

A Fast Algorithm and Datalog Inexpressibility for Temporal Reasoning

Manuel Bodirsky

Laboratoire d'informatique (LIX, CNRS UMR 6171),

École Polytechnique, Paris-Palaiseau, France

bodirsky@lix.polytechnique.fr

and

Jan Kára

Charles University, Prague, Czech Republic

kara@kam.mff.cuni.cz

We introduce a new tractable temporal constraint language, which strictly contains the Ord-Horn language of Bürkert and Nebel and the class of AND/OR precedence constraints. The algorithm we present for this language decides whether a given set of constraints is consistent in time that is quadratic in the input size. We also prove that (unlike Ord-Horn) the constraint satisfaction problem of this language cannot be solved by Datalog or by establishing local consistency.

Categories and Subject Descriptors: F.2.2 [**Theory of Computation**]: ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY—*Nonnumerical Algorithms and Problems*

Additional Key Words and Phrases: constraint satisfaction, temporal reasoning, computational complexity, Ord-Horn, algorithms, Datalog, precedence constraints

1. INTRODUCTION

Temporal reasoning plays an important role in Artificial Intelligence. Almost any area in AI – for instance common-sense reasoning, natural language processing, scheduling, planning – involves some sort of temporal reasoning. In 1993, Golumbic and Shamir [GS33] listed applications of temporal reasoning problems in archeology, behavioral psychology, operations research, and circuit design. Since then, temporal reasoning became one of the benchmark applications of constraint processing in general [Dec03]. Contributions to the field have various background, for example database theory [vdM97], scheduling [MSS04], constraint satisfaction complexity [BJK05], the theory of relation algebras [LM94, Due05], combinatorics [GS33], and artificial intelligence [FGV05].

This paper deals with *temporal constraint languages*. A *temporal constraint lan-*

Jan Kára was supported by Project 1M0021620808 of the Ministry of Education of the Czech Republic.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 2009 ACM 1529-3785/2009/0700-0001 \$5.00

guage (TCL) is a countable collection of relations with a first-order definition in $(\mathbb{Q}, <)$, the linear order of the rational numbers; a detailed definition is given in Section 2. One of the most fundamental TCLs is the so-called *point algebra*. This language contains relations for $=, <, \leq,$ and \neq , interpreted over an infinite dense linear order in the usual way. Vilain, Kautz, and van Beek showed that consistency of a given set of constraints over this language (aka the *constraint satisfaction problem* for this language) can be decided in polynomial time by local consistency techniques [VKvB89]. Later, van Beek described an algorithm that runs in $O(n^2)$, where n is the number of variables [vB92].

A considerably larger tractable TCL was introduced by Bürkert and Nebel [NB95]. Their language, called *Ord-Horn*, strictly contains the point algebra. Bürkert and Nebel used resolution to show that consistency of a set of Ord-Horn constraints can be decided in $O(s^3)$, where s is the size of the input. They also showed that establishing path-consistency can be used to decide whether a given set of Ord-Horn constraints has a solution. Koubarakis [Kou01] later presented an algorithm with a running time in $O(s^2)$.

Ord-Horn is motivated by temporal reasoning tasks for constraints on *time intervals*. The study of constraints on intervals (which can be used to model temporal information about events) was initiated by Allen [All83], who introduced an algebra of binary constraint relations on intervals. The complexity to decide the consistency of a given set of constraints from Allen’s algebra is NP-complete in general [All83]. However, for several fragments of Allen’s interval algebra the consistency problem is decidable in polynomial time. All such fragments have been classified [DJ97, KJJ03]. It is well-known that every relation on *intervals* from Allen’s algebra can be translated to a relation on *time points*. Hence, algorithmic results for temporal reasoning with time points can be used for reasoning with time intervals as well. Bürkert and Nebel used this translation to identify one of the tractable fragments of Allen’s interval algebra, namely the set of all interval constraints that translate to Ord-Horn constraints on points.

Another important temporal constraint language with applications in scheduling are AND/OR-precedence constraints [MSS04]. An AND-constraint can be used to express that some job cannot be started before a set of other jobs has been completed. An OR-constraint can be used to express that a job cannot be started before *one* of a given set of jobs has been completed. Feasibility of AND/OR precedence constraints can indeed be modeled as a constraint satisfaction problem for a TCL: AND-constraints can be represented by conjunctions of formulas of the form $x > y$, and OR-constraints by formulas of the form $x > x_1 \vee \dots \vee x > x_n$.

There are temporal constraint languages where one cannot expect a polynomial time algorithm. A well-known TCL with an NP-complete consistency problem consists of a single ternary relation, the *betweenness relation* $\{(x, y, z) \mid x < y < z \vee z < y < x\}$; another example of such an NP-complete language consists of the *cyclic ordering relation*, which is the ternary relation $\{(x, y, z) \mid x < y < z \vee y < z < x \vee z < x < y\}$. The constraint satisfaction problems for these two languages are listed as NP-complete in the book of Garey and Johnson [GJ78]. We want to remark that the complexity of temporal constraint satisfaction problems for a *fixed and finite* number of time points was completed recently [CHKS05]; however, the restriction

to a finite number of time points changes the nature of the problem considerably.

We present a new tractable TCL that strictly contains all Ord-Horn relations and all AND-OR precedence constraints (and also contains relations that are neither Ord-Horn nor AND-OR precedence constraints). Our language is defined by a universal-algebraic closure property, and we call it the class of *ll-closed* relations. We show in Section 3 that a relation is *ll-closed* if and only if it can be defined by a formula of the form

$$(x_1 = y_1 \wedge \cdots \wedge x_k = y_k) \rightarrow (z_0 > z_1 \vee \cdots \vee z_0 > z_l), \text{ or} \\ (x_1 = y_1 \wedge \cdots \wedge x_k = y_k) \rightarrow (z_0 > z_1 \vee \cdots \vee z_0 > z_l \vee (z_0 = z_1 = \cdots = z_l))$$

(where k and l might be 0). It has been shown in [BK08] that *ll-closed* constraints are a *largest tractable language* in the sense that every TCL that strictly contains one of our two languages has an NP-complete constraint satisfaction problem. The presented algorithm for *ll-closed* constraints has a running time that is quadratic in the size of its input.

Traditionally, one of the main algorithmic tools in constraint satisfaction, and in particular in temporal reasoning, are *local consistency techniques* [All83, VKvB89, GS33, DJ97, NB95], for instance algorithms based on establishing path-consistency. Consistency based algorithms can be formulated conveniently as Datalog programs [BD06, FV99, KV98]. Roughly speaking, Datalog is Prolog without function symbols, and comes from Database theory [EF99]. We show that, unlike Ord-Horn [NB95], *ll-closed* and dual *ll-closed* constraints can *not* be solved by a Datalog program. In our proof we apply a pebble-game argument that was originally introduced for finite domains [FV99, KV98], but has been shown to generalize to a wide range of infinite domain constraint languages, including TCLs [BD06]. This is interesting from a theoretical point of view: for constraint satisfaction problems of languages over a finite domain, all known algorithms are essentially based on algebraic algorithms or Datalog [FV99]. However, the algorithm we present for temporal reasoning is neither algebraic nor based on Datalog.

