta_2(y))), \beta_2(y)),$$

and show that  $f_k$  has the required properties. It follows from the assumptions on  $f$  that for all positive integers  $x$  we have  $f(\beta_1(0), \beta_2(0)) = f(x_1, y) > f(\beta_1(x), y_1) = f(\beta_1(x), \beta_2(0))$ , and due to the properties of  $\alpha_k$  it holds that  $f_k(0, 0) < f_k(x, 0)$  for all integers  $x \in [k]$ .

We also have for every  $x \in [k]$  that  $\beta_2(x) > y_1$  and  $\alpha_k(f(\beta_1(0), \beta_2(x))) > x_1$ . Because  $f$  preserves  $<$ , this shows that  $f_k(0, x) = f(\alpha_k(f(\beta_1(0), \beta_2(x))), \beta_2(x)) > f(x_1, y_1)$ . Moreover,  $f_k(0, 0) = f(\alpha_k(f(x_1, y_1)), y_1) < f(x_1, y_1)$  by the assumptions on  $f$ . Hence,  $f_k(0, x) > f(x_1, y_1) > f_k(0, 0)$  for all  $x \in [k]$ .  $\square$

The following lemma contains a simple application of (a special case of) Ramsey's theorem. More substantial applications of Ramsey theory can be found in Section 6.3.

**Theorem 33** (Infinite Version of Ramsey's theorem; see e.g. Theorem 5.6.1 in [28]). *Let  $D$  be a countably infinite set, and let  $m, r$  be finite integers. When  $\chi$  is a mapping from the  $m$ -element subsets of  $D$  into  $[r]$ , then there exists an infinite subset  $P$  of  $D$  such that  $\chi$  is constant on all  $m$ -element subsets of  $P$ .*

**Lemma 34.** *Let  $f$  be a binary operation preserving  $<$  such that there is an infinite sequence  $x_1 < x_2 < \dots$  and  $y_1 \in \mathbb{Q}$  satisfying  $f(x_1, y_1) > f(x_i, y_1) = f(x_j, y_1)$  for all  $1 < i < j$ . Then  $\{f, pp\}$  generates an operation providing min-intersection or min-xor closure.*

*Proof.* By the infinite pigeon-hole principle there must be an infinite sequence  $y_2 < y_3 < \dots$  of elements of  $\mathbb{Q}$  larger than  $y_1$  such that

1.  $f(x_2, y_1) = f(x_1, y_i)$  for all  $i \geq 2$ , or
2.  $f(x_2, y_1) > f(x_1, y_i)$  for all  $i \geq 2$ , or
3.  $f(x_2, y_1) < f(x_1, y_i)$  for all  $i \geq 2$ .

In case 1,  $f$  generates an operation providing min-xor closure and we are done. In case 2, we apply Ramsey's theorem (Theorem 33) in the special case of  $m = 2$ ,  $r = 3$  as follows. Let  $D$  be  $\{y_1, y_2, \dots\}$ . For  $i < j$ , define  $\chi(\{y_i, y_j\}) = 1$  if  $f(x_1, y_i) = f(x_1, y_j)$ ,  $\chi(\{y_i, y_j\}) = 2$  if  $f(x_1, y_i) > f(x_1, y_j)$ , and  $\chi(\{y_i, y_j\}) = 3$  if  $f(x_1, y_i) < f(x_1, y_j)$ . Then Theorem 33 applied to  $\chi$  shows that there exists an infinite subsequence  $z_1 < z_2 < \dots$  of  $y_1 < y_2 < \dots$  such that

- 2a.  $f(x_1, z_i) = f(x_1, z_j)$  for all  $1 \leq i < j$ , or
- 2b.  $f(x_1, z_i) > f(x_1, z_j)$  for all  $1 \leq i < j$ , or
- 2c.  $f(x_1, z_i) < f(x_1, z_j)$  for all  $1 \leq i < j$ .

In case 2a, we swap arguments of  $f$  and proceed as in case 3. In case 2b, we swap arguments of  $f$ , apply Lemma 32, and conclude that  $f$  generates an operation providing min-intersection closure. In case 2c, note that  $f(x_1, y_1) > f(x_2, y_1) > f(x_1, y_i)$  for all  $i \geq 2$ , and thus we can apply Lemma 30 to conclude that  $f$  generates an operation providing min-intersection closure.

In case 3, we show that similarly as in Lemma 32 there is a sequence of operations  $f_1, f_2, \dots$  generated by  $\{f, pp\}$  such that for each  $f_k$  it holds that  $f_k(0, 0) < f_k(x, 0)$  and  $f_k(0, 0) < f_k(0, x)$  for all integers  $x \in [k]$ , and conclude by application of Lemma 31. See Figure 5 for an illustration.

Let  $\alpha_k$  be from  $\text{Aut}(\mathbb{Q}; <)$  such that it maps  $f(x_2, y_1)$  to  $x_1$  and  $\{f(x_1, y_i) \mid 1 \leq i \leq k\}$  to  $\{x_2, \dots, x_{k+1}\}$ . Furthermore let  $\beta_1, \beta_2 \in \text{Aut}(\mathbb{Q}; <)$  be such that  $\beta_1$  maps  $0, 1, 2, \dots$  to  $x_1, x_2, x_3, \dots$  and  $\beta_2$  maps  $0, 1, 2, \dots$  to  $y_1, y_2, y_3, \dots$ . We define

$$f_k(x, y) := f(\alpha_k(f(\beta_1(x), \beta_2(y))), \beta_2(y)).$$

Then  $f_k(0, 0) = f(\alpha_k(f(x_1, y_1)), y_1) = f(x_2, y_1)$  and  $f_k(x, 0) = f(\alpha_k(f(\beta_1(x), y_1)), y_1) = f(x_1, y_1)$  for all integers  $x > 0$ . Hence  $f_k(0, 0) < f_k(x, 0)$  for all integers  $x > 0$ . Finally, as  $\beta_2(x) > y_1$  and  $\alpha_k(f(x_1, \beta_2(x))) > x_1$  for all integers  $x > 0$ , we have that  $f_k(0, x) > f(x_1, y_1) > f(x_2, y_1) = f_k(0, 0)$ .  $\square$

The previous two lemmas are combined in the following result.

**Lemma 35.** *Let  $f$  be a binary operation that preserves  $<$  and violates the relation  $\leq$ . Then  $\{f, pp\}$  generates an operation providing min-intersection or min-xor closure.*

*Proof.* As  $f$  violates  $\leq$ , we can without loss of generality assume that there is  $y \in \mathbb{Q}$  and  $x_1, x_2 \in \mathbb{Q}$ ,  $x_1 < x_2$ , such that  $f(x_1, y) > f(x_2, y)$ .

We claim that there are only three possibilities:

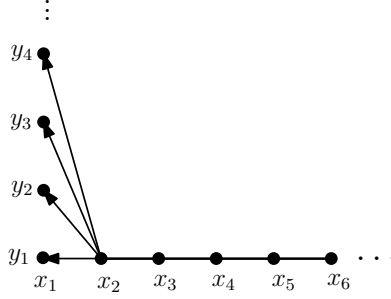


Figure 5: Illustration for Case 3 of Lemma 34.

- a) There is an infinite sequence  $x_3 < x_4 < \dots$  such that  $x_2 < x_3$  and  $f(x_i, y) > f(x_2, y)$  for all  $i > 2$ .
- b) There is an infinite sequence  $x_3 < x_4 < \dots$  such that  $x_2 < x_3$  and  $f(x_i, y) > f(x_j, y)$  for all  $2 \leq i < j$ .
- c) There is an infinite sequence  $x_3 < x_4 < \dots$  such that  $x_2 < x_3$  and  $f(x_i, y) = f(x_2, y)$  for all  $i > 2$ .

To show this claim, observe that by the infinite pigeon-hole principle there is an infinite sequence  $x_3 < x_4 < \dots$  with  $x_2 < x_3$  such that  $f(x_i, y) > f(x_2, y)$  for all  $i > 2$ ,  $f(x'_i, y) = f(x_2, y)$  for all  $i > 2$ , or  $f(x_i, y) < f(x_2, y)$  for all  $i > 2$ . In the first and the second case the claim holds. In the third case, we repeat the argument with  $x_2 < x_3$  instead of  $x_1 < x_2$ . Again, we distinguish three cases, and as before in two of them we are immediately done. In the third case, we repeat again. If we repeat this for infinitely many times we obtain a sequence  $x_3 = x'_3 < x'_4 < \dots$  such that  $x_2 < x_3$  and  $f(x'_i, y) > f(x'_j, y)$  for all  $2 \leq i < j$ .

In a) the conditions of Lemma 30 are satisfied and we conclude that  $\{f, pp\}$  generates an operation providing min-intersection closure. In b) Lemma 32 shows that  $\{f, pp\}$  generates an operation providing min-intersection closure. In c) we apply Lemma 34 and conclude that  $\{f, pp\}$  generates an operation providing min-intersection or min-xor closure.  $\square$

The following is the main result of this subsection. Recall that the relation  $S$  was defined in Definition 3 to be

$$\{(x, y, z) \in \mathbb{Q}^3 \mid x = y < z \vee x = z < y\}.$$

**Lemma 36.** *Let  $f$  be a binary operation that preserves  $<$  and violates the relation  $S$ . Then  $\{f, pp\}$  generates  $min$ ,  $mi$ , or  $mx$ .*

*Proof.* By Proposition 25, 27, and 29, it suffices to show that  $\{f, pp\}$  generates an operation providing min-intersection, min-union, or min-xor closure. If  $f$  violates  $\leq$ , then we are immediately done by Lemma 35. So we further assume that  $f$  preserves  $\leq$ .

