

## TWO-VARIABLE LOGIC WITH COUNTING AND A LINEAR ORDER

WITOLD CHARATONIK AND PIOTR WITKOWSKI

Institute of Computer Science, University of Wrocław, Poland  
*e-mail address:* Witold.Charatonik@cs.uni.wroc.pl, pwit@pwit.info

**ABSTRACT.** We study the finite satisfiability problem for the two-variable fragment of first-order logic extended with counting quantifiers ( $C^2$ ) and interpreted over linearly ordered structures. We show that the problem is undecidable in the case of two linear orders (in the presence of two other binary symbols). In the case of one linear order it is NEXPTIME-complete, even in the presence of the successor relation. Surprisingly, the complexity of the problem explodes when we add one binary symbol more:  $C^2$  with one linear order and in the presence of other binary predicate symbols is equivalent, under elementary reductions, to the emptiness problem for multcounter automata.

### 1. INTRODUCTION

Since 1930s, when Alonzo Church and Alan Turing proved that the satisfiability problem for first-order logic is undecidable, much effort was put to find decidable subclasses of this logic. One of the most prominent decidable cases is the two-variable fragment  $FO^2$ .  $FO^2$  is particularly important in computer science because of its decidability and connections with other formalisms like modal, temporal or description logics or applications in XML or ontology reasoning. The satisfiability of  $FO^2$  was proved to be decidable in [32, 24] and NEXPTIME-complete in [9].

All decidable fragments of first-order logic have limited expressive power and a lot of effort is being put to extend them beyond first-order logic while preserving decidability. Many extensions of  $FO^2$ , in particular with transitive closure or least fixed-point operators, quickly lead to undecidability [8, 12]. Extensions that go beyond first order logic, but their (finite) satisfiability problem remains decidable, include  $FO^2$  over restricted classes of structures where one [16] or two relation symbols [17] are interpreted as equivalence relations (but there are no other binary symbols); where one [26] or two relations are interpreted as linear orders [31]; where two relations are interpreted as successors of two linear orders [21, 7, 4]; where one relation is interpreted as linear order, one as its successor and another one as equivalence [1]; where one relation is transitive [33]; where an equivalence closure can be applied to two binary predicates [15]; where deterministic transitive closure can be applied to one binary relation [3]. It is known that the finite satisfiability problem is undecidable

---

2012 ACM CCS: [Theory of computation]: Logic.

*Key words and phrases:* Two-variable logic, counting quantifiers, linear order, satisfiability, complexity.  
Research supported by NCN Grant no. 2011/03/B/ST6/00346.

for  $\text{FO}^2$  with two transitive relations [14], with three equivalence relations [16], with one transitive and one equivalence relation [17], with three linear orders [13], with two linear orders and their two corresponding successors [21]. A summary of complexity results for extensions of  $\text{FO}^2$  with binary predicates being the order relations (in the absence of other binary relations) can be found in [22].

When studying extensions of  $\text{FO}^2$  it is enough to consider relational signatures with symbols of arity at most 2 [9]. Some of the above mentioned decidability results, e. g., [1, 3, 7, 21, 31], hold under the assumption that there are no other binary predicates in the signature; some, like [4, 15, 16, 17, 26, 33] are valid in the general setting. Similarly, for undecidability results additional binary symbols are required e. g., in [8, 12, 17], but not required in [13, 14, 16, 21].