2. TEMPORAL CONSTRAINT LANGUAGES

A (*qualitative*) *temporal relation* is a relation that is first-order definable in an unbounded countable dense strict linear order. All such linear orders are isomorphic [Hod97, Mar02], but for convenience we always use $(\mathbb{Q}, <)$, i.e., the dense linear order on the rational numbers¹. An example of a temporal relation is the ternary *Betweenness* relation $\{(x, y, z) \in \mathbb{Q}^3 \mid (x < y \wedge y < z) \vee (z < y \wedge y < x)\}$ mentioned in the introduction. It is well-known that every temporal relation also has a *quantifier-free* definition [Hod97, Mar02], i.e., we can define every temporal relation with a formula that is a Boolean combination of literals of the form $x < y$ (as above in the case of the *Betweenness* relation).

A *temporal constraint language (TCL)* is an (at most countable) set of relation symbols R_1, R_2, \dots , where each relation symbol R_i is associated with an arity $k_i \geq 2$, and is interpreted by a k_i -ary temporal relation. For simplicity, we use the

¹One could also consider dense linear orders on arbitrary infinite base sets, e.g. $(\mathbb{R}, <)$; but it is easy to see all the results in this paper also apply to this case.

same symbol for the relation symbol and the corresponding temporal relation. As an example, consider the set of binary relation symbols $\Gamma_0 := \{\neq, \leq, <, =\}$, with the obvious interpretation over $(\mathbb{Q}, <)$.

The *constraint satisfaction problem* of a temporal constraint language Γ is the following computational problem.

CSP(Γ)

INSTANCE: A first-order formula Φ of the form $\phi_1 \wedge \dots \wedge \phi_p$, where each ϕ_i is an atomic formula with variables from x_1, \dots, x_n and a relation symbol from Γ .

QUESTION: Is there an assignment of rational numbers to x_1, \dots, x_n such that Φ is satisfied?

The atomic formulas ϕ_1, \dots, ϕ_p are called the *constraints* of the instance Φ of CSP(Γ). For a constraint $\phi = R(x_1, \dots, x_k)$ we say that ϕ has *arity* $ar(\phi) = k$ and for x_i from $\{x_1, \dots, x_k\}$ we say that ϕ is *imposed* on x_i . A tuple $(a_1, \dots, a_n) \in \mathbb{Q}^n$ is called a *solution* for Φ if the assignment $x_i := a_i$ satisfies all formulas in Φ . If there is no solution for Φ , then we say that Φ is *unsatisfiable*, and *satisfiable* (or *consistent*) otherwise. Thus, CSP(Γ) is the problem to decide whether a given set of constraints over relations from Γ is satisfiable.

Example. Let $R(x, y, u, v)$ be the 4-ary temporal relation defined by $(x=y \wedge y < u \wedge u \wedge u=v) \vee (x < y \wedge y < u \wedge u < v)$. Consider the instance

$$\Phi_1 := \{R(x_1, x_2, y_1, y_2), R(x_1, x_2, y_2, y_3), R(x_1, x_2, y_3, y_1)\}$$

of CSP($\{R\}$). It is easy to see that the sentence $\exists x_1, x_2, y_1, y_2, y_3. \bigwedge_{\phi \in \Phi_1} \phi$ is true, and a solution to Φ_1 is $(0, 0, 1, 1, 1)$.

A *finite* constraint language Γ is called *tractable* if CSP(Γ) can be solved in polynomial time (note that because Γ is finite this concept is independent from the representation of the relation symbols in the input). The constraint language Γ_0 mentioned at the beginning of this section, for example, corresponds to the well-studied point-algebra that we mentioned in the introduction, and is tractable. An *infinite* constraint language Γ is called *locally tractable* if every finite subset of the constraint language is tractable. The algorithmic results presented in this paper show more than local tractability of constraint languages. To formulate our results, we have to discuss how to represent temporal relations in instances of CSP(Γ) when Γ is infinite.

It is straightforward to verify that whether or not an n -tuple t is in a temporal relation only depends on the weak linear order $tp(t)$ defined on $\{1, \dots, n\}$ by $(i, j) \in tp(t)$ iff $t[i] \leq t[j]$. We also say that t *satisfies* $tp(t)$ ². This observation leads to a natural way to represent temporal relations. If R is a k -ary temporal relation, R can be represented by a set \mathcal{R} of weak linear orders on $\{1, \dots, k\}$ as follows. For every k -tuple $t \in R$, the weak linear order $tp(t)$ is contained in \mathcal{R} . Conversely, for every weak linear order w in \mathcal{R} there is a k -tuple $t \in R$ such that $w = tp(t)$. For example, the relation R in the example above can be characterized as the set of all

²The notation tp is motivated by the concept of (*complete*) *types* in model theory [Hod97, Mar02].

tuples that satisfy either $tp((0, 0, 1, 1))$ or $tp((0, 1, 2, 3))$.

If Γ is the set of all temporal relations, then $\text{CSP}(\Gamma)$ is well-known to be NP-complete (here we assume that temporal relations are represented by sets of weak linear orders). For containment in NP, note that one can verify in polynomial time whether a given weak linear order on n variables corresponds to a solution for a given instance Φ with n variables. We can therefore decide non-deterministically in polynomial time whether there exists a weak linear order on n elements and an (arbitrary) n -tuple t satisfying this weak linear order such that t is a solution to Φ . For NP-hardness, recall that already the constraint language that only contains a single relation symbol for the Betweenness relation is NP-complete [GJ78].

For a fixed way how to represent the relations from a constraint language Γ (such as the representation of temporal relations by sets of weak linear orders as discussed above), we say that Γ is (*globally*) *tractable* if $\text{CSP}(\Gamma)$ can be solved in polynomial time. The representation of a temporal relation by a set of weak linear orders corresponds to the standard representation of a relation over a finite domain by its set of tuples³. For finite domains, it is an open problem whether the notion of local tractability and the notion of global tractability with respect to the standard representation coincide (and in fact it has been conjectured that they do [BKJ05]).

Another natural way to represent a temporal relation is by specifying a formula that defines the relation. However, general Boolean combinations of literals of the form $x < y$ are obviously too expressive if we are interested in efficient algorithms (it is NP-hard to decide whether such a formula represents a non-empty relation), so we have to restrict the set of all formulas appropriately; such a syntactic restriction will be presented in Section 3.

In this paper we present an algorithm that shows that a large temporal constraint language is globally tractable, both with respect to the representation of constraint relations by sets of weak linear orders and with respect to the representation of temporal relations by formulas. Even though this requires that we have to go into more detail as compared to a local tractability result, we would like to present the stronger global tractability result in this paper, because this allows us to relate our algorithm with previous algorithms in temporal reasoning, for example for Ord-Horn (which is an infinite temporal constraint language).