Because  $f$  preserves  $<$  and violates  $S$ , we can assume without loss of generality (possibly after swapping arguments) that there are  $x_1, x_2, y_1, y_2 \in \mathbb{Q}$  such that  $x_1 < x_2$ ,  $y_1 < y_2$  and  $t := (f(x_1, y_1), f(x_2, y_1), f(x_1, y_2)) \notin S$ . Because  $f$  preserves  $\leq$  we have that  $f(x_1, y_1) \leq f(x_2, y_1)$  and  $f(x_1, y_1) \leq f(x_1, y_2)$ . Since  $t \notin S$ , there are only two possibilities:

1.  $t[1] < t[2]$  and  $t[1] < t[3]$ . In this case, choose infinite sequences  $x_3 < x_4 < \dots$  and  $y_3 < y_4 < \dots$  such that  $x_2 < x_3$ ,  $y_2 < y_3$ . Because  $f$  preserves  $\leq$ , we have for all  $i > 1$  that  $f(x_2, y_1) \leq f(x_i, y_1)$  and  $f(x_1, y_2) \leq f(x_1, y_i)$ . Since  $t[1] = f(x_1, y_1) < t[2] = f(x_2, y_1)$  we have that  $f(x_1, y_1) < f(x_i, y_1)$  for all  $i > 1$ , and since  $t[1] = f(x_1, y_1) < t[3] = f(x_1, y_2)$  we have that  $f(x_1, y_1) < f(x_1, y_i)$  for all  $i > 1$ . Hence,  $f$  provides min-intersection closure.

2.  $t[1] = t[2] = t[3]$ . In this case we can choose infinite sequences  $x'_2 < x'_3 < \dots$  and  $y'_2 < y'_3 < \dots$  such that  $x_1 < x'_2$ ,  $y_1 < y'_2$ , and for all  $i > 1$ ,  $x'_i < x_2$  and  $y'_i < y_2$ . As  $f$  preserves  $\leq$ , we see that  $f(x'_i, y_1) = f(x_1, y_1) = f(x_1, y'_i)$  for all  $i > 1$  and thus  $f$  provides min-union closure.  $\square$

## 5 Algorithms for shuffle-closed languages

In this section we present three algorithms, for the languages preserved by  $mi$ , by  $min$ , and by  $mx$ , respectively. All three algorithms follow a common strategy. They are searching for a variable that can have the minimal value in a solution. If they have found such a variable, say  $x$ , the algorithms add equalities and inequalities that are implied by all constraints under the assumption that  $x$  denotes the minimal value in *all* solutions. Next, the algorithms recursively solve the instance consisting of the projections of all constraints to the variables that do not denote the minimal value in all solutions. We later show that for languages preserved by  $pp$  it is true that if the instance has a solution, it also has a solution that satisfies all the additional constraints.

For the formulation of the algorithms and their correctness proofs it will be convenient to work with an expanded constraint language, that contains the binary relation  $=$  for the equality relation. We also add to the temporal constraint language  $\Gamma$  several other temporal relations that are primitive positive definable in  $\Gamma$ .

**Definition 12.** *Let  $R$  be an  $n$ -ary temporal relation and  $L = \{p_1, \dots, p_k\} \subseteq [n]$  where  $p_1 < \dots < p_k$ . Let  $\{q_1, \dots, q_l\}$  be  $[n] \setminus L$ . Then the ordered projection of  $R$  to  $L$  is the  $k$ -ary relation  $R'$  with the primitive positive definition*

$$R'(x_{p_1}, \dots, x_{p_k}) \equiv \exists x_{q_1}, \dots, x_{q_l}. R(x_1, \dots, x_n) \wedge \bigwedge_{i \in [n] \setminus L, j \in L} x_i < x_j.$$

Note that if  $\Gamma$  is a finite temporal constraint language, then there are only finitely many projections and ordered projections of relations in  $\Gamma$ . In case that there is a primitive positive definition of  $<$  in  $\Gamma$ , ordered projections are primitive positive definable. By Lemma 7, we can assume in this case that  $\Gamma$  contains all relations that can be defined by ordered projections from relations in  $\Gamma$ .

To formally introduce our algorithms, we also need the concept of an ordered projection of *instances* of the CSP.

**Definition 13.** *Let  $\Gamma$  be a temporal constraint language that contains all ordered projections of relations from  $\Gamma$ . Let  $\Phi$  be an instance of  $\text{CSP}(\Gamma)$  and  $X \subseteq V(\Phi)$ . Then the ordered projection of  $\Phi$  to  $X$  is the instance of  $\text{CSP}(\Gamma)$  that contains for each constraint  $R(x_1, \dots, x_n)$  in  $\Phi$ , with not necessarily distinct variables  $x_1, \dots, x_n$ , the constraint  $R'(x_{k_1}, \dots, x_{k_l})$  where  $k_1 < \dots < k_l$  are such that  $\{k_1, \dots, k_l\} = \{k \in [n] \mid x_k \in X\}$ , and  $R'$  is the ordered projection of  $R$  to  $\{k_1, \dots, k_l\}$ .*

Let  $\Phi$  be an instance of a temporal CSP.

**Definition 14.** *If  $\psi = R(x_1, \dots, x_k)$  is a constraint from  $\Phi$ , then a subset  $X$  of the variables of  $\psi$  is called a *min-set* (of  $\psi$ ) if there exists a  $k$ -tuple  $t$  satisfying  $\psi$  such that  $x \in X$  iff the value for  $x$  in  $t$  is the minimum of all entries of  $t$ . A set of variables  $S \subseteq V(\Phi)$  is called *free* iff it is non-empty and for all constraints  $R(x_1, \dots, x_k)$  in  $\Phi$  the set  $S \cap \{x_1, \dots, x_k\}$  is either empty or a min-set of  $R$ .*

We will show how to use the concept of freeness to solve instances of  $\text{CSP}(\Gamma)$  for shuffle closed temporal constraint languages.

**Lemma 37.** *Let  $\Phi$  be an instance of  $\text{CSP}(\Gamma)$  for some shuffle closed  $\Gamma$ , and let  $S$  be a free set of variables of  $\Phi$ . Then  $\Phi$  has a solution if and only if the ordered projection  $\Phi'$  of  $\Phi$  to  $V(\Phi) \setminus S$  has a solution.*

```

Solve( $\Phi$ ) {
  // Input: An instance  $\Phi$  of CSP( $\Gamma$ )
  // for a shuffle closed temporal language  $\Gamma$ 
  // Output: A solution  $s$  to  $\Phi$ , or false if there is no solution.
   $i := 0$ 
  while  $V(\Phi) \neq \emptyset$  do begin
     $S := \text{FindFreeSet}(\Phi)$ 
    if  $S = \text{false}$  then return false
    for each  $x \in S$  do  $s(x) := i$ 
     $i := i + 1$ 
     $\Phi :=$  ordered projection of  $\Phi$  to  $V(\Phi) \setminus S$ 
  end
  return  $s$  }

```

Figure 6: An algorithm that efficiently solves instances of a shuffle closed constraint language if free sets can be computed efficiently.

*Proof.* First suppose  $\Phi'$  has a solution  $s'$ . Let  $\psi = R(x_1, \dots, x_m)$  be a constraint of  $\Phi$  such that  $V(\psi) \cap S = \{x_{p_1}, \dots, x_{p_k}\} \neq \emptyset$ . Let  $\{x_{q_1}, \dots, x_{q_l}\} = V(\psi) \setminus S$  for  $q_1 < \dots < q_l$ . By the definition of an ordered projection, there is a tuple  $t_1 \in R$  such that  $s'(x_i) = t_1[i]$  for all  $i \in \{q_1, \dots, q_l\}$ . Since  $V(\psi) \cap S$  is a min-set of  $R$ , there is a tuple  $t_2 \in R$  such that  $M(t_2) = \{p_1, \dots, p_k\}$ . Let  $\alpha \in \text{Aut}(\mathbb{Q}; <)$  be such that  $\alpha$  maps the minimal value of  $t_2$  to 0. Because  $R$  is preserved by  $pp$ , the tuple  $t_3 := pp(\alpha(t_2), t_1)$  is in  $R$ . It is easy to verify that  $M(t_3) = \{p_1, \dots, p_k\}$  and that there is  $\beta \in \text{Aut}(\mathbb{Q}; <)$  such that  $\beta(t_3[i]) = s'(x_i)$  for  $i \in \{q_1, \dots, q_l\}$ . Because we can find such a tuple for all the constraints  $\psi$  in  $\Phi$  where  $V(\psi) \cap S \neq \emptyset$ , we conclude that a solution  $s'$  of  $\Phi'$  can be extended to a solution  $s$  of  $\Phi$  by setting all the variables in  $S$  to some value that is smaller than the smallest value in  $\{s'(x) \mid x \in V(\Phi')\}$ . Clearly, all the constraints  $\psi$  in  $\Phi$  with  $V(\psi) \cap S = \emptyset$  or  $V(\psi) \subset S$  are satisfied by  $s$  as well.