The two-variable fragment with counting quantifiers ( $\text{C}^2$ ) extends  $\text{FO}^2$  by allowing counting quantifiers of the form  $\exists^{<k}$ ,  $\exists^{\leq k}$ ,  $\exists^{=k}$ ,  $\exists^{\geq k}$  and  $\exists^{>k}$ , for all natural numbers  $k$ . The two problems of satisfiability and finite satisfiability for  $\text{C}^2$  (which are two different problems as  $\text{C}^2$  does not have a finite model property) were proved to be decidable in [11]. Another decidability proof together with a  $\text{NEXPTIME}$ -completeness result under unary encoding of numbers in counting quantifiers can be found in [27]. Pratt-Hartmann in [28] established  $\text{NEXPTIME}$ -completeness of both satisfiability and finite satisfiability under binary encoding of numbers in counting quantifiers. All these algorithms are quite sophisticated, a significant simplification can be found in [29]. There are not many known decidable extensions of  $\text{C}^2$ . In [4] it is shown that finite satisfiability for  $\text{C}^2$  interpreted over structures where two binary relations are interpreted as forests of finite trees (which subsumes the case of two successor relations on two linear orders) is  $\text{NEXPTIME}$ -complete. [30] shows that the satisfiability and finite satisfiability problems for  $\text{C}^2$  with one equivalence relation are both  $\text{NEXPTIME}$ -complete. All extensions of  $\text{C}^2$  mentioned above allow arbitrary number of other binary relations.

In this paper we study extensions of  $\text{C}^2$  with linear orders. We show that the finite satisfiability problem for  $\text{C}^2$  with two linear orders, in the presence of other binary predicate symbols, is undecidable. For  $\text{C}^2$  with one linear order, even if the successor of this linear order is present, in the absence of other binary predicate symbols, it is decidable and  $\text{NEXPTIME}$ -complete. A surprising result is that when we add one more binary predicate symbol, the complexity of the problem explodes: the finite satisfiability problem for  $\text{C}^2$  with one linear order and in the presence of one more binary predicate symbol is equivalent, under elementary reductions, to the emptiness problem for multcounter automata. Thus it is decidable, but as complex as the reachability problem for vector addition systems.

Multicounter automata (MCA) are a very simple formalism equivalent to Petri Nets and vector addition systems (VAS) [25], which are used e. g., to describe distributed, concurrent systems and chemical/biological processes. One of the main reasoning tasks for VAS is to determine reachability of a given vector. It is known that this problem is decidable [18, 23, 19] and  $\text{EXPSpace}$ -hard [20], but precise complexity is not known, and after over 40 years of research it is even not known if the problem is elementary. We present a reduction from the emptiness problem of MCA (which is equivalent to the reachability for VAS) to finite satisfiability of  $\text{C}^2$  with one linear order and one more binary predicate symbol. Although we show that  $\text{C}^2$  with one linear order and its successor, in the presence of arbitrary number of binary predicate symbols is decidable, it is very unlikely that it has an elementary decision algorithm since existence of such an algorithm implies existence of an elementary algorithm for VAS reachability.

The current paper is a revised and extended version of [5].

## 2. PRELIMINARIES

We will consider finite satisfiability problems for the two-variable logic with counting ( $C^2$  for short) over finite structures, where some distinguished binary symbols are interpreted as linear orders or successors of linear orders. We will be interested in largest antireflexive relations  $<$  contained in linear orders  $\leq$ ; for a linear order  $\leq$  we write  $a < b$  iff  $a \leq b$  and  $a \neq b$ . In the rest of the paper by a linear order we mean the relation  $<$  rather than  $\leq$ . We use symbols  $<$ ,  $<_1$ ,  $<_2$  to denote linear orders and  $\#$ ,  $\#_1$ ,  $\#_2$  to denote their respective successors. Given a finite signature  $\Sigma$  we write  $\mathcal{O}(\Sigma, <, \#)$  to denote the class of finite structures over  $\Sigma$ , where  $<$  is interpreted as a linear order and  $\#(x, y)$  means that  $y$  is a successor of  $x$  in this order. We also adopt a similar notation for other classes of structures. Logics we consider will be denoted by  $C^2(\#_u, \#_b, I)$ , where  $\#_u, \#_b \in \mathbb{N} \cup \{*\}$  denote the number of unary resp. binary symbols allowed in formulas, and  $I$  is the signature of distinguished binary symbols. We use  $*$  to denote that  $\#_u$  resp.  $\#_b$  are unbounded and we do not count the size of  $I$  in  $\#_b$ .