3. LL-CLOSED LANGUAGES

We first introduce fundamental concepts from model theory and universal algebra; they are standard, see e.g. [Hod97, Sze86]. We say that a k -ary function (also called *operation*) $f : \mathbb{Q}^k \rightarrow \mathbb{Q}$ *preserves* an m -ary relation $R \subseteq \mathbb{Q}^m$ if whenever $R(a_1^i, \dots, a_m^i)$ holds for all $1 \leq i \leq k$, then $R(f(a_1^1, \dots, a_1^k), \dots, f(a_m^1, \dots, a_m^k))$ holds as well. If f preserves all relations of a TCL Γ , we say that f is a *polymorphism* of Γ . Unary bijective polymorphisms are called the *automorphisms* of Γ ; the set of all automorphisms of Γ is denoted by $\text{Aut}(\Gamma)$.

Let lex be a binary operation on \mathbb{Q} such that $\text{lex}(a, b) < \text{lex}(a', b')$ if either $a < a'$, or $a = a'$ and $b < b'$. It is easy to see that the set of temporal relations preserved by lex is not affected by the choice of the binary operation lex if lex has the properties

³Also for finite domains, a representation by formulas might sometimes be more natural, for example for boolean Horn satisfiability.

above. Thus for all the arguments in this paper, it does not matter which operation lex we are choosing. Also note that every such operation is by definition injective.

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

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

See Figure 1 for illustration. In diagrams like in Figure 1 we draw a directed edge from (a, b) to (a', b') if $\text{ll}(a, b) < \text{ll}(a', b')$. Again, it is easy to see that the set of temporal relations preserved by ll is not affected by the exact choice of the binary operation ll . Also observe that every temporal relation that is preserved by ll is also preserved by lex . We say that a relation is ll -closed if it is preserved by ll .

It is possible to decide algorithmically whether a constraint language is ll -closed.

PROPOSITION 3.1. *Given a finite constraint language where all relations are represented as lists of weak linear orders, one can decide in polynomial time in the input size whether the constraint language is ll -closed.*

PROOF. We test for each relation R in the constraint language separately whether it is ll -closed. A k -ary relation R is preserved by ll if and only if for every two weak orders o_1 and o_2 in R and every index $e \leq k$ the weak order o_3 is also in R , where o_3 is defined as follows: $(i, j) \in o_3$ iff one of the following holds

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

For all pairs (o_1, o_2) of weak linear orders on $\{1, \dots, k\}$ in the representation of R , and for each index $e \leq k$, we can verify in linear time in k whether the weak linear order o_3 as described above is also contained in the representation of R . \square

Similarly to the ll operation we can define a dual ll operation, as depicted in Figure 1. To show that the language of all ll -closed relations is different from the language of all dual ll -closed relations, we use the following two relations, which will be also of importance in later arguments.

DEFINITION 3.2. *We define R^{\min} to be the ternary relation $\{(x, y, z) \mid x > y \vee x > z\}$, and R^{\max} to be $\{(x, y, z) \mid x < y \vee x < z\}$.*

Observe that $R^{\min}(x, y, z)$ holds if and only if x is larger than the minimum of y and z . Similarly, $R^{\max}(x, y, z)$ holds if and only if x is smaller than the maximum of y and z . It was shown in [MSS04] and independently in [GM06] that $\text{CSP}(\mathbb{Q}, R^{\max})$ can be solved in polynomial time. For the proof of the next lemma we prove that the relation R^{\min} is ll -closed; the proof can be adapted easily to show that all k -ary relations defined by formulas of the form $x_1 > x_2 \vee \dots \vee x_1 > x_k$ are ll -closed as well, which are the relations to model AND/OR precedence constraints.

PROPOSITION 3.3. *The language of ll -closed relations does not contain the class of dual ll -closed relations and vice versa.*

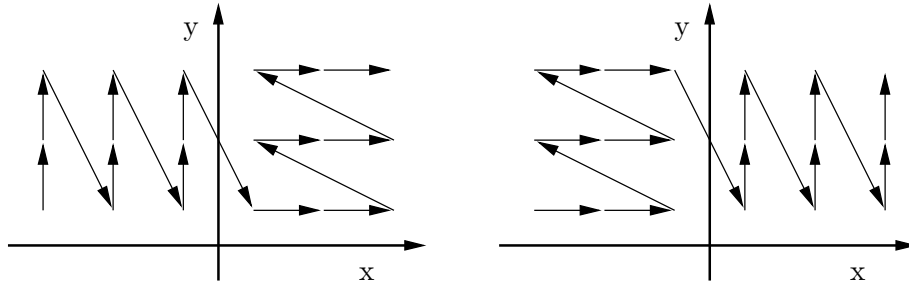


Fig. 1. A visualization of the ll (left side) and the dual ll operation (right side).

PROOF. To show that the language of ll-closed constraints does not contain the language of dual ll-closed constraints, we show that there is a temporal relation that is preserved by ll but not by dual ll. We claim that the relation R^{min} is preserved by the ll operation: Let (x_1, x_2, x_3) and (y_1, y_2, y_3) be triples that are both in the relation R^{min} . Without loss of generality, $x_1 > x_2$ (note that the relation is symmetric in the second and third argument). If in this case $y_1 \geq y_2$, then, because ll preserves \leq , we have that $ll(x_1, y_1) \geq ll(x_2, y_2)$, and because ll is injective, we have that $ll(x_1, y_1) > ll(x_2, y_2)$. Therefore $(ll(x_1, y_1), ll(x_2, y_2), ll(x_3, y_3))$ is in R^{min} , and we are done. So let us assume that $y_1 < y_2$ and therefore $y_1 > y_3$. We can again apply the previous argument to show that $(ll(x_1, y_1), ll(x_2, y_2), ll(x_3, y_3))$ is in R^{min} unless $x_1 < x_3$. So let us assume that $x_1 < x_3$. Now, in case that $x_2 > 0$, the operation ll preserves R^{min} , since in this case ll acts like a lexicographic order on the two triples. Otherwise, $x_2 \leq 0$. It is easy to check that then $ll(x_2, y_2) < ll(x_1, y_1)$ because $x_1 > x_2$.

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

This shows that the language of ll-closed constraints does not contain the language of dual ll-closed constraints. Analogously, we can use the relation R^{max} to show that the language of dual ll-closed constraints does not contain the language of ll-closed constraints. \square

The temporal constraint language of all ll-closed relations also contains the important class of Ord-Horn relations, introduced by Bürckert and Nebel [NB95] to identify a tractable class of interval constraints. A relation is Ord-Horn if it can be defined by a conjunction of formulas of the form

$$(x_1 = y_1 \wedge \dots \wedge x_k = y_k) \rightarrow x_0 O y_0,$$

where $O \in \{=, <, \leq, \neq\}$. It is always possible to translate interval constraints into temporal constraints [VKvB89]. If the translation of an interval constraint language falls into a tractable TCL, the interval constraint language is tractable as well. Bürckert and Nebel showed that the class of *interval* constraints having a translation into Ord-Horn temporal constraints is a largest tractable fragment of Allen's interval algebra. Note that this does not imply that the class of Ord-Horn

constraints is a largest tractable TCL on time points. Indeed, this is not the case. Proposition 3.4 below shows that the class of Ord-Horn constraints is ll-closed. Since the relation R^{min} defined in this section is ll-closed but not Ord-Horn, the class of ll-closed constraints is *strictly* larger than Ord-Horn. Finally, we prove in Section 4 that ll-closed constraints are tractable.