Now suppose that  $\Phi$  has a solution  $s$ . Let  $x_1, \dots, x_n$  be the variables of  $\Phi$ , and let  $\{x_{r_1}, \dots, x_{r_{|S|}}\}$  be  $S$ . Let  $s'$  be a mapping from  $V(\Phi)$  to  $\mathbb{Q}$  such that  $M((s'(x_1), \dots, s'(x_n))) = \{r_1, \dots, r_{|S|}\}$ , and  $s'(x) = s(x)$  for  $x \in V(\Phi) \setminus S$ . We claim that  $s'$  is a solution for  $\Phi'$ . Let  $\psi = R(y_1, \dots, y_m)$  be a constraint of  $\Phi$  such that  $V(\psi) \cap S \neq \emptyset$ . Clearly,  $t_1 := (s(y_1), \dots, s(y_m))$  is in  $R$  since  $s$  is a solution of  $\Phi$ . Let  $\{y_{p_1}, \dots, y_{p_l}\}$  be  $S \cap \{y_1, \dots, y_m\}$ . Since  $\{y_{p_1}, \dots, y_{p_l}\}$  is a min-set of  $R$ , there is a tuple  $t_2 \in R$  such that  $M(t_2) = \{p_1, \dots, p_l\}$ . Let  $\alpha \in \text{Aut}(\mathbb{Q}; <)$  be such that  $\alpha$  maps the minimal value of  $t_2$  to 0. Because  $R$  is preserved by  $pp$ , the tuple  $t_3 := pp(\alpha(t_2), t_1)$  is in  $R$ . It is easy to verify that  $M(t_3) = \{p_1, \dots, p_l\}$ , and that there is an automorphism  $\beta$  such that  $\beta(t_3)[i] = s(y_i)$  for  $i \in [m] \setminus \{p_1, \dots, p_l\}$ . Clearly, the restriction of  $s'$  to  $V(\Phi) \setminus S$  is a solution to the ordered projection  $\Phi'$  of  $\Phi$  to  $V(\Phi) \setminus S$  since  $s'$  also satisfies all the inequalities imposed by the ordered projection. Therefore  $\Phi'$  is satisfied by  $s'$ .  $\square$

The above lemma asserts that if we are able to identify a free set for instances of CSP( $\Gamma$ ) for a shuffle-closed temporal language  $\Gamma$  in polynomial time, then we also have a polynomial time algorithm that solves CSP( $\Gamma$ ). The running time of the algorithm is  $O(n \cdot (m + t(n, m)))$ , where  $n = |V|$ ,  $m$  is the number of constraints in  $\Phi$ , and  $t(n, m)$  is the running time of the procedure that computes the free set of an instance with  $n$  variables and  $m$  constraints.

## 5.1 An algorithm for languages preserved by *min*

Now, we concentrate on the problem to find a free set of  $\Phi$  if  $\Gamma$  is preserved by the operation *min*.

Let  $\psi = R(x_1, \dots, x_k)$  be a constraint where  $R$  is from  $\Gamma$  and let  $L$  be a subset of  $\{x_1, \dots, x_k\}$ . Let  $A_1, \dots, A_l$  be all min-sets of  $\psi$  that are contained in  $L$ . When  $l \geq 1$ , i.e., when such min-sets exist, there is a unique set  $A_j$ ,  $j \in [l]$ , with the property that  $A_i \subseteq A_j$  for all  $i \in [l]$ , because  $R$  is preserved by *min*, and thus min-union closed by Lemma 24. We call this min-set the *maximal*

```

FindFreeSetUC( $\Phi$ ) {
  // Input: An instance  $\Phi$  of CSP( $\Gamma$ ) with variables  $V$ 
  // for a temporal constraint language  $\Gamma$  preserved by min.
  // Output: A free set  $S \subseteq V$  of  $\Phi$ , or false.
  // If the output is false,  $\Phi$  is unsatisfiable
   $S := V$ 
  recheck := true
  while recheck do begin
    recheck := false
    for all  $\psi \in \Phi$  do begin
      if  $S \cap V(\psi) \neq \emptyset$  then begin
         $S := (S \setminus V(\psi)) \cup$  the maximal min-set of  $\psi$  contained in  $S \cap V(\psi)$ 
        if  $S$  changed then recheck := true
      end
    end
  end
  if  $S \neq \emptyset$  then return  $S$ 
  else return false
end }

```

Figure 7: A polynomial time algorithm that computes free sets for constraint languages preserved by *min*.

*min*-set of  $\psi$  contained in  $L$ . Note that for some  $L$  it could be that  $l = 0$ , i.e.,  $L$  does not contain min-sets of  $R$ .

Figure 7 shows our procedure for finding a free set for a min-union closed constraint language. It is straightforward to check that the procedure `FindFreeSetUC` has a running time  $O(nm)$ , where  $n$  is the number of variables and  $m$  is the number of constraints of  $\Phi$ .

**Lemma 38.** *The procedure FindFreeSetUC in Figure 7 returns a free set of  $\Phi$ , or false. If it returns false,  $\Phi$  is unsatisfiable.*

*Proof.* Suppose that the algorithm returns a (non-empty) set  $S$ . Then *recheck* must be set to **false**. Therefore, for all constraints  $R(x_1, \dots, x_k)$  of  $\Phi$  such that  $S \cap \{x_1, \dots, x_k\} \neq \emptyset$  the maximal min-set of  $\psi$  contained in  $S$  equals  $S \cap \{x_1, \dots, x_k\}$ . We conclude that  $S$  is a free set of  $\Phi$ .

We now have to argue that in case that  $\Phi$  is satisfiable, the algorithm does not return **false** (i.e., it finds a free set). If  $\Phi$  has a solution, there is some set  $S'$  of variables that have the minimal value in this solution. At the beginning of the procedure,  $S$  is set to  $V$  and therefore  $S' \subseteq S$ . We show that  $S' \subseteq S$  during the entire execution of the procedure. Let  $\psi = R(x_1, \dots, x_k)$  be a constraint from  $\Phi$ . Because  $S' \cap \{x_1, \dots, x_k\}$  is a min-set of  $\psi$  that is contained in  $S$ , the maximal min-set of  $\psi$  added to  $S \setminus \{x_1, \dots, x_k\}$  certainly contains  $S' \cap \{x_1, \dots, x_k\}$ . Therefore, after the modification to  $S$  it still holds that  $S \supseteq S'$ . When the procedure terminates, it returns the set  $S$ , because  $\emptyset \neq S' \subseteq S$ .  $\square$

**Theorem 39.** *If  $\Gamma$  is preserved by *min* there is an algorithm solving CSP( $\Gamma$ ) in time  $O(n^2m)$ .*

*Proof.* We use the procedure `FindFreeSetUC` in Figure 7 for the subroutine `FindFreeSet` in Figure 6. Then Lemma 37 and Lemma 38 imply the correctness of the resulting algorithm.  $\square$

## 5.2 An algorithm for languages preserved by *mi*

In this subsection we describe how to find free sets in instances of CSP( $\Gamma$ ) for languages  $\Gamma$  that are preserved by *mi*. We define the notion of a minimal min-set: Let  $\psi = R(x_1, \dots, x_k)$  be a constraint from an instance  $\Phi$  of CSP( $\Gamma$ ), and let  $L \subseteq \{x_1, \dots, x_k\}$ . Let  $A_1, \dots, A_l$  be all min-sets of  $\psi$  that contain  $L$ . Because  $R$  is preserved by *mi*, and thus is min-intersection closed by

```

FindFreeSetIC( $\Phi$ ) {
  // Input: An instance  $\Phi$  of CSP( $\Gamma$ ) where  $\Gamma$  is preserved by  $mi$ 
  // Output: A free set  $S \subseteq V(\Phi)$  of  $\Phi$ , or false
  // If the output is false,  $\Phi$  is unsatisfiable
  for all  $x \in V(\Phi)$  do begin
     $S := \{x\}$ 
     $recheck := \mathbf{true}; correct := \mathbf{true}$ 
    while  $recheck \wedge correct$  do begin
       $recheck := \mathbf{false}$ 
      for all constraints  $\psi$  of  $\Phi$  such that  $(V(\psi) \cap S) \neq \emptyset$  do begin
        if there is no min-set of  $\psi$  containing  $S \cap V(\psi)$  then  $correct := \mathbf{false}$ 
        else begin
           $S := S \cup$  the minimal min-set of  $\psi$  containing  $S \cap V(\psi)$ 
          if  $S$  changed then  $recheck := \mathbf{true}$ 
        end
      end
    end
  end
  if  $correct$  then return  $S$ 
end
return false}

```

Figure 8: A polynomial time algorithm that computes free sets for min-intersection and shuffle closed constraint languages.

Lemma 26, there is a min-set  $A_j$  of  $\psi$  that is a subset of every min-set containing  $L$ . We call  $A_j$  the *minimal min-set of  $R$  containing  $L$* .

The procedure for finding a free set for min-intersection closed constraint languages is given in Figure 8. It is straightforward to verify that the above algorithm runs in time  $O(n^2m)$  where  $n$  is the number of variables and  $m$  is the number of constraints in  $\Phi$ .

**Lemma 40.** *The procedure FindFreeSetIC in Figure 8 returns a free set  $S$  of  $\Phi$ , or **false**. If it returns **false**,  $\Phi$  is unsatisfiable.*

*Proof.* Suppose that the algorithm returns a set  $S$ . The variable *correct* must then be equal to **true**. When the while loop terminates, *recheck* equals **false**, and so for all constraints  $\psi \in \Phi$  such that  $V(\psi) \cap S \neq \emptyset$  the set  $S$  did not change. This implies that for all these constraints the minimal min-set of  $\psi$  containing  $S \cap V(\psi)$  is equal to  $S \cap V(\psi)$ . We conclude that  $S$  is a free set of  $\Phi$ .

We now have to argue that in case that  $\Phi$  is satisfiable, the algorithm does not return **false**. If  $\Phi$  has a solution, then there is some set  $S'$  of variables that have the minimal value in this solution. Consider a run of the while loop in the procedure FindFreeIC for some variable  $x \in S'$ . In the beginning, it holds that  $S = \{x\} \subseteq S'$ . For each constraint  $\psi$  from  $\Phi$  we have that  $S' \cap V(\psi)$  is a min-set of  $\psi$  if  $S' \cap V(\psi)$  is non-empty. Therefore, the program variable *correct* cannot be set to **false** while  $S \subseteq S'$ . Because we always add only variables of the *minimal* min-set of  $\psi$  containing  $S \cap V(\psi)$  to  $S$ , all these variables are always in  $S'$ . Therefore,  $S$  remains a subset of  $S'$  all the time, and the algorithm does not return **false**.  $\square$

**Theorem 41.** *If  $\Gamma$  is preserved by  $mi$  there is an algorithm solving CSP( $\Gamma$ ) in time  $O(n^3m)$ .*

*Proof.* We use the procedure FindFreeSetIC in Figure 8 for the sub-routine FindFreeSet in Figure 6. Lemma 37 and Lemma 40 imply the correctness of these algorithms.  $\square$

### 5.3 An algorithm for languages preserved by $mx$

Finally, we consider languages  $\Gamma$  preserved by  $mx$ . Let  $R$  be a relation from  $\Gamma$ . For a tuple  $t \in R$ , we define  $\chi_{min}(t)$  to be a vector from  $\{0, 1\}^k$  such that  $\chi_{min}(t)[i] = 1$  if and only if  $t[i]$  is minimal

in  $t$ . We define  $\chi_{min}(R)$  to be  $\{\chi_{min}(t) \mid t \in R\}$ . Since  $R$  is preserved by  $mx$  and hence min-xor closed by Lemma 28, the set  $\chi_{min}(R)$  is closed under addition of distinct vectors over  $GF(2)$  and so  $\chi_{min}(R) \cup \{0^k\}$  is exactly the set of solutions of a system of linear equations; see e.g. [19].