Specifically, we will be interested in the following logics:  $C^2(*, *, \{<\})$ ,  $C^2(*, *, \{<, \#\})$  and  $C^2(*, 2, \{<_1, <_2\})$ . By  $C^2(*, *, \{<\})$  we mean the logic  $C^2$  over a signature that contains an arbitrary number of unary and binary predicates, and  $<$  is interpreted as a linear order. The definition of  $C^2(*, *, \{<, \#\})$  is similar, with the exception that  $\#$  is interpreted as the successor of  $<$ . The logic  $C^2(*, 2, \{<_1, <_2\})$  allows an arbitrary number of unary and at most 4 binary symbols, two of them are interpreted as linear orders. Notice that we do not allow constant symbols in signatures, but this does not cause loss of generality since constants can be simulated by unary predicates and counting quantifiers.

## 3. TWO LINEAR ORDERS

We start with the observation that the induced successor relation of a linear order can be expressed in  $C^2(*, *, \{<\})$ . More precisely, let  $s$  be a non-distinguished binary predicate. The following lemma says that  $s$  can be defined to mean the successor of  $<$  in  $C^2(*, 1, \{<\})$ . Intuitively, it is enough to state that  $s$  is a subrelation of  $<$  such that each element (with the exception of the least and the greatest one) has exactly one  $s$ -successor and exactly one  $s$ -predecessor.

**Lemma 3.1.** *There exists a formula  $\varphi_s$  of  $C^2(*, 1, \{<\})$  such that for every finite structure  $\mathcal{M}$ , we have  $\mathcal{M} \models \varphi_s$  if and only if  $s^{\mathcal{M}}$  is the induced successor relation of  $<^{\mathcal{M}}$ .*

*Proof.* Define  $\varphi_s$  as conjunction of the following three formulas.

$$\forall x \forall y. s(x, y) \rightarrow x < y \tag{3.1}$$

$$\forall x. (\forall y. y < x \vee y = x) \vee \exists^=1 y. s(x, y) \tag{3.2}$$

$$\forall y. (\forall x. y < x \vee y = x) \vee \exists^=1 x. s(x, y) \tag{3.3}$$

Conjunct (3.1) of  $\varphi_s$  says that  $s^{\mathcal{M}}$  is a subrelation of  $<^{\mathcal{M}}$ . Conjuncts (3.2) and (3.3) state that every non-least (respectively, non-greatest) element w.r.t.  $<^{\mathcal{M}}$  has precisely one  $s$ -successor (respectively,  $s$ -predecessor).

Let  $\mathcal{M}$  be a finite model of  $\varphi_s$  and let  $e_1$  be the least w.r.t.  $<^{\mathcal{M}}$  element of  $\mathcal{M}$ . By a simple inductive argument we can construct a sequence  $e_1, \dots, e_k$  of elements of  $\mathcal{M}$  such that  $s^{\mathcal{M}}(e_i, e_{i+1})$  holds for all  $i \in \{1, \dots, k-1\}$  and  $e_k$  is the greatest element in  $\mathcal{M}$ . By conjuncts (3.2) and (3.3) no element occurs more than once in this sequence.

Observe that if the sequence  $e_1, \dots, e_k$  contains all elements of  $\mathcal{M}$  then  $s^{\mathcal{M}}$  is the induced successor relation of  $<^{\mathcal{M}}$ . Now, seeking for contradiction, assume that  $e$  is an element of  $\mathcal{M}$  that does not appear in the sequence. Then, again by induction, we can construct another finite sequence that contains  $e$ , is disjoint with  $e_1, \dots, e_k$  (so it contains neither the least nor the greatest element in  $\mathcal{M}$ ), and where every element has an  $s$ -successor and an  $s$ -predecessor. It follows that the second sequence is an  $s$ -cycle, which contradicts conjunct (3.1).