PROPOSITION 3.4. *All relations in Ord-Horn are preserved by ll and by dual ll.*

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

It turns out that a temporal relation is preserved by ll if and only if it can be defined by a class of formulas which we call *ll-Horn formulas*. This class properly extends the class of Ord-Horn formulas. A formula is called *ll-Horn* if it is a conjunction of formulas of the following form (slightly abusing terminology, we call these formulas the *clauses* of the ll-Horn formula)

$$(x_1 = y_1 \wedge \cdots \wedge x_k = y_k) \rightarrow (z_0 > z_1 \vee \cdots \vee z_0 > z_l), \text{ or}$$

$$(x_1 = y_1 \wedge \cdots \wedge x_k = y_k) \rightarrow (z_0 > z_1 \vee \cdots \vee z_0 > z_l \vee (z_0 = z_1 = \cdots = z_l))$$

where $0 \leq k, l$. Note that k or l might be 0: if $k = 0$, we obtain a formula of the form $z_0 > z_1 \vee \cdots \vee z_0 > z_l$ or $(z_0 > z_1 \vee \cdots \vee z_0 > z_l \vee (z_0 = z_1 = \cdots = z_l))$, and if $l = 0$ we obtain a disjunction of disequalities. Also note that the variables $x_1, \dots, x_k, y_1, \dots, y_k, z_0, \dots, z_l$ need not be pairwise distinct. Also note that the clause $z_1 > z_2 \vee z_3 > z_4$ is *not* equivalent to an ll-Horn formula.

PROPOSITION 3.5. *A temporal relation is ll-closed if and only if it can be defined by an ll-Horn formula.*

We first prove Lemma 3.6 below about relations that only contain *injective* tuples. A tuple is said to be *injective* if all entries of the tuple are pairwise distinct. Note that every temporal relation R can be defined by a quantifier-free formula in conjunctive normal form where all literals are of the form $x > y$ or $x = y$; to see this, take any quantifier-free formula in conjunctive normal form that defines R and

- replace $x \leq y$ by the two literals $x < y \vee x = y$;
- replace $x \neq y$ by the two literals $x > y \vee y > x$;
- replace $x < y$ by $y > x$.

We call formulas in quantifier-free conjunctive normal form where all literals are of the form $x > y$ or $x = y$ *standard formulas*. A clause is *bad* if it is not of the form $z_0 > z_1 \vee \cdots \vee z_0 > z_l$.

LEMMA 3.6. *Let R be a temporal relation that only contains injective tuples, and let ϕ be a standard formula with minimal number of bad clauses such that*

- a) *an injective tuple is in R if and only if it satisfies ϕ ;*
- b) *any formula obtained from ϕ by removing a literal from a clause does not have this property.*

If R is ll-closed, then ϕ does not contain bad clauses.

PROOF. Suppose that R is n -ary, and that x_1, \dots, x_n are the variables of ϕ . Because R only contains injective tuples, we can remove literals of the form $y = z$ from ϕ . But this would contradict assumption a), so we assume that ϕ only contains literals of the form $x > y$.

Suppose for contradiction that ϕ contains a bad clause C . Then C must contain two literals $l_1 := x_u > x_v$ and $l_2 := x_r > x_s$ where x_u and x_r are distinct variables. We claim that there is an injective tuple t_1 such that l_1 is the only literal satisfied in C if we assign $t_1[i]$ to x_i for $1 \leq i \leq n$. Otherwise, the formula obtained from ϕ by removing l_1 from C still has the property that every injective tuple is in R if and only if it satisfies ϕ . Moreover, the number of bad clauses in the new formula is also minimal, which is impossible by the choice of ϕ . Similarly one can see that there is an injective tuple t_2 such that l_2 is the only literal satisfied in C if we assign $t_2[i]$ to x_i .

We first study the case that t_1 can be chosen such that $t_1[r]$ is smaller than $t_1[s]$, $t_1[u]$, and $t_1[v]$. Let α be an automorphism of $(\mathbb{Q}, <)$ such that $t_1[r]$ is mapped to 0. Consider the tuple $t = ll(\alpha(t_1), t_2)$. Observe that t is injective since ll preserves \neq . If a literal in some clause of ϕ is not satisfied in both tuples t_1 and t_2 , then it is also not satisfied in t , because ll and α preserve \leq . Therefore only the literals l_1 and l_2 of C can be satisfied by t . Since $t[r]$ is strictly smaller than $t[s]$ (by the properties of ll), the literal l_2 cannot be satisfied by t in C . Since $t_2[v] > t_2[u]$, it also holds that $t[v] > t[u]$ (by the properties of ll), and hence l_1 is not satisfied in t either. So t does not give a satisfying assignment for ϕ , in contradiction with the assumption that R is ll-closed.

An analogous argument shows that t_2 cannot be chosen such that $t_2[u]$ is smaller than $t_2[v]$, $t_2[r]$, and $t_2[s]$. We claim that any injective tuple that satisfies ϕ also satisfies $(x_u > x_v \vee x_u > x_s)$. If there was an injective tuple t with $t[u] < t[v]$ and $t[u] < t[s]$, then $t[u] > t[r]$ to satisfy the property from the beginning of the paragraph. Hence, $t[r] < t[v]$ and $t[r] < t[s]$. But then $t[r]$ is smaller than $t_2[v]$, $t_2[r]$, and $t_2[s]$, in contradiction to what we have shown before. Analogously we can show that any injective tuple that satisfies ϕ also satisfies $(x_r > x_v \vee x_r > x_s)$.

Let ϕ' be the formula obtained from ϕ by removing C and adding these two clauses. We show that an injective tuple satisfies ϕ' if and only if it satisfies ϕ . By what we have seen above, it suffices to show that ϕ' implies ϕ . Let t be any satisfying assignment of ϕ' . Clearly, all the clauses of ϕ except for C are satisfied by t , because they are also present in ϕ' . We can reformulate the two additional clauses in ϕ' to

$$(x_u > x_v \wedge x_r > x_v) \vee (x_u > x_v \wedge x_r > x_s) \vee (x_u > x_s \wedge x_r > x_v) \vee (x_u > x_s \wedge x_r > x_s).$$

If the first, the second, or the fourth disjunct is satisfied by t , then $t[x_u] > t[x_v]$ \vee

$t[x_r] > t[x_s]$, and therefore C holds in t . If the third disjunct is satisfied by t and the literal l_1 does not hold (i.e., $t[x_u] < t[x_v]$), we have the chain of inequalities $t[x_s] < t[x_u] < t[x_v] < t[x_r]$ and hence $t[x_r] > t[x_s]$. Thus, also in this last case C holds.