**Theorem 42.** *If  $\Gamma$  is preserved by  $mx$  there is an algorithm solving  $CSP(\Gamma)$  in time  $O(n^4)$ .*

*Proof.* To find a free set of variables of an instance  $\Phi$  of  $CSP(\Gamma)$  (if it exists), we first construct a system  $S$  of linear equations over  $GF(2)$  with variable set  $\{x_v \mid v \in V\}$  and linear equations as described above for each constraint in  $\Phi$ . It is well-known that a solution of  $S$  that is distinct from  $0^k$  can be computed in cubic time (by Gaussian elimination). If there is such a solution, then the set of variables mapped to 1 is a free set of  $\Phi$ . If the system has no such solution, then there is no free set of variables, and there is no solution for  $\Phi$ . Now the claim follows from Lemma 37 as in Theorem 39 and Theorem 41.  $\square$

## 6 The complete classification

Temporal constraint languages where not all first-order definable relations are primitive positive definable can be divided into four (non-disjoint) groups: those preserved by a constant operation, by  $pp$ , by  $dual\text{-}pp$ , or by an operation called  $lex$ , which will be introduced in the next subsection<sup>3</sup>. None of the three polymorphisms  $pp$ ,  $dual\text{-}pp$ , and  $lex$  alone guarantees tractability of  $CSP(\Gamma)$ . An illustration of the classification result for the languages that preserve  $<$  can be found in Figure 15.

### 6.1 The operations $lex$ and $ll$

An important class of temporal constraint languages are the languages preserved by the operation  $lex$ . Let  $lex$  be a binary operation on  $\mathbb{Q}$  such that  $lex(a, b) < lex(a', b')$  if either  $a < a'$ , or  $a = a'$  and  $b < b'$ . Clearly, such an operation exists; by Observation 1, all such operations generate the same clone. For our definition of  $lex$ , we can choose an arbitrary operation with these properties. Note that  $lex$  is injective. We also write

- $lex_{y,x}$  for the operation  $(x, y) \mapsto lex(y, x)$ ,
- $lex_{y,-x}$  for the operation  $(x, y) \mapsto lex(y, -x)$ ,
- $lex_{x,-y}$  for the operation  $(x, y) \mapsto lex(x, -y)$ ,
- $lex_{x,y}$  for the operation  $(x, y) \mapsto lex(x, y)$ ,
- $p_x$  for the operation  $(x, y) \mapsto x$ , and
- $p_y$  for the operation  $(x, y) \mapsto y$ .

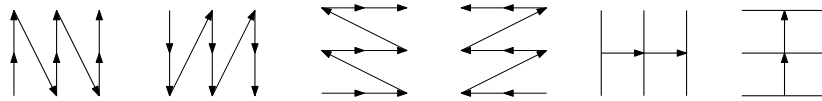


Figure 9: Illustrations of the six basic operations  $lex_{x,y}$ ,  $lex_{x,-y}$ ,  $lex_{y,x}$ ,  $lex_{y,-x}$ ,  $p_x$ ,  $p_y$ .

It is easy to see that the relation  $Betw$  is preserved by  $lex$ . Therefore, we are interested in further restrictions of languages preserved by  $lex$  that imply tractability of the corresponding CSP.

A large tractable temporal constraint language has been introduced in [5]. The language is defined in terms of a binary polymorphism, denoted by  $ll$ , and again it has a dual version, which is

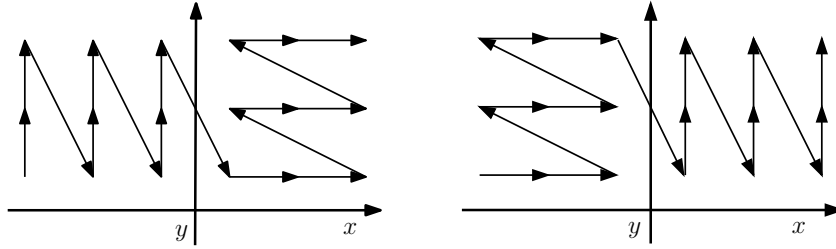


Figure 10: A visualization of  $ll$  (left) and  $dual-ll$  (right).

tractable as well. It was shown in [5] that the constraint language  $\text{Inv}(ll)$  strictly contains the class of Ord-Horn constraints, a well-known tractable constraint language in temporal reasoning [37].

Let  $ll$  be a binary operation on  $\mathbb{Q}$  such that  $ll(a, b) < ll(a', b')$  if

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

All operations satisfying these conditions are by definition injective, and they all generate the same clone. For an illustration of  $ll$  and its dual, see Figure 10. It is easy to see that  $ll$  generates  $lex$ .

## 6.2 Operations generating $ll$ , $dual-ll$ , or $lex$

In this section we present operations that generate  $ll$ ,  $dual-ll$ , or  $lex$ .

To describe properties of an operation on restricted subsets of the domain  $\mathbb{Q}$ , the following concepts are useful: If  $S_1, \dots, S_d$  are sets, we call a set of the form  $S_1 \times \dots \times S_d$  a *grid*, and also write  $S^d$  for a product of the form  $S \times \dots \times S$  with  $d$  factors. A  $[k]^d$ -*subgrid* of a grid  $S_1 \times \dots \times S_d$  is a subset of  $S_1 \times \dots \times S_d$  of the form  $S'_1 \times \dots \times S'_d$ , where  $S'_i$  is a  $k$ -element subset of  $S_i$ . We say that a  $k$ -ary operation  $f$  *behaves like a  $k$ -ary operation  $g$  on a subgrid  $G$*  of  $\mathbb{Q}^k$  if for all  $t, t' \in G$  we have  $f(t) \leq f(t')$  iff  $g(t) \leq g(t')$ . That is, the weak linear order induced by  $f$  on the tuples from  $G$  (in the sense as in Observation 1 in Section 2.5) is the same as the weak linear order induced on these tuples by  $g$ . If  $f$  behaves like  $g$  on the entire set  $\mathbb{Q}^k$ , we simply say that  $f$  *behaves like  $g$* .

Let  $\mathbb{Q}^+$  denote the set of all positive rational numbers, and let  $\mathbb{Q}_0^-$  denote  $\mathbb{Q} \setminus \mathbb{Q}^+$ .

**Definition 15.** Let  $f, g$  be from  $\mathbb{Q}^2 \rightarrow \mathbb{Q}$ . Then  $[f|g]$  denotes an arbitrary operation from  $\mathbb{Q}^2 \rightarrow \mathbb{Q}$  with the following properties. For all  $x, x', y, y' \in \mathbb{Q}$ ,

- if  $x \leq 0$  and  $x' > 0$  then  $[f|g](x, y) < [f|g](x', y')$ ;
- $[f|g]$  behaves like  $f$  on  $\mathbb{Q}_0^- \times \mathbb{Q}$ ;
- $[f|g]$  behaves like  $g$  on  $\mathbb{Q}^+ \times \mathbb{Q}$ ;

For example, if  $f = lex_{x,y}$  and  $g = lex_{x,y}$ , then  $[f|g]$  behaves like  $ll$ .

**Lemma 43.** Let  $f, g \in \{lex_{x,y}, lex_{x,-y}, lex_{y,x}, lex_{y,-x}, p_x, p_y\}$ , and let  $f'$  ( $g'$ ) be  $lex_{x,y}$  if  $f$  ( $g$ ) is dominated by the first argument, and  $lex_{y,x}$  otherwise. Then  $\{lex, [f|g]\}$  generates  $[f'|g'](x, y)$ .

<sup>3</sup>In the terminology of [2], the operations  $pp$ ,  $dual-pp$ , and  $lex$  define *maximal* constraint languages.

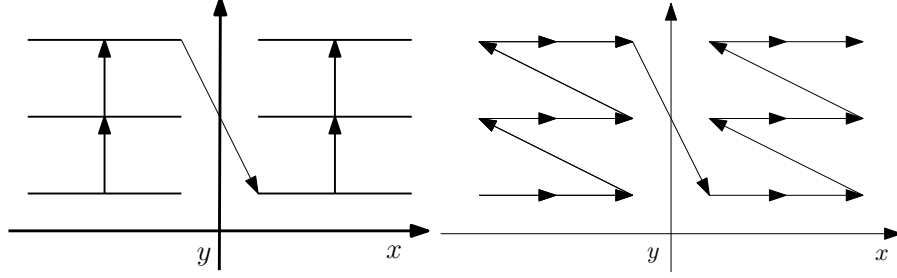


Figure 11: An illustration of the operation  $[p_y | p_y]$  (on the left) and the operation  $[lex_{y,x} | lex_{y,x}]$  (on the right).

*Proof.* By Lemma 12 it suffices to show that every relation  $R$  preserved by  $lex$  and  $[f|g]$  is preserved by  $[f'|g']$ . So let  $R$  be an arbitrary relation preserved by  $lex$  and  $[f|g]$ , let  $k$  denote its arity, and let  $t_1, t_2$  be  $k$ -tuples from  $R$ . We have to show that  $t_3 := [f'|g'](t_1, t_2)$  is in  $R$ .