For the other direction, if  $\mathcal{M}$  is such that  $s^{\mathcal{M}}$  is the successor relation of linear order  $<^{\mathcal{M}}$ , then it can be checked by inspection that  $\mathcal{M} \models \varphi_s$ .  $\square$

**Corollary 3.2.** *Finite satisfiability of  $C^2(*, *, \{<, \neq\})$  is reducible in linear time to finite satisfiability of  $C^2(*, *, \{<\})$ . Finite satisfiability of  $C^2(*, 0, \{<_1, \neq_1, <_2, \neq_2\})$  is reducible in linear time to finite satisfiability of  $C^2(*, 2, \{<_1, <_2\})$ .*

Since  $\text{FO}^2(*, 0, \{<_1, \neq_1, <_2, \neq_2\})$ , i. e., the two-variable logic with two linear orders and their corresponding successors, is undecidable [21], we have the following conclusion.

**Corollary 3.3.** *Finite satisfiability problem of  $C^2(*, *, \{<_1, <_2\})$  is undecidable. This remains true even for  $C^2(*, 2, \{<_1, <_2\})$ .*

**Remark 3.4.** Observe that the proof of Lemma 3.1 works also in a setting where the symbol  $<$  is interpreted as an arbitrary acyclic relation (a relation is acyclic if its diagram is an acyclic graph). Actually, assuming that  $<^{\mathcal{M}}$  is an acyclic relation, the formula  $\varphi_s$  forces it to be the linear order induced by  $s^{\mathcal{M}}$ .

#### 4. $C^2(*, 0, \{<, \neq\})$ IS NEXPTIME-COMPLETE

We will show that finite satisfiability problem for  $C^2(*, 0, \{<, \neq\})$  is NEXPTIME-complete. Since the lower bound follows from the complexity of  $\text{FO}^2$  with only unary predicates [6, Theorem 11], we will concentrate on proving the upper bound. The proof presented here is inspired by a corresponding result [2] on  $\text{FO}^2$  on finite trees: we bring the input formula to a normal form and then check its local and global consistency, using a notion of types that is similar to the one introduced in [2].

We will be interested in  $C^2(*, 0, \{<, \neq\})$  formulas  $\varphi$  in normal form

$$\varphi = \forall x \forall y. \chi(x, y) \wedge \bigwedge_{h=1}^m \forall x \exists^{\leq C_h} y. \chi_h(x, y), \quad (4.1)$$

where  $\chi, \chi_1, \dots, \chi_m$  are quantifier-free formulas. Here symbols  $\leq_h$ , for  $h = \{1, \dots, m\}$ , denote either  $\leq$  or  $\geq$ , and  $C_1, \dots, C_m$  are positive integers encoded in binary. We refer to the number  $c = \max\{C_h \mid h \in \{1, \dots, m\}\}$  as the *height* of the formula  $\varphi$ . It is well known [10, Theorem 2.2] that by adding additional unary predicates each  $C^2$  formula  $\varphi$  can be transformed in polynomial time to a formula in normal form that is equisatisfiable with  $\varphi$  over models of cardinality at least  $c$ .

Observe that  $C^2(*, 0, \{<, \neq\})$  may be seen as a fragment of the weak monadic second-order logic with one successor WS1S, where unary relations are simulated by second-order existential quantifiers and counting quantifiers by first-order ones (e.g., a formula of the form  $\exists^{\leq k} x. \chi(x)$  can be replaced by an equivalent formula with  $k + 1$  universal quantifiers). However, this view leads to formulas with three alternations of quantifiers that can be checked for satisfiability in 4EXPTIME, which is not a desired complexity bound.