The formula ϕ' has fewer bad clauses than ϕ . Let ϕ'' be the formula obtained from ϕ' by repeatedly removing literals from clauses as long as an injective tuple is in R if and only if it satisfies ϕ'' . Since removing literals does not create new bad clauses, we eventually obtain a formula that contradicts the choice of ϕ .

We thus have shown that ϕ cannot contain bad clauses. \square

PROOF OF PROPOSITION 3.5. The proof that every relation defined by an ll-Horn formula is ll-closed is similar to the proof of Proposition 3.4. We just need to additionally check that the relation defined by $z_0 > z_1 \vee \dots \vee z_0 > z_l$ and the relation defined by $z_0 > z_1 \vee \dots \vee z_0 > z_l \vee (z_0 = \dots = z_l)$ are preserved by ll, which is straightforward.

The proof of the reverse implication is by induction on the arity n of the temporal relation R . We assume that R is ll-closed. For $n = 2$ the statement of the proposition holds, because all binary temporal relations can be defined by ll-Horn formulas. For $n > 2$, we construct the formula ψ that defines R as follows.

Let ϕ be a standard formula with minimal number of bad clauses such that a) an injective tuple is in R if and only if it satisfies ϕ , and b) any formula obtained from ϕ by removing a literal from a clause does not satisfy condition a). Clearly, such a formula exists: we can start from any standard formula that defines R and has a minimal number of bad clauses, and then remove repeatedly literals from clauses if the resulting formula still satisfies a); since deleting literals does not create bad clauses, we eventually find a formula that satisfies both conditions a) and b). Lemma 3.6 shows that ϕ does not contain bad clauses.

For all pairs of entries $i, j \in \{1, \dots, n\}$, $i < j$, let $R_{i,j}$ be the projection of the relation $R(x_1, \dots, x_{i-1}, x_j, x_{i+1}, \dots, x_n)$ to $x_1, \dots, x_{j-1}, x_{j+1}, \dots, x_n$. Because also $R_{i,j}$ is ll-closed, it has an ll-Horn definition $\phi_{i,j}$ by inductive assumption. We add to each clause of $\phi_{i,j}$ a literal $x_i = x_j$ to the premise of the implication, such that $\phi_{i,j}$ remains an ll-Horn formula.

Let ψ be the formula that is a conjunction of

- all the modified clauses from all formulas $\phi_{i,j}$;
- all clauses $C(z_0, \dots, z_l)$ of ϕ such that R does not contain a tuple where z_0, z_1, \dots, z_l all get the same value;
- the formula $C(z_0, \dots, z_l) \vee (z_0 = z_1 = \dots = z_l)$ for all other clauses C of ϕ with variables z_0, z_1, \dots, z_l .

Obviously, ψ is an ll-Horn formula. We have to verify that ψ defines R . Let t be an n -tuple such that $t \notin R$. If t is injective, then some clause $C(z_0, z_1, \dots, z_l)$ of ϕ is not satisfied by t . The variables z_0, z_1, \dots, z_l of C cannot all have the same value in t , and so ψ is not satisfied either. If there are i, j such that $t[i] = t[j]$ then the tuple $t_j = (t[1], \dots, t[j-1], t[j+1], \dots, t[n]) \notin R_{i,j}$. Therefore some clause C of $\phi_{i,j}$ is not satisfied by t_j , and $C \vee x_i \neq x_j$ is not satisfied by t . Thus, in this case t does not satisfy ψ , too.

We also have to verify that all $t \in R$ satisfy ψ . Let C be a conjunct of ψ created from some clause in $\phi_{i,j}$. If $t[i] \neq t[j]$, then C is satisfied by t because C contains $x_i \neq x_j$. If $t[i] = t[j]$, then $(t[1], \dots, t[j-1], t[j+1], \dots, t[n]) \in R_{i,j}$ and thus this tuple satisfies $\phi_{i,j}$. This also implies that t satisfies C .

Finally, let C be a conjunct of ψ created from some clause of ϕ . Then C is of the form $x_{u_0} > x_{u_1} \vee \dots \vee x_{u_0} > x_{u_m}$ or of the form $x_{u_0} > x_{u_1} \vee \dots \vee x_{u_0} > x_{u_m} \vee (x_{u_0} = x_{u_1} = \dots = x_{u_m})$. If t is constant on the variables of C , then, by construction of ψ , C contains the disjunct $x_{u_0} = x_{u_1} = \dots = x_{u_m}$ and is satisfied. So suppose t is not constant on the variables of C . Assume for contradiction that $t[u_0] \leq t[u_i]$ for all $i \in [m]$. Since t is not constant, there is a $j \in [m]$ such that $t[u_0] < t[u_j]$. By our assumptions on ϕ , there is an injective tuple $t' \in R$ that satisfies only the literal $x_{u_0} > x_{u_j}$ in C . Consider the tuple $t'' = \text{lex}(t, t')$. Because t' has pairwise distinct entries and lex is an injective operation, t'' also has pairwise distinct entries. Since $t[u_0] \leq t[u_i]$ for all $i \in [m]$, $t'[u_0] < t'[u_i]$ for all $i \in [m] \setminus \{j\}$, and because lex preserves \leq , we get that $t''[u_0] < t''[u_i]$ for all $i \in [m] \setminus \{j\}$. Finally, since $t[u_0] < t[u_j]$ we also have that $t''[u_0] < t''[u_j]$ by the properties of lex . Hence, $t''[u_0] < t''[u_i]$ for all $i \in [m]$. Therefore t'' does not satisfy C and thus also does not satisfy ϕ . But t'' is injective and is from R (because R is ll-closed), in contradiction to the properties of ϕ . \square

The syntactic characterization of ll-closed temporal relations motivates our the second way of representing the constraints in the input instances of the constraint satisfaction problem for ll-closed temporal constraint languages — the constraints may be given as ll-Horn formulas. We would like to remark that there are ll-closed temporal relations where the tuple representation is more succinct than the ll-Horn representation, and vice versa. Note that since every Ord-Horn formula is obviously an ll-Horn formula, the algorithm we present for the ll-Horn representation in Section 4 strictly generalizes the existing algorithms for Ord-Horn constraints. For simplicity, we assume that in instances of the constraint satisfaction problem for ll-closed TCLs the formulas representing the constraints consist of just one clause (we can always transform a constraint into several constraints of this form).

4. AN ALGORITHM FOR LL-CLOSED CONSTRAINTS

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

One of the underlying ideas of the algorithm is to use a subroutine that tries to find a solution where every variable has a different value. If this is impossible, the subroutine must return a set of at least two variables that denote the same value in all solutions. It is one of the fundamental properties of ll-closed constraints that this is always possible.

To formally introduce our algorithm we need the following definitions. Let $\phi = R(x_1, \dots, x_k)$ be an atomic formula where R is a temporal relation that is preserved by an operation f . Clearly, for all x_i from x_1, \dots, x_k the temporal relation defined by $\exists x_i. \phi$ is preserved by f as well. Therefore, if Φ is an instance of the CSP with constraints that are preserved by f , and \bar{y} is a sequence of some of the variables of Φ , then $\Phi' := \{\exists \bar{y}. \phi \mid \phi \in \Phi\}$ can also be viewed as an instance of the CSP with

constraints preserved by f . We call Φ' the *projection* of Φ to $X \setminus \overline{y}$. Note that if Φ' is unsatisfiable, then Φ is unsatisfiable as well.