Let  $\alpha \in \text{Aut}(\mathbb{Q}; <)$  be such that for each entry  $x$  of  $t_1$  and for each entry  $y$  of  $t_2$ , the value of  $\alpha(lex(x, y))$  is negative when  $x \leq 0$ , and positive otherwise. We will show that there is an automorphism of  $(\mathbb{Q}; <)$  that maps the tuple

$$s := [f|g](\alpha(lex(t_1, t_2)), lex(t_2, t_1))$$

to  $t_3$ , which proves that  $t_3$  is in  $R$ . It suffices to show for  $j_1, j_2 \in [k]$  that

$$s[j_1] \leq s[j_2] \text{ if and only if } t_3[j_1] \leq t_3[j_2]. \quad (2)$$

We can assume that  $t_1[j_1] \leq t_1[j_2]$  by exchanging the name of  $j_1$  and  $j_2$  if necessary, and distinguish three cases:

- $t_1[j_1] \leq 0, t_1[j_2] > 0$ . Then  $t_3[j_1] < t_3[j_2]$  by definition of  $[f'|g']$ . Since for  $j \in [k]$ , the value of  $\alpha(lex(t_1[j], t_2[j]))$  is positive if and only if the value of  $t_1[j]$  is positive, we have  $s[j_1] < s[j_2]$  by definition of  $[f|g]$ . Thus we have verified (2) in this case.
- $t_1[j_2] \leq 0$ . Then, writing  $a[j]$  for  $lex(t_1[j], t_2[j])$  and  $b[j]$  for  $lex(t_2[j], t_1[j])$ , we have

$$\begin{aligned} t_3[j_1] \leq t_3[j_2] & \text{ iff } f'(t_1[j_1], t_2[j_1]) \leq f'(t_1[j_2], t_2[j_2]) \\ & \text{ iff } f(a[j_1], b[j_1]) \leq f(a[j_2], b[j_2]) \\ & \text{ iff } f(\alpha(a[j_1]), b[j_1]) \leq f(\alpha(a[j_2]), b[j_2]) \\ & \text{ iff } s[j_1] \leq s[j_2] \end{aligned}$$

- $t_1[j_1] > 0$ . This case is analogous to the previous one and left to the reader. □

**Lemma 44.** For  $f, g \in \{p_y, lex_{y,x}\}$  the operation  $[f|g]$  generates  $[lex_{x,y}|g]$ .

If particular, for  $f = g = lex_{y,x}$  the lemma shows that  $[f|g]$  generates  $\mathbb{ll}$ . For  $f = g = p_y$ , the lemma shows that  $[f|g]$  generates  $[lex_{x,y}|p_y]$  and in particular  $lex_{x,y}$ . See Figure 11 for illustrations of those two cases.

*Proof of Lemma 44.* We show that every relation  $R$  preserved by  $[f|g]$  is preserved by  $[lex_{x,y}|g]$ , and conclude by Lemma 12 that  $[f|g]$  generates  $[lex_{x,y}|g]$ . So let  $R$  be an arbitrary relation preserved by  $[f|g]$ , let  $k$  denote its arity, and let  $t_1, t_2$  be  $k$ -tuples from  $R$ . We have to show that  $t_3 := [lex_{x,y}|g](t_1, t_2)$  is in  $R$ .

Let  $l$  denote the number of non-positive values in  $t_1$ . We take  $\alpha_1, \dots, \alpha_l$  from  $\text{Aut}(\mathbb{Q}; <)$  such that  $\alpha_i$  maps all but the  $i$  smallest values in  $t_1$  to positive values. We define a sequence of tuples  $s_1, \dots, s_l$  as follows:  $s_1 = t_2$ , and for  $i \geq 2$

$$s_i := [f|g](\alpha_i(t_1), s_{i-1}).$$

Clearly, for all  $i \in [l]$  the tuple  $s_i$  is in  $R$ . We will show that there is an automorphism of  $(\mathbb{Q}; <)$  that maps  $s_l$  to  $t_3$ , which proves that  $t_3$  is also in  $R$ . By symmetry it is enough to show for  $j_1, j_2 \in [k]$  with  $t_1[j_1] \leq t_1[j_2]$  that

$$s_l[j_1] \leq s_l[j_2] \text{ if and only if } t_3[j_1] \leq t_3[j_2]. \quad (3)$$

We distinguish three cases:

- $t_1[j_1] = t_1[j_2] \leq 0$ . Since  $\alpha_i(t_1[j_1]) = \alpha_i(t_1[j_2])$  for all  $i \in [l]$ , we have  $s_i[j_1] \leq s_i[j_2]$  if and only if  $s_1[j_1] \leq s_1[j_2]$ . Since  $s_1 = t_2$  and  $t_1[j_1] \leq 0$ , and because  $f$  is dominated by the second argument,  $s_1[j_1] \leq s_1[j_2]$  if and only if  $t_3[j_1] \leq t_3[j_2]$ , which proves (3).
- $t_1[j_1] < t_1[j_2]$ ,  $t_1[j_1] \leq 0$ . Let  $i \in [l]$  be such that  $\alpha_i(t_1[j_1]) \leq 0$  and  $\alpha_i(t_1[j_2]) > 0$ . By definition of  $[f|g]$  we see that  $s_i[j_1] < s_i[j_2]$ . Because  $\alpha_i(t_1[j_1]) < \alpha_i(t_1[j_2])$  for all  $i \in [l]$ , and because  $[f|g]$  preserves  $<$ , by induction on  $i' \geq i$  we have that  $s_{i'}[j_1] < s_{i'}[j_2]$ . In particular,  $s_l[j_1] < s_l[j_2]$ . On the other hand,  $t_3[j_1] < t_3[j_2]$  by definition of  $\text{lex}_{x,y}$  and  $[\text{lex}_{x,y}|g]$ , and so (3) also holds in this case.
- $t_1[j_1] > 0$ . Observe that by the choice of  $l$  we have  $\alpha_i(t_1[j_1]) > 0$  for all  $i \in [l]$ . Thus (3) holds, because both  $[f|g]$  and  $[\text{lex}_{x,y}|g]$  behave like  $g$  on  $\mathbb{Q}^+ \times \mathbb{Q}$ .

□

### 6.3 The product Ramsey theorem

In the proof of the classification result, we make essential use of the so-called *product Ramsey theorem (PRT)*, which can be easily derived from the classical infinite Ramsey theorem; see [38] for a general introduction to Ramsey theory. Subsets of a set of cardinality  $m$  will be called  $m$ -subsets in the following. Let  $\binom{S}{m}$  denote the set of all  $m$ -subsets of  $S$ . The classical Ramsey theorem can be stated as follows.

**Theorem 45** (Finite version of Ramsey's theorem; see e.g. Theorem 5.6.2 in [28]). *For all positive integers  $r, m, k$  there is a positive integer  $l = \mathbf{R}(r, m, k)$  such that for every every  $\chi : \binom{[l]}{m} \rightarrow [r]$  there exists a  $k$ -subset  $S$  of  $[l]$  such that  $\chi$  is constant on  $\binom{S}{m}$ .*

The following theorem is known; however, we give the short proof from the classical Ramsey theorem for the convenience of the reader. We also refer to mappings  $f : S \rightarrow [r]$  as a *coloring* of  $S$  (with the  $r$  colors  $1, \dots, k$ ). The notion of grids and subgrids has been introduced in Section 6.2.

**Theorem 46** (Product Ramsey Theorem). *For all positive integers  $d, r, m$ , and  $k \geq m$ , there is a positive integer  $L = \mathbf{R}(d, r, m, k)$  such that for every coloring of the  $[m]^d$  subgrids of  $[L]^d$  with  $r$  colors there exists a monochromatic  $[k]^d$  subgrid  $G$  of  $[L]^d$ , i.e.,  $G$  is such that all its  $[m]^d$  subgrids have the same color.*

*Proof.* Let  $d, r, m$ , and  $k \geq m$  be positive integers. We claim that we can choose  $L = \mathbf{R}(d, r, m, k)$  to be  $\mathbf{R}(r, dm, dk)$ . To verify this, let  $\chi$  be a coloring of the  $[m]^d$  subgrids of  $[L]^d$  with  $r$  colors. We have to find a monochromatic subgrid of  $[L]^d$ .

We use  $\chi$  to define an  $r$ -coloring  $\xi$  of the  $dm$ -subsets of  $[L]$  as follows. Let  $S = \{s_1, s_2, \dots, s_{dm}\}$  be a  $dm$ -subset of  $[L]$ , with  $s_1 < s_2 < \dots < s_{dm}$ . Then define

$$\xi(S) = \chi(\{s_1, \dots, s_m\} \times \dots \times \{s_{m(d-1)+1}, \dots, s_{dm}\}).$$

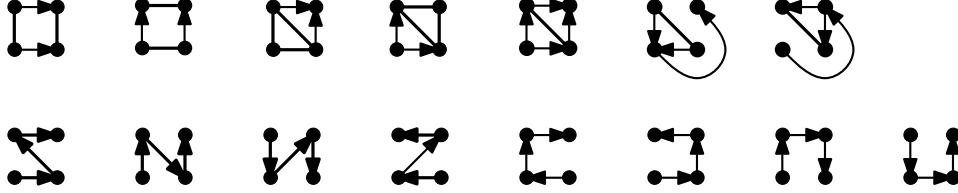


Figure 12: All weak linear orders on  $[2] \times [2]$  grids.



Figure 13: If  $f$  is a binary injective operation that preserves  $<$  and is homogeneous on  $S \times T$ , then all  $[2] \times [2]$  subgrids of  $S \times T$  have one out of the following four weak linear orders.