Because an element of a model of  $\varphi$  may require up to  $c$  witnesses for satisfaction, we will be interested in multisets counting these witnesses. Let  $\mathbb{N}_c = \{n \in \mathbb{N} \mid n \leq c\} \cup \{\infty\}$ . For  $k, k' \in \mathbb{N}_c$  define  $\text{cut}_c(k) = k$  if  $k \leq c$  and  $\text{cut}_c(k) = \infty$  if  $k > c$ . Define  $k \oplus_c k' = \text{cut}_c(k + k')$ . A  $c$ -multiset of elements from a given set  $A$  is any function  $f : A \rightarrow \mathbb{N}_c$ . For a given element  $a$  in  $A$ , the singleton  $\{a\}$  is the multiset defined by  $\{a\}(x) = 1$  if  $x = a$  and  $\{a\}(x) = 0$  for  $x \neq a$ . The union of two multisets  $f$  and  $g$  is a function denoted  $f \cup g$  such that  $(f \cup g)(x) = f(x) \oplus_c g(x)$ . We say that  $f$  is a *submultiset* of  $g$ , written  $f \subseteq g$ , if  $f(a) \leq g(a)$  for all  $a \in A$ . The empty multiset, denoted  $\emptyset$ , is the constant function equal 0 for all arguments. The following fact is obvious.

**Fact 4.1.** Every ascending (resp. descending) w.r.t.  $\subseteq$  chain of  $c$ -multisets consists of at most  $|A| * (c + 2)$  distinct elements.

Let us call maximal consistent formulas specifying the relative position of a pair of elements in a structure in  $\mathcal{O}(\Sigma, <, \#)$  *order formulas*. There are five possible order formulas:  $x=y \wedge \neg\#(y, x) \wedge \neg\#(x, y) \wedge y \not< x \wedge x \not< y$ ,  $x \neq y \wedge \#(y, x) \wedge \neg\#(x, y) \wedge y < x \wedge x \not< y$ ,  $x \neq y \wedge \#(x, y) \wedge \neg\#(y, x) \wedge x < y \wedge y \not< x$ ,  $x \neq y \wedge \neg\#(x, y) \wedge \neg\#(y, x) \wedge x < y \wedge y \not< x$ , and  $x \neq y \wedge \neg\#(x, y) \wedge \neg\#(y, x) \wedge y < x \wedge x \not< y$ . They are denoted, respectively, as:  $\theta_=$ ,  $\theta_{-1}$ ,  $\theta_{\#}$ ,  $\theta_{<}$ ,  $\theta_{>}$ . Let  $\Theta$  be the set of these five formulas.

A *1-type* over the signature  $\Sigma$  is a maximal consistent conjunction of atomic and negated atomic formulas over  $\Sigma$  involving only the variable  $x$ . The set of all 1-types over  $\Sigma$  will be denoted  $\Pi(\Sigma)$ . The family of all  $c$ -multisets of 1-types over the signature  $\Sigma$  is denoted  $\mathbb{N}_c^{\Pi(\Sigma)}$ .

To be able to check local consistency of a formula (see Definition 4.4 below) we introduce the notion of a *full type*. Intuitively, a full type of an element  $e$  in a structure contains enough information to check that the formula is (locally) true in  $e$ . The information stored in the full type of  $e$  tells us about 1-types of all other elements  $e'$  in the structure, divided into five multisets depending on relative positions of  $e$  and  $e'$ .

**Definition 4.2** (Full type over  $\Sigma$  w.r.t.  $c$ ). A *full type* over  $\Sigma$  w.r.t.  $c$  is a function  $\sigma : \Theta \rightarrow \mathbb{N}_c^{\Pi(\Sigma)}$ , such that  $\sigma(\theta_{-1})$  and  $\sigma(\theta_{\#})$  are singletons or empty, and  $\sigma(\theta_=)$  is a singleton.

**Definition 4.3** (Full type in  $\mathcal{A}$  w.r.t.  $c$ ). Let  $\mathcal{A}$  be a structure over a signature  $\Sigma$  and let  $a$  be an element of  $\mathcal{A}$ . The *full type* of  $a$  in  $\mathcal{A}$ , denoted  $\text{ft}^{\mathcal{A}}(a)$  is the function  $\sigma : \Theta \rightarrow \mathbb{N}_c^{\Pi(\Sigma)}$  such that