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

DEFINITION 4.1. *Let R be a k -ary relation. A set $S \subseteq [k]$ is called a min-set for the i -th entry in R if there exists a tuple $t \in R$ such that the i -th entry is minimal in t , and for all $j \leq k$ it holds that $j \in S$ if and only if $t[i] = t[j]$. We say that t is a witness for this min-set.*

Let R be a k -ary relation that is preserved by lex (recall that ll-closed constraints are preserved by lex as well), and suppose that the i -th entry has the min-sets S_1, \dots, S_l , for $l \geq 1$, with the corresponding witnesses t_1, \dots, t_l . Consider the tuple $t := \text{lex}(t_1, \text{lex}(t_2, \dots, \text{lex}(t_{l-1}, t_l)))$. Since the entry i is minimal in every tuple t_1, \dots, t_l , and since lex preserves both $<$ and \leq , it is also minimal in t . Because lex is injective, we have that $t[i] = t[j]$ if and only if these two entries are equal in each tuple t_1, \dots, t_l . Hence, the min-set for the i -th entry in R witnessed by the tuple t is a subset of every other min-set S_1, \dots, S_l . We then call this set the *minimal min-set* for the i -th entry in R .

LEMMA 4.2. *Let R be a k -ary relation preserved by lex , $t \in R$, $i \in [k]$ and S be the minimal min-set for the i -th entry in R . If t is such that $t[j] \geq t[i]$ for every $j \in S$, then $t[i] = t[j]$ for every $j \in S$.*

PROOF. Let $t' \in R$ be the tuple that witnesses the minimal min-set S . Suppose there is a tuple $t \in R$ such that not all entries in S are equal (in particular, $|S| > 1$). Consider the tuple $t'' := \text{lex}(t', t)$. By the properties of lex it holds that $t''[i] < t''[j]$ for every $j \in [k] \setminus S$. Furthermore, $t''[i] \leq t''[j]$ for $j \in S$ if and only if $t[i] \leq t[j]$. Thus, unless t'' witnesses a smaller min-set for i in R (which would be a contradiction), we have that $t''[i] > t''[j]$ for some $j \in S$. \square

To develop our algorithm, we use a specific notion of *constraint graph* of a temporal CSP instance, defined as follows.

DEFINITION 4.3. *The constraint graph G_Φ of a temporal CSP instance Φ is a directed graph (X, E) defined on the variables X of Φ . For each constraint of the form $R(x_1, \dots, x_k)$ from Φ we add a directed edge $x_i x_j$ to E if in every tuple from R where the i -th entry is minimal the j -th entry is minimal as well.*

Example. We return to the example from Section 2. The constraint graph G_{Φ_1} for the instance Φ_1 in this example has the vertices x_1, x_2, y_1, y_2, y_3 , edges from each of y_1, y_2, y_3 to all other variables, and an edge from x_2 to x_1 .

DEFINITION 4.4. *If Φ contains a constraint ϕ imposed on y such that ϕ does not admit a solution where y denotes the minimal value, then we say that y is blocked (by ϕ).*

We can easily determine for each constraint which variables are blocked by this constraint: For a constraint represented by weak linear orders we just check all weak linear orders and build a set of variables that are not minimal in any of them. For a constraint represented by an ll-Horn formula, a variable x_i is blocked if and

only if the formula is of the form $x_i > z_1 \vee \dots \vee x_i > z_l$. Thus, by inspecting all the constraints it is possible to compute the blocked variables in linear time in the input size. We would like to use the constraint graph to identify variables that have to denote the same value in all solutions, and therefore introduce the following concepts.

DEFINITION 4.5. *A strongly connected component K of the constraint graph G_Φ for a temporal CSP instance Φ is called a sink component if no edge in G_Φ leaves K , and no variable in K is blocked. A vertex of G that belongs to a sink component of size one is called a sink.*

Example. In the previous example, the variables y_1, y_2, y_3 are blocked, and x_1 and x_2 are not blocked. The set of vertices $\{y_1, y_2, y_3\}$ forms a strongly connected component, which is not a sink component, because there are outgoing edges. (Moreover, the variables in K are blocked.) The singleton-set $\{x_1\}$ is a strongly connected component without outgoing edges and without blocked vertices, and thus x_1 is a sink.

The following lemma shows an important consequence of lex-closure of constraints.

LEMMA 4.6. *Let K be a sink component of the graph G_Φ for an instance Φ with lex-closed constraints. Then all variables from K must have equal values in all solutions of Φ .*

PROOF. We assume that Φ has a solution, and that K has at least two vertices (otherwise the lemma is trivial). Let t be a solution of Φ , and let $M \subseteq K$ be the set of variables that have in t the minimal value among the variables of the sink component K . If $M = K$, we are done.

Otherwise, because K is a strongly connected component, there is an edge in G_Φ from some vertex $u \in M$ to some vertex $v \in K \setminus M$. By the definition of G_Φ , there is a constraint ϕ in Φ such that whenever u denotes the minimal value of a solution of ϕ , then v has to denote the minimal value as well. By permuting arguments, we can assume without loss of generality that ϕ is of the form $R(w_1, \dots, w_k)$ where $w_1 = u$ and $w_2 = v$. Because K is a sink component, the variable u cannot be blocked, and hence there is a minimal min-set S for the first entry in R . Clearly, S contains 2, because v is the second argument of ϕ .

Note that G_Φ contains an edge from u to w_i for all $i \in S$. Since K is a strongly connected component, all these variables w_i are in K . Because u has in t the minimal value among the variables in K , there is no variable w_i , $i \in S$, which has a smaller value than u in t . This contradicts Lemma 4.2, because the value for u in t is different than the value for v . \square

Lemma 4.6 immediately implies that we can add constraints of the type $x = y$ for all variables x, y from the same sink component K . Equivalently, we can consider the CSP instance Φ' where all the variables in K are *contracted*, i.e., where all variables from K are replaced by the same variable. In some cases, a solution to a projected instance with ll-closed constraints can be used to construct a solution to the original constraint. We say that a tuple (in particular, a solution of an instance) \bar{x} is *injective* if $x_i \neq x_j$ for all $i \neq j$.

LEMMA 4.7. *Let Φ be an instance of the CSP with variables X and ll -closed constraints. Let x be a sink in G_Φ . If the projection Φ' of Φ to $X \setminus \{x\}$ has an injective solution, then Φ has an injective solution as well.*

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

We are ready to state our algorithm for instances with ll -closed constraints; the algorithm works for both representations of the constraints (sets of weak linear orders, ll -Horn formulas).