By the second part of Theorem 45, there is a  $dk$ -subset  $\{t_1, t_2, \dots, t_{dk}\}$  of  $[L]$  such that  $\xi$  is constant on the  $dm$ -element subsets of  $\{t_1, \dots, t_{dk}\}$ . Suppose that  $t_1 < t_2 < \dots < t_{dk}$ . Then  $G = \{t_1, \dots, t_k\} \times \dots \times \{t_{k(d-1)+1}, \dots, t_{dk}\}$  is a subgrid of  $[L]^d$  that is monochromatic with respect to  $\chi$ .  $\square$

Since we use the above theorem for  $d = m = 2$  and since  $r$  is always obvious from the context, we use just  $\mathbf{R}(k)$  instead of  $\mathbf{R}(d, r, m, k)$ .

**Lemma 47.** *Let  $f$  be a binary operation that preserves  $<$ , and let  $S_1, T_1 \subseteq \mathbb{Q}$  be sets of cardinality at least  $\mathbf{R}(k)$ . Then there exist sets  $S_2 \subseteq S_1, T_2 \subseteq T_1$  of cardinality  $k$  such that  $f$  behaves on  $S_2 \times T_2$  like one out of the following operations.*

- $p_x$  or  $p_y$  (a projection to the first or to the second argument);
- $lex_{x,y}$  or  $lex_{y,x}$ ;
- $lex_{x,-y}$  or  $lex_{y,-x}$ .

*Proof.* We apply the product Ramsey theorem for  $d = m = 2$ , and color the  $[2] \times [2]$  subgrids of  $S_1 \times T_1$  according to the weak linear order defined by  $f$  on these subgrids. The possible weak linear orders on a  $[2] \times [2]$  grid are shown in Figure 12. Because  $S_1$  and  $T_1$  have cardinality  $\mathbf{R}(k)$ , we obtain subsets  $S_2 \subseteq S_1$  and  $T_2 \subseteq T_1$  of size  $k$  such that all the  $[2] \times [2]$  subgrids of  $S_2 \times T_2$  are colored by the same color. The only possible weak linear orders of a  $[2] \times [2]$  subgrid that can be present on all  $[2] \times [2]$  subgrids of a large grid are the first two pictures in the first row and the four first pictures in the second row of Figure 12. It follows that  $f$  behaves as one out of the six operations stated in the lemma.  $\square$

If  $f$  behaves on a grid  $G$  like one of the operations  $p_x, p_y, lex_{x,y}, lex_{y,x}, lex_{x,-y}$ , or  $lex_{y,-x}$ , we say that  $f$  is *homogeneous on  $G$*  (or simply *homogeneous* if  $G = \mathbb{Q}^2$ ).

## 6.4 Proof of the main result

In this subsection, we combine all the previous results to show that every temporal constraint language has a polynomial-time constraint satisfaction problem, or is NP-complete.

**Lemma 48.** *Let  $f$  be a binary operation violating *Betw* and preserving  $<$ . Then there are  $t_1, t_2 \in \text{Betw}$  such that  $f(t_1, t_2)$  has three distinct entries and  $f(t_1, t_2) \notin \text{Betw}$ .*

*Proof.* Since  $f$  violates *Betw*, there are two triples  $t_1, t_2 \in \text{Betw}$  such that  $t := f(t_1, t_2) \notin \text{Betw}$ . Because  $f$  preserves  $<$ , we can assume without loss of generality that  $t_1[1] < t_1[2] < t_1[3]$  and  $t_2[1] > t_2[2] > t_2[3]$ . If  $t$  has three distinct entries (in this case, we also say that  $t$  is injective), we are done. Otherwise we distinguish two cases:

1.  $t[1] = t[2] = t[3]$ : In that case, take a triple  $s_1$  such that  $s_1[1] < t_1[1]$ ,  $s_1[2] = t_1[2]$ , and  $s_1[3] = t_1[3]$ . We also choose a triple  $s_2$  such that  $t_2[2] < s_2[1] < t_2[1]$ ,  $s_2[2] = t_2[2]$ , and  $s_2[3] = t_2[3]$ . It is straightforward to check that  $s_1[1] < s_1[2] < s_1[3]$  and  $s_2[1] > s_2[2] > s_2[3]$  and thus both triples belong to *Betw*. Now, consider  $s := f(s_1, s_2)$ . We have that  $s[2] = t[2]$ ,  $s[3] = t[3]$ , and  $s[1] < t[1] = s[2] = s[3]$  because  $f$  preserves  $<$ . Therefore  $s \notin \text{Betw}$ . Take  $s_1$  instead of  $t_1$ ,  $s_2$  instead of  $t_2$  and proceed with case 2.
2. If exactly two entries in  $t$  have the same value, let  $i, j$  be their indices and let  $k$  be the index of the entry with the unique value. We assume that  $t[k] > t[i]$  (the other case is symmetric). It is straightforward to verify that there is an entry in  $t$  such that making the value of this entry smaller would make  $t$  injective and it would still not be in *Betw*. We can assume without loss of generality that  $i$  is an index of such an entry. We choose  $s_1$  so that  $s_1[i] < t_1[i]$ ,  $s_1[j] = t_1[j]$ ,  $s_1[k] = t_1[k]$ , and  $s_1[1] < s_1[2] < s_1[3]$ . We choose  $s_2$  such that  $s_2[i] < t_2[i]$ ,  $s_2[j] = t_2[j]$ ,  $s_2[k] = t_2[k]$ , and  $s_2[1] > s_2[2] > s_2[3]$ . Note that  $s_1, s_2 \in \text{Betw}$ . The tuple  $s := f(s_1, s_2)$  satisfies  $s[i] < t[i]$ ,  $s[j] = t[j]$ , and  $s[k] = t[k]$ . By the choice of  $i$  we conclude that  $s$  is injective,  $s \notin \text{Betw}$  and we are done.

□

A  $k$ -ary operation  $f : \mathbb{Q}^k \rightarrow \mathbb{Q}$  is *dominated* by the  $i$ -th argument when  $f(a_1, \dots, a_k) \leq f(b_1, \dots, b_k)$  if and only if  $a_i \leq b_i$ . Examples of operations dominated by the first argument are  $p_x$ ,  $\text{lex}_{x,y}$ , and  $\text{lex}_{x,-y}$ , and examples of operations dominated by the second argument are  $p_y$ ,  $\text{lex}_{y,x}$ ,  $\text{lex}_{y,-x}$ . We use the product Ramsey theorem (Theorem 46) to prove the following.

**Lemma 49.** *Let  $f$  be a binary operation that preserves  $<$  and violates *Betw*. Then  $f$  generates *ll*, *dual-ll*, *pp*, or *dual-pp*.*

*Proof.* If  $f$  violates *Betw* and preserves  $<$ , then Lemma 48 asserts that there are  $t_1, t_2 \in \text{Betw}$  such that  $t := f(t_1, t_2) \notin \text{Betw}$  and  $t$  is injective. As  $f$  preserves  $<$ , we can assume without loss of generality that  $t_1[1] < t_1[2] < t_1[3]$  and  $t_2[1] > t_2[2] > t_2[3]$  (otherwise, we apply the argument to  $f(y, x)$ ).

Either the triple  $t$  satisfies  $t[1] > t[2] < t[3]$  or  $t[1] < t[2] > t[3]$ . In the first case, let  $S_1 := \{x \in \mathbb{Q} \mid t_1[1] < x < t_1[2]\}$ ,  $S_2 := \{x \in \mathbb{Q} \mid t_1[3] < x\}$ ,  $T_1 := \{y \in \mathbb{Q} \mid t_2[3] < y < t_2[2]\}$ , and  $T_2 := \{y \in \mathbb{Q} \mid t_2[1] < y\}$ . In the second case, let  $S_1 := \{x \in \mathbb{Q} \mid t_1[2] < x < t_1[3]\}$ ,  $S_2 := \{x \in \mathbb{Q} \mid x < t_1[1]\}$ ,  $T_1 := \{y \in \mathbb{Q} \mid t_2[2] < y < t_2[1]\}$ , and  $T_2 := \{y \in \mathbb{Q} \mid y < t_2[3]\}$ . See Figure 14 for an illustration of these sets.

For each  $k \in \mathbb{N}$ , we define sets  $S_1^{(k)}, T_1^{(k)}, S_2^{(k)}, T_2^{(k)}$  as follows. We apply Lemma 47 to the grid  $S_1 \times T_1$  (both  $S_1$  and  $T_1$  are infinite and in particular larger than  $\mathbf{R}(\mathbf{R}(k))$ ), and obtain subsets  $U^{(k)} \subseteq S_1$  and  $V^{(k)} \subseteq T_1$  such that  $|U^{(k)}| \geq \mathbf{R}(k)$ ,  $|V^{(k)}| \geq \mathbf{R}(k)$ , and  $f$  is homogeneous on  $U^{(k)} \times V^{(k)}$ . Similarly, we apply Lemma 47 to the grid  $U^{(k)} \times T_2$  and obtain subsets  $S_1^{(k)} \subseteq U^{(k)}$  and  $T_2^{(k)} \subseteq T_2$  of cardinality at least  $k$  such that  $f$  is homogenous on  $S_1^{(k)} \times T_2^{(k)}$ . We finally apply Lemma 47 to the grid  $S_2 \times V^{(k)}$  and obtain subsets  $S_2^{(k)} \subseteq S_2$  and  $T_1^{(k)} \subseteq V^{(k)}$  of cardinality at least  $k$  such that  $f$  is homogeneous on  $S_2^{(k)} \times T_1^{(k)}$ . Note that  $f$  is in particular homogeneous on  $S_1^{(k)} \times T_1^{(k)}$ .

There are just  $6^3$  possibilities for how  $f$  behaves on those grids for given  $k$ . Hence, there is an infinite set  $K \subseteq \mathbb{N}$  such that  $f$  behaves in the same way on  $S_1^{(k)} \times T_1^{(k)}$  for all  $k \in K$ , in the same way on  $S_1^{(k)} \times T_2^{(k)}$  for all  $k \in K$ , and in the same way on  $S_2^{(k)} \times T_1^{(k)}$  for all  $k \in K$ .