- $\sigma(\theta_=)$  is the singleton of the 1-type of  $a$  in  $\mathcal{A}$ ,
- $\sigma(\theta_{-1})$  is the singleton of the 1-type of the predecessor of  $a$  (if  $a$  has a predecessor) or empty multiset (if  $a$  has no predecessor),
- $\sigma(\theta_{\#})$  is the singleton of the 1-type of the successor of  $a$  (if  $a$  has a successor) or empty multiset (if  $a$  has no successor),
- $\sigma(\theta_{<})$  is the  $c$ -multiset of 1-types of elements strictly smaller than  $a$  in  $\mathcal{A}$ , excluding the predecessor (if it exists), and
- $\sigma(\theta_{>})$  is the  $c$ -multiset of 1-types of elements strictly greater than  $a$  in  $\mathcal{A}$ , excluding the successor (if it exists).

A structure  $\mathcal{A}$  is said to *realise* a full type  $\sigma$  if  $\text{ft}^{\mathcal{A}}(a) = \sigma$  for some  $a \in \mathcal{A}$ . In the following we identify a full type  $\sigma$ , which is a function, with the tuple  $\langle \sigma(\theta_{-1}), \sigma(\theta_=), \sigma(\theta_{\#}), \sigma(\theta_{<}), \sigma(\theta_{>}) \rangle$ .

Let  $\sigma$  be a full type w.r.t.  $c$  such that  $\sigma(\theta_=) = \{\pi\}$  and let  $\forall x \exists^{\leq h} C_h y. \chi_h(x, y)$  be a conjunct in  $\varphi$ . The following five functions are used to count witnesses w.r.t. this conjunct

for elements of full type  $\sigma$ .

$$\begin{aligned}
W_{=}^{\chi_h}(\sigma) &= \begin{cases} 1 & \text{if } \pi(x) \models \chi_h(x, x) \\ 0 & \text{otherwise} \end{cases} \\
W_{-1}^{\chi_h}(\sigma) &= \begin{cases} 1 & \text{if } \sigma(\theta_{-1}) = \{\pi'\} \text{ and } \pi(x) \wedge \pi'(y) \wedge \theta_{-1}(x, y) \models \chi_h(x, y) \\ 0 & \text{otherwise} \end{cases} \\
W_{\#}^{\chi_h}(\sigma) &= \begin{cases} 1 & \text{if } \sigma(\theta_{\#}) = \{\pi'\} \text{ and } \pi(x) \wedge \pi'(y) \wedge \theta_{\#}(x, y) \models \chi_h(x, y) \\ 0 & \text{otherwise} \end{cases} \\
W_{<}^{\chi_h}(\sigma) &= \text{cut}_c \left( \sum_{\pi': \pi(x) \wedge \pi'(y) \wedge \theta_{<}(y, x) \models \chi_h(x, y)} (\sigma(\theta_{<}))(\pi') \right) \\
W_{>}^{\chi_h}(\sigma) &= \text{cut}_c \left( \sum_{\pi': \pi(x) \wedge \pi'(y) \wedge \theta_{>}(y, x) \models \chi_h(x, y)} (\sigma(\theta_{>}))(\pi') \right)
\end{aligned}$$

Note that in the definition above  $(\sigma(\theta_{>}))(\pi')$  is simply the number of occurrences of the 1-type  $\pi'$  in the multiset  $\sigma(\theta_{>})$ .

The following definition and lemma formalise local consistency of a formula.

**Definition 4.4** (Compatible full types). Let  $\sigma$  be a full type w.r.t.  $c$  such that  $\sigma(\theta_{=}) = \{\pi\}$  and let  $\varphi$  be a formula of height  $c$  in normal form (4.1). We say that  $\sigma$  is *compatible* with  $\varphi$  if the following conditions are satisfied.