THEOREM 4.8. *The procedure $\text{Solve}(\Phi)$ in Algorithm 0?? decides whether a given set of ll -closed constraints Φ (where the relations are either represented by sets of weak linear orders, or by ll -Horn formulas) has a solution. There is an implementation of the algorithm that runs in time $O(nm)$, where n is the number of variables of Φ and m is the size of the input.*

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

In the first case, $Y = X$. We prove by induction that Φ has an injective solution. Let x_1, \dots, x_n be the elements from Y in the reverse order in which they were included into Y . For $0 \leq i \leq n$, let Φ_i be the instance Φ projected to $X \setminus \{x_1, \dots, x_i\}$. Note that $\Phi_0 = \Phi$, and that $\Phi_n = \Phi'$ is the projection of Φ to the empty set, which trivially has an injective solution. We inductively assume that Φ_i , for $i \leq n$, has an injective solution. Then Lemma 4.7 applied to x_i , the instance Φ_{i-1} , and the injective solution to Φ_i implies that also Φ_{i-1} has an injective solution. By induction, Φ_i has an injective solution for all $0 \leq i \leq n$, and in particular $\Phi_0 = \Phi$ has an injective solution. Therefore, the output true of Spec is correct.

Otherwise, in the second case, G' contains a sink component S with $|S| \geq 2$. We claim that for all variables $x, y \in S$ we have $x = y$ in all solutions to Φ . Lemma 4.6 applied to the projection of Φ to $X \setminus Y$ implies that whenever some variables are in the same sink component, they must have the same value in every solution, and hence the output is correct in this case as well.

In the third case, $Y \neq X$, but G' does not contain a sink component. Note that in every solution to Φ' some variable must take the minimal value. However, since each strongly connected component without outgoing edges contains a blocked vertex, there is no variable that can denote the minimal element, and hence Φ' has no solution. Because Φ' is a projection of Φ to $X \setminus Y$, the instance Φ is inconsistent as well.

```

Spec( $\Phi$ ) {
  // Input:  $\Phi$  constraints with variables  $X$ 
  // Output: If algorithm returns false then  $\Phi$  has no solution
  // If  $\Phi$  has an injective solution, then return true
  // Otherwise return  $S \subseteq X$ ,  $|S| \geq 2$ , s.t. for all
  //  $x, y \in S$  we have  $x = y$  in all solutions of  $\Phi$ 
   $G := \text{ConstructGraph}(\Phi)$ 
   $Y := \emptyset$ ,  $\Phi' := \Phi$ ,  $G' := G$ 
  While  $G'$  contains a sink  $s$ 
     $Y := Y \cup \{s\}$ 
     $\Phi' := \text{projection of } \Phi' \text{ to } X \setminus Y$ 
     $G' := \text{ReconstructGraph}(\Phi')$ 
  If  $Y = X$  then return true
  else if  $G'$  has sink component  $S$ 
    return  $S$ 
  else return false
  end if }

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

```

Fig. 2. An algorithm for ll-closed constraints.

Since in each recursive call of Solve the instance in the argument has at least one variable less, Solve is executed at most n times. It is not difficult to implement the algorithm such that the total running time is cubic in the input size. However, it is possible to implicitly represent the constraint graph and to implement all sub-procedures such that the total running time is in $O(nm)$, for both types of representations of the constraints studied in this paper. We will now describe the details how this can be achieved.

We have already shown the correctness of the algorithm, and only have to discuss how to implement the algorithm such that it runs in $O(nm)$. In fact, we describe an implementation of the procedure Spec that is linear in the input size.

First we show how to deal with constraints represented by ll-Horn clauses. Observe that if an ll-Horn clause has a non-empty left hand side of the implication, then a constraint for this clause creates neither edges nor blocked vertices in the constraint graph. Also constraints of the type $z_0 > z_1 \vee \dots \vee z_0 > z_l$ do not create edges in G_Φ . Thus, when constructing G_Φ , we only care about constraints of the type $z_0 > z_1 \vee \dots \vee z_0 > z_l \vee (z_0 = z_1 = \dots = z_l)$. For such constraints we add

edges from z_0 to z_1, \dots, z_l to G_Φ .

When the constraints in the input instance are represented by sets of weak linear orders we have to be more careful, and do not represent the edges of G_Φ explicitly, since there might be quadratically many edges which spoils the desired running time. We sort the weak linear orders \prec in each set according to the number of equivalence classes (of the equivalence relation defined by $x \prec y \wedge y \prec x$). Now, the data structure contains for each variable and each constraint that is imposed on this variable a reference to the weak linear order \prec in this constraint such that v is smallest with respect to \prec , and \prec has the largest number of equivalence classes. Moreover, for each element in each weak linear order \prec we create a list that contains the elements from the same equivalence class in \prec . Finally, for each variable v we also have a list that contains the constraints that are imposed on v and that block v . With bucket sort, the total cost to set up this data structure is linear in the input size. Even though the constraint graph G_Φ is not explicitly represented, it is possible to use the above data structure to compute the strongly connected components of G_Φ in linear time, using depth-first search.

Now we have to describe how the algorithm finds sinks, how the data structure is updated after projections, and how the algorithm finds sink components if there is no sink left and not all variables have been projected out. To find sinks and sink components, we also have to be able to determine efficiently whether a node is blocked or not.

Initially, because we have computed the strongly connected components, and because we know which variables are blocked, we can create a list that contains all sinks of the initial instance. Suppose that s is a sink of G at some iteration of the while-loop. We then first compute the projection of Φ' to $X \setminus Y$ by updating only the constraints imposed on s in Φ' . At this step we can also determine whether a constraint no longer blocks a variable v , and in this case we can update the list of blocking constraints for v . As soon as this list becomes empty, we know that v is no longer blocked. In this case, if v does not have outgoing edges in the current constraint graph, which we can determine efficiently using our updated data structure, we add v to the list of sinks. The total number of operations we have to perform in all iterations of the while-loop is then bounded by m .

Finally, if there is no sink left, but not all variables have been projected out, then we can compute the strongly connected components of the resulting constraint (again, this can be done in linear time using depth-first search on our data structure), and since we know which variables are blocked, we can also find the sink components.

Note that we can assume that n is smaller than m . Otherwise, the constraint is not connected (we use the notion of connectivity for instances of the CSP as e.g. in [HN04]). We can in this case use the same implementation, analyse the running time for each of the connected components separately, and get the same result.

This concludes the proof that for both representations studied in this paper the algorithm can be implemented such that it runs in time $O(nm)$. \square

5. LL-CLOSED CONSTRAINTS AND DATALOG

In this section, we prove that the constraint satisfaction problem for ll-closed constraints cannot be solved by Datalog programs⁴. For simplicity, the definition of the semantics of Datalog that we use here will be purely operational; for the standard semantical approach to the evaluation of Datalog programs see [EF99]. A Datalog program is a finite set of Horn clauses, i.e., clauses of the form $\psi \leftarrow \phi_1, \dots, \phi_l$, where $l \geq 0$ and where $\psi, \phi_1, \dots, \phi_l$ are atomic formulas of the form $R(\bar{x})$. The formula ψ is called the *head* of the rule, and ϕ_1, \dots, ϕ_l are called the *body*. We assume that all variables in the head also occur in the body. The relation symbols occurring in the head of some clause are called *intentional*, and all other relation symbols in the clauses are called *extensional*.