The following observations will be obvious by inspection of Figure 14, left side. In case that  $S_1^{(k)}$  is before  $S_2^{(k)}$  (that is, all elements in  $S_1^{(k)}$  are smaller than all elements in  $S_2^{(k)}$ ) and  $T_1^{(k)}$  is

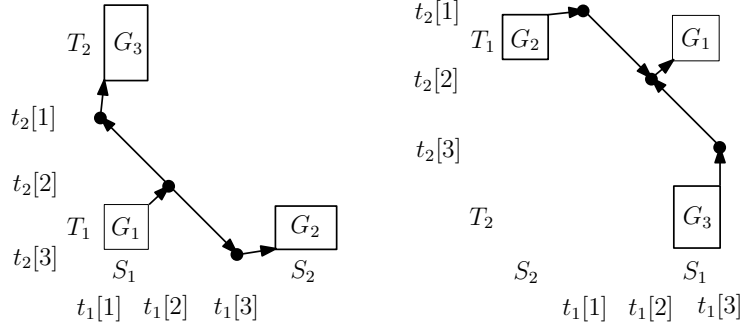


Figure 14: Grids chosen for the application of the product Ramsey theorem. The depicted ordering on the values of  $f$  follows from the choice of  $t_1, t_2$  and because  $f$  preserves  $<$ .

before  $T_2^{(k)}$ , then by the choice of  $S_1^{(k)}$ ,  $S_2^{(k)}$ ,  $T_1^{(k)}$ , and  $T_2^{(k)}$ , and because  $f$  preserves  $<$ , we have

$$f(x, y) < f(t_1[2], t_2[2]) < f(t_1[1], t_2[1]) < f(x', y')$$

for all  $(x, y) \in S_1^{(k)} \times T_1^{(k)}$  and  $(x', y') \in (S_1^{(k)} \times T_2^{(k)})$ . Similarly,

$$f(x, y) < f(t_1[2], t_2[2]) < f(t_1[3], t_2[3]) < f(x'', y'')$$

for all  $(x, y) \in S_1^{(k)} \times T_1^{(k)}$  and  $(x'', y'') \in (S_2^{(k)} \times T_1^{(k)})$ . The other case is that  $S_2^{(k)}$  is before  $S_1^{(k)}$  and  $T_2^{(k)}$  is before  $T_1^{(k)}$  (see the right side of Figure 14 for an illustration). In this case  $f(x, y) > f(t_1[2], t_2[2]) > f(x', y')$  for all  $(x, y) \in S_1^{(k)} \times T_1^{(k)}$  and  $(x', y') \in (S_1^{(k)} \times T_2^{(k)}) \cup (S_2^{(k)} \times T_1^{(k)})$ .

First suppose that  $f$  is dominated by the same argument on all the grids  $S_1^{(k)} \times T_1^{(k)}$ ,  $S_1^{(k)} \times T_2^{(k)}$ , and  $S_2^{(k)} \times T_1^{(k)}$  for all  $k \in K$ . We can assume that  $f$  is dominated on these grids by the second argument; otherwise we swap the arguments of  $f$ . Let  $g, h \in \{lex_{y,x}, lex_{y,-x}, p_y\}$  be such that  $f$  behaves like  $g$  on  $S_1^{(k)} \times T_1^{(k)}$  and like  $h$  on  $S_2^{(k)} \times T_1^{(k)}$ . Then by the above observations and local interpolation  $f$  generates  $[g|h]$  if  $S_1$  is before  $S_2$ , and  $[h|g]$  if  $S_2$  is before  $S_1$ . Moreover, we show that  $f$  also generates  $lex$ .

- If  $g$  or  $h$  is  $lex_{x,y}$  or  $lex_{y,x}$ , then  $f$  clearly generates  $lex$ .
- If  $g$  or  $h$  is  $lex_{x,-y}$  or  $lex_{y,-x}$ , then  $f$  generates  $lex$  as well, because  $lex(x, -lex(x, -y))$  behaves like  $lex(x, y)$ .
- If  $g$  is  $p_y$  and  $h$  is  $p_y$ , then  $f$  generates  $lex$  by Lemma 44.

Note that the operation  $[g|h]$  satisfies the conditions in Lemma 43, and hence  $\{lex, [g|h]\}$  generates  $[lex_{y,x}|lex_{y,x}]$ . By Lemma 44,  $f$  generates  $ll$ .

Now we consider the case that  $f$  is dominated by different arguments on the grids  $S_1^{(k)} \times T_1^{(k)}$  and  $S_1^{(k)} \times T_2^{(k)}$ , or by different arguments on the grids  $S_1^{(k)} \times T_1^{(k)}$  and  $S_2^{(k)} \times T_1^{(k)}$ , for all  $k \in K$ . We only consider the first case; the second case is symmetric under swapping the arguments of  $f$ . Let  $g, h$  be from  $\{lex_{x,y}, lex_{x,-y}, lex_{y,x}, lex_{y,-x}, p_x, p_y\}$  such that  $f$  behaves like  $h$  on the grids  $S_1^{(k)} \times T_1^{(k)}$  and like  $g$  on the grids  $S_2^{(k)} \times T_1^{(k)}$ . Again, by local interpolation  $f$  generates  $[h|g]$  if  $S_1$  is before  $S_2$ , and  $[g|h]$  if  $S_2$  is before  $S_1$ . We assume without loss of generality that  $f$  generates  $[h|g]$  (in the other case we can exchange the names of  $h$  and  $g$  and proceed in the same way).

If  $h$  is  $p_y$  and  $g$  is  $p_x$ , then  $[h|g]$  behaves like  $pp$ ; hence  $f$  generates  $pp$  and we are done. Dually, if  $h$  is  $p_x$  and  $g$  is  $p_y$ , then  $f$  generates  $dual\text{-}pp$ . In all other cases, either  $h$  or  $g$  is from  $lex_{x,y}, lex_{y,x}, lex_{x,-y}$ , or  $lex_{y,-x}$ , and thus  $f$  generates  $lex$  as we have already seen before. But then Lemma 43 shows that  $f$  generates  $ll$  or  $dual\text{-}ll$ .  $\square$

We are now ready to prove our classification result.

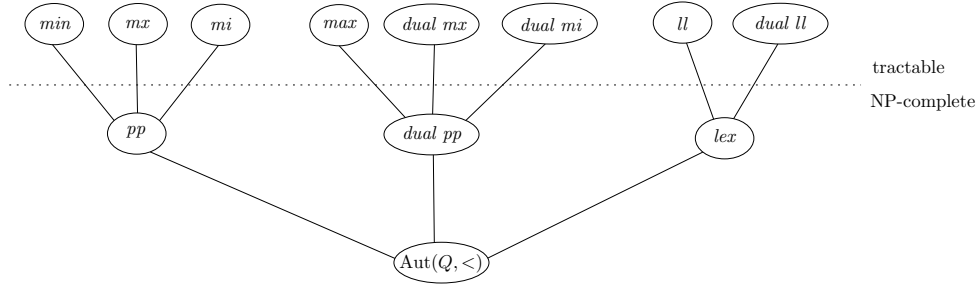


Figure 15: An illustration of the classification result for constraint languages that contain  $<$ .

**Theorem 50.** *A temporal constraint language  $\Gamma$  has a tractable CSP if  $\Gamma$  is preserved by at least one of the following nine operations:  $ll, min, mi, mx$ , their duals, or a constant operation. Otherwise,  $CSP(\Gamma)$  is NP-complete.*

*Proof.* If  $\Gamma$  is preserved by a constant operation, then assigning the same value to every variable of an instance of  $CSP(\Gamma)$  is a solution to this instance, unless there is a constraint for false in the instance, in which case we simply reject. Therefore,  $CSP(\Gamma)$  can trivially be solved in polynomial time. In case that  $\Gamma$  is preserved by  $ll$  or  $dual-ll$ , there is a quadratic-time algorithm that solves  $CSP(\Gamma)$ , see [5]. If  $\Gamma$  is preserved by  $min, mi, mx$  or one of their duals, tractability is shown in Section 5.

Theorem 20 asserts that one of the following cases is true:

1.  $CSP(\Gamma)$  is NP-complete, because there is a primitive positive definition of some relation with an NP-complete CSP,
2.  $Pol(\Gamma)$  contains a constant operation. In this case  $CSP(\Gamma)$  is tractable as we have argued above),
3.  $Pol(\Gamma)$  contains all permutations of  $\mathbb{Q}$ , or
4. there is some binary  $f \in Pol(\Gamma)$  that preserves  $<$  and violates *Betw*.

In the third case,  $\Gamma$  is an equality constraint language, and the statement follows easily from Theorem 13 and Theorem 15 in [4]. In the fourth case, Lemma 49 implies that the operation  $f$  generates  $pp, dual-pp, ll$ , or  $dual-ll$ . If  $f$  generates  $ll$  or  $dual-ll$  we are in one of the described tractable cases. If  $f$  generates  $pp$  then Lemma 36 shows that either  $f$  preserves  $S$  and thus  $CSP(\Gamma)$  is NP-hard by Proposition 15, or  $\Gamma$  is preserved by  $min, mi$ , or  $mx$ . Dually, if  $f$  generates  $dual-pp$  then either  $f$  preserves  $-S$  and  $CSP(\Gamma)$  is NP-hard, or  $\Gamma$  is preserved by one of the duals of  $min, mi$ , or  $mx$ , which completes the proof.  $\square$

By inspection of all the temporal relations that were used to show hardness, we can also describe the main result relationally as follows.