- $\pi(x) \models \chi(x, x)$ ,
- $\pi(x) \wedge \pi'(y) \wedge \theta(x, y) \models \chi$  for all  $\theta \in \{\theta_{-1}, \theta_{\#}, \theta_{<}, \theta_{>}\}$  and all  $\pi' \in \sigma(\theta)$ , and
- for each conjunct  $\forall x \exists^{<_h C_h} y. \chi_h(x, y)$  of  $\varphi$  we have

$$W_{=}^{\chi_h}(\sigma) + W_{-1}^{\chi_h}(\sigma) + W_{\#}^{\chi_h}(\sigma) + W_{<}^{\chi_h}(\sigma) + W_{>}^{\chi_h}(\sigma) \leq_h C_h$$

It is quite obvious that whenever  $\mathcal{A} \models \varphi$ , all full types realised in  $\mathcal{A}$  are compatible with  $\varphi$ . It is not difficult to see that the converse is also true, as the following lemma says.

**Lemma 4.5.** *For any ordered structure  $\mathcal{A}$  and any  $C^2(*, 0, \{<, \#\})$  formula  $\varphi$  in normal form, if all full types realised in  $\mathcal{A}$  are compatible with  $\varphi$  then  $\mathcal{A} \models \varphi$ .*

*Proof.* Take arbitrary two elements of the structure  $\mathcal{A}$ . If the two elements are equal then the first item of Definition 4.4 guarantees that they satisfy the conjunct  $\chi$  of  $\varphi$ . If they are different,  $\chi$  is satisfied by the second item. In any case, the conjunct  $\chi$  is satisfied. Similarly, take any element  $e$  of the structure and consider any conjunct  $\forall x \exists^{<_h C_h} y. \chi_h(x, y)$  of  $\varphi$ . Each element  $e'$  such that  $\mathcal{A} \models \chi(e, e')$  belongs to exactly one of the five sets: the singleton of  $e$ , the singleton of the predecessor of  $e$ , the singleton of the successor of  $e$ , elements smaller than the predecessor of  $e$  and the elements greater than the successor of  $e$ . The five functions used in the third item of Definition 4.4 correctly (up to  $c$ ) count the number of such elements  $e'$  in these five sets. Since the constant  $C_h$  does not exceed  $c$ , the condition in the third item guarantees that the conjunct is satisfied.  $\square$

To be able to check global consistency of a formula (see Lemma 4.6 below) we introduce the notion of the graph  $G_{\Sigma,c}$ . It is the graph  $\langle V, E \rangle$  where the set  $V$  of nodes is the set of full types over  $\Sigma$  w.r.t.  $c$  and the set  $E$  of edges is defined as follows.

$$\langle \langle \Pi_{-1}, \{\pi\}, \{\pi_{\neq}\}, \Pi_{<}, \Pi_{>} \rangle, \langle \{\pi\}, \{\pi_{\neq}\}, \Pi'_{\neq}, \Pi'_{<}, \Pi'_{>} \rangle \rangle \in E \quad \text{iff} \quad \begin{aligned} \Pi'_{<} &= \Pi_{<} \cup \Pi_{-1} \quad \text{and} \\ \Pi'_{>} &= \Pi'_{>} \cup \Pi'_{\neq} \end{aligned}$$

Intuitively, a path in this graph describes a possible evolution of full types in a structure. Each edge in the graph describes the change in a full type when we move from an element in the structure to its successor: (the singleton containing the 1-type of) the successor is moved to the position of current element; the current element is moved to the predecessor position; the predecessor is added to (the multiset of the 1-types of the) strictly smaller elements; and finally the multiset of strictly greater elements is split into the new multiset of strictly greater elements and the new successor element.

We define the graph  $G_{\Sigma,c}^{\varphi}$  as the subgraph of  $G_{\Sigma,c}$  consisting of nodes compatible with  $\varphi$ . The nodes of the form  $\langle \emptyset, \dots, \dots, \emptyset, \dots \rangle$  are called *source* nodes; the nodes of the form  $\langle \dots, \dots, \emptyset, \dots, \emptyset \rangle$  are called *target* nodes. Intuitively, a source node corresponds to a full type of the least element in some model of  $\varphi$  while a target node corresponds to the greatest element in some model.

**Lemma 4.6.** *