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

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

We prove that already for the TCL that only consists of R^{\min} there is no Datalog program that solves the corresponding constraint satisfaction problem. We use a pebble-game characterization of the expressive power of Datalog, which was originally shown in [FV99] and [KV98] for finite domain constraint satisfaction, and which holds for a wide variety of infinite domain constraint languages as well, including qualitative TCLs (see the journal version of [BD08]).

Let Γ be a finite TCL, and let Φ be an instance of $\text{CSP}(\Gamma)$. Then the *existential k-pebble game on Φ* is the following game between the players *Spoiler* and *Duplicator*. Spoiler has k pebbles p_1, \dots, p_k . He places his pebbles on variables from Φ . Initially, no pebbles are placed. In each round of the game Spoiler picks some of these pebbles. If they are already placed on Φ , then Spoiler first removes them from Φ . He then places the pebbles on variables from Φ , and Duplicator responds by assigning elements from \mathbb{Q} to these variables. This assignment has to satisfy all the

⁴This result should not be confused with the weaker fact that establishing k -consistency does not imply global consistency, for any k . This was shown for Ord-Horn in [Kou01]. But recall that Ord-Horn *can* be solved by a Datalog program [NB95].

constraints $\phi \in \Phi$ where all variables in ϕ are pebbled, otherwise Spoiler wins the game. Duplicator wins, if the game continues *forever*, i.e., if Spoiler can never win the game.

THEOREM 5.1 FROM [BD06]. *Let Γ be a finite TCL. There is no Datalog program that solves $CSP(\Gamma)$ if and only if for every k there exists an inconsistent instance of $CSP(\Gamma)$ such that Duplicator wins the existential k -pebble game on Φ .*

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

THEOREM 5.2. *There is no Datalog program that solves $CSP(\{R^{\min}\})$.*

PROOF. Let k be an arbitrary number. To apply Theorem 5.1 we have to construct an inconsistent instance Φ of $CSP(\{R^{\min}\})$ such that Duplicator wins the existential k -pebble game on Φ .

For this, let G be a 4-regular graph of girth at least $2k + 1$, i.e., all cycles in G have more than $2k$ vertices. It is known and easy to see that such graphs exist, e.g. with the methods in [JLR00]. Orient the edges in G such that there are exactly two outgoing and two incoming edges for each vertex in G . Since G is 4-regular, there exists an Euler tour for G (see e.g. [Die05]), which shows that such an orientation exists.

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

We now show that Duplicator has a winning strategy for the existential k -pebble game on this instance. Consider a connected non-empty subgraph G' of G having at most $2k$ vertices where only one vertex r has no outgoing edges, and where all vertices have either two or no incoming edges. Since G has girth $2k + 1$, G' must be a binary tree with root r . We call G' *dominated*, if all leaves in G' are pebbled.

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

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

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

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

6. CONCLUSION

While most of the polynomial algorithms that are known and used to solve infinite-domain constraint satisfaction problems are based on local consistency techniques, we used graph algorithms on an appropriately defined notion of constraint graph to both improve applicability (the constraint languages we can solve with our approach contain constraint relations whose CSP can not be solved with local consistency techniques – Theorem 5.2) and running time (our algorithm has quadratic running time, whereas resolution or establishing path consistency would require cubic time). We believe that similar approaches can lead to faster algorithms and larger tractable languages for many problems where the only known algorithms are based on local consistency techniques.

ACKNOWLEDGMENTS

We would like to thank Christopher Rudolf for implementing the algorithm, and the anonymous referees for their helpful comments.

REFERENCES

- J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
- Manuel Bodirsky and Víctor Dalmau. Datalog and constraint satisfaction with infinite templates. In *Proceedings of STACS'06*, pages 646–659. Springer, 2006.
- Manuel Bodirsky and Víctor 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.
- 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.
- Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. In *Proceedings of STOC'08*, pages 29–38, 2008.
- A. Bulatov, A. Krokhin, and P. G. Jeavons. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34:720–742, 2005.
- N. Creignou, M. Hermann, A. Krokhin, and G. Salzer. Complexity of clausal constraints over chains. Research Report, Ecole Polytechnique, 2005.
- Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
- Reinhard Diestel. *Graph Theory, 3rd edition*. Springer–Verlag, New York, 2005.
- Thomas Drakengren and Peter Jonsson. Twenty-one large tractable subclasses of Allen’s algebra. *Artificial Intelligence*, 93:297–319, 1997.

- Ivo Duentzsch. Relation algebras and their application in temporal and spatial reasoning. *Artificial Intelligence Review*, 23:315–357, 2005.
- H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1999. 2nd edition.
- M. Fisher, D. Gabbay, and L. Vila. *Handbook of Temporal Reasoning in Artificial Intelligence*. Elsevier, 2005.
- 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.
- Michael Garey and David Johnson. *A guide to NP-completeness*. CSLI Press, 1978.
- 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.
- Martin Charles Golumbic and Ron Shamir. Complexity and algorithms for reasoning about time: a graph-theoretic approach. *Journal of the ACM*, 40(5):1108 – 1133, 1993.
- Pavol Hell and Jaroslav Nešetřil. *Graphs and Homomorphisms*. Oxford University Press, 2004.
- Wilfrid Hodges. *A shorter model theory*. Cambridge University Press, 1997.
- S. Janson, T. Luczak, and A. Rucinski. *Random Graphs*. John Wiley and Sons, 2000.
- Andrei Krokhin, Peter Jeavons, and Peter Jonsson. Reasoning about temporal relations: The tractable subalgebras of Allen’s interval algebra. *JACM*, 50(5):591–640, 2003.
- M. Koubarakis. Tractable disjunctions of linear constraints: Basic results and applications to temporal reasoning. *Theoretical Computer Science*, 266:311–339, 2001.
- Ph. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proceedings of PODS'98*, pages 205–213, 1998.
- Peter B. Ladkin and Rober D. Maddux. On binary constraint problems. *Journal of the Association for Computing Machinery*, 41(3):435–469, 1994.
- David Marker. *Model Theory: An Introduction*. Springer, 2002.
- Rolf H. Möhring, Martin Skutella, and Frederik Stork. Scheduling with and/or precedence constraints. *SIAM J. Comput.*, 33(2):393–415, 2004.
- Bernhard Nebel and Hans-Jürgen Bürkert. Reasoning about temporal relations: A maximal tractable subclass of Allen’s interval algebra. *JACM*, 42(1):43–66, 1995.
- Agnes Szendrei. *Clones in universal Algebra*. Séminaire de Mathématiques Supérieures. Les Presses de L’Université de Montréal, 1986.
- Peter van Beek. Reasoning about qualitative temporal information. *Artificial Intelligence*, 58:297–326, 1992.
- R. van der Meyden. The complexity of querying indefinite information about linearly ordered domains. *Journal of Computer and Systems Science*, 54(1):113–135, 1997.
- Marc Vilain, Henry Kautz, and Peter van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. *Reading in Qualitative Reasoning about Physical Systems*, pages 373–381, 1989.

Received August 2008; accepted February 2009