**Corollary 51.** *If there is a primitive positive definition of *Betw, Cycl, Sep, S, -S*, or  $\{(x, y, z) \in \mathbb{Q}^3 \mid x = y \neq z \vee x \neq y = z\}$  in  $\Gamma$ , then  $CSP(\Gamma)$  is NP-complete. Otherwise,  $CSP(\Gamma)$  is tractable.*

*Proof.* The result follows from the proof of the previous theorem and the theorems referenced therein, and the observation that temporal languages that are preserved by all permutations either have a primitive positive definition of  $\{(x, y, z) \in \mathbb{Q}^3 \mid x = y \neq z \vee x \neq y = z\}$ , or are tractable (this follows easily from Theorem 13 and Theorem 15 in [4]).  $\square$

## 7 Concluding remarks

We have completely classified the complexity of the constraint satisfaction problem for temporal constraint languages. By Theorem 5, temporal constraint languages are precisely the highly set-transitive structures; hence, we have obtained a complexity classification for the CSP of those templates that have in a certain sense that largest possible degree of symmetry.

See Figure 16 for an overview over the nine largest tractable temporal constraint languages; the entries also mention *typical relations* for the respective language, i.e., a set of relations that is contained in the language, but not contained in any other of the nine languages – hence, these relations show that all the languages are distinct.

Polymorphism	Typical Relations	Complexity	Reference
min	$\{U, <\}$	$O(n^2m)$	Theorem 39
mi	$\{I\}$	$O(n^3m)$	Theorem 41
mx	$\{X\}$	$O(n^4)$	Theorem 42
max = dual min	$\{-U, <\}$	$O(n^2m)$	
dual mi	$\{-I\}$	$O(n^3m)$	
dual mx	$\{-X\}$	$O(n^4)$	
ll	$\{(u \neq v) \vee x > y \vee x > z\}$	$O(nm)$	See [5]
dual ll	$\{(u \neq v) \vee x < y \vee x < z\}$	$O(nm)$	
constant	$\{(x \leq y \leq z) \vee (z \leq y \leq x)\}$	$O(m)$	

Figure 16: Summary of the various tractable languages. For the last three operations, the typical relations are given by their first-order definition; in all other cases, see Section 4.

Finally, we would like to remark that it follows from the descriptions of the constraint languages via the polymorphisms given in the paper that the so-called *meta-problem* for tractability is decidable; this is formally stated in the following corollary.

**Corollary 52.** *Let  $(\mathbb{Q}; R_1, \dots, R_n)$  be a finite temporal constraint language and let  $\phi_1, \dots, \phi_n$  be quantifier-free first-order formulas that define  $R_1, \dots, R_n$  over  $(\mathbb{Q}; <)$ , respectively. There is an algorithm that, given  $\phi_1, \dots, \phi_n$ , decides whether  $\Gamma$  is tractable or has an NP-complete CSP.*

*Proof.* Theorem 50 shows that it suffices to test for each of the nine (at most binary) polymorphisms whether all relations  $R_1, \dots, R_n$  are preserved by this polymorphism. Suppose that  $R_i$  is  $k$ -ary. Recall the observation from Proposition 6 that whether or not a  $k$ -tuple  $t \in \mathbb{Q}^k$  is in  $R_i$  only depends on the weak linear order  $\preceq$  of the entries of  $t$ . Now observe that for  $f \in \{\text{min}, \text{mi}, \text{mx}, \text{max}, \text{dual-mi}, \text{dual-mx}\}$  the weak linear order of the  $k$ -tuple  $f(t_1, t_2)$  is determined by the weak linear order of the  $2k$ -tuple  $(t_1, t_2)$ . Hence, to test whether  $R_i$  is preserved by  $f$  it suffices to verify for a finite number of weak linear orders whether they satisfy  $\phi_i$ . A procedure that decides whether  $R_i$  is preserved by  $ll$  or  $dual-ll$  has been described in [5]. Testing whether  $R_i$  is preserved by a constant polymorphism is trivial, we only have to verify whether the  $k$ -tuple  $(0, \dots, 0)$  satisfies  $\phi_i$ .  $\square$

**Acknowledgements.** We would like to thank the referees for their many very helpful comments.

## References

- [1] M. Bodirsky. Cores of countably categorical structures. *Logical Methods in Computer Science (LMCS)*, 2007. DOI: 10.2168/LMCS-3(1:2).
- [2] M. Bodirsky, H. Chen, J. Kara, and T. von Oertzen. Maximal infinite-valued constraint languages. *Theoretical Computer Science (TCS)*. A preliminary version appeared at *ICALP'07*, 410:1684–1693, 2009.

- [3] M. Bodirsky and V. Dalmau. Datalog and constraint satisfaction with infinite templates. *An extended abstract appeared in the proceedings of STACS'06. The full version is available online at arXiv:0809.2386 [cs.LO]*, 2008.
- [4] M. Bodirsky and J. Kára. The complexity of equality constraint languages. *Theory of Computing Systems*, 3(2):136–158, 2008. A conference version appeared in the proceedings of CSR'06.
- [5] M. Bodirsky and J. Kára. A fast algorithm and Datalog inexpressibility for temporal reasoning. *Preprint, CoRR abs/0805.1473. Accepted for publication in the ACM Transactions of Computing Systems*, 2008.
- [6] M. Bodirsky and J. Nešetřil. Constraint satisfaction with countable homogeneous templates. *Journal of Logic and Computation*, 16(3):359–373, 2006.
- [7] M. Bodirsky and M. Pinsker. All reducts of the random graph are model-complete. Preprint, arXiv:0810.2270.
- [8] V. G. Bodnarčuk, L. A. Kalužnin, V. N. Kotov, and B. A. Romov. Galois theory for post algebras, part I and II. *Cybernetics*, 5:243–539, 1969.
- [9] M. Broxvall and P. Jonsson. Point algebras for temporal reasoning: Algorithms and complexity. *Artif. Intell.*, 149(2):179–220, 2003.
- [10] A. Bulatov. Tractable conservative constraint satisfaction problems. In *Proceedings of LICS'03*, pages 321–330, 2003.
- [11] A. Bulatov. A graph of a relational structure and constraint satisfaction problems. In *Proceedings of LICS'04, Turku, Finland*, 2004.
- [12] A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006.
- [13] 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 Series II: Mathematics, Physics, Chemistry, 207:181–213, 2005.
- [14] A. Bulatov, A. Krokhin, and P. G. Jeavons. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34:720–742, 2005.
- [15] P. J. Cameron. Transitivity of permutation groups on unordered sets. *Math. Z.*, 148:127–139, 1976.
- [16] P. J. Cameron. *Oligomorphic Permutation Groups*. Cambridge Univ. Press, 1990.
- [17] G. Cantor. Über unendliche, lineare Punktmannigfaltigkeiten. *Mathematische Annalen*, 23:453–488, 1884.
- [18] D. A. Cohen, P. Jeavons, P. Jonsson, and M. Koubarakis. Building tractable disjunctive constraints. *J. ACM*, 47(5):826–853, 2000.
- [19] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*. SIAM Monographs on Discrete Mathematics and Applications 7, 2001.
- [20] V. Dalmau and J. Pearson. Closure functions and width 1 problems. *Proceedings of CP'99*, pages 159–173, 1999.
- [21] T. Drakengren and P. Jonsson. Twenty-one large tractable subclasses of Allen's algebra. *Artificial Intelligence*, 93:297–319, 1997.
- [22] T. Drakengren and P. Jonsson. Reasoning about set constraints applied to tractable inference in intuitionistic logic. *J. Log. Comput.*, 8(6):855–875, 1998.
- [23] 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.
- [24] M. Garey and D. Johnson. *A guide to NP-completeness*. CSLI Press, 1978.
- [25] D. Geiger. Closed systems of functions and predicates. *Pacific Journal of Mathematics*, 27:95–100, 1968.
- [26] A. Goerdts. On random ordering constraints. In *Proceedings of Computer Science Russia, LNCS*, 2009. To appear.

- [27] W. Guttman and M. Maucher. Variations on an ordering theme with constraints. In *Fourth IFIP International Conference on Theoretical Computer Science (TCS'06)*, IFIP International Federation for Information Processing 209, pages 77–90, 2006.
- [28] W. Hodges. *A shorter model theory*. Cambridge University Press, 1997.
- [29] P. M. Idziak, P. Markovic, R. McKenzie, M. Valeriote, and R. Willard. Tractability and learnability arising from algebras with few subpowers. In *Proceedings of LICS'07*, pages 213–224, 2007.
- [30] P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *JACM*, 44(4):527–548, 1997.
- [31] P. G. Jeavons and M. C. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2):327–339, 1995.
- [32] P. Jonsson and T. Drakengren. A complete classification of tractability in RCC-5. *J. Artif. Intell. Res.*, 6:211–221, 1997.
- [33] M. Junker and M. Ziegler. The 116 reducts of  $(\mathbb{Q}, <, a)$ . *Journal of Symbolic Logic*, 74(3):861–884, 2008.
- [34] A. Krokhin, P. Jeavons, and P. Jonsson. Reasoning about temporal relations: The tractable subalgebras of Allen’s interval algebra. *JACM*, 50(5):591–640, 2003.
- [35] D. Marker. *Model Theory: An Introduction*. Springer, 2002.
- [36] R. H. Möhring, M. Skutella, and F. Stork. Scheduling with and/or precedence constraints. *SIAM J. Comput.*, 33(2):393–415, 2004.
- [37] B. Nebel and H.-J. Bürckert. Reasoning about temporal relations: A maximal tractable subclass of Allen’s interval algebra. *JACM*, 42(1):43–66, 1995.
- [38] 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.
- [39] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of STOC'78*, pages 216–226, 1978.
- [40] A. Szendrei. *Clones in universal Algebra*. Séminaire de Mathématiques Supérieures. Les Presses de L’Université de Montréal, 1986.
- [41] M. Vilain, H. Kautz, and P. van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. *Reading in Qualitative Reasoning about Physical Systems*, pages 373–381, 1989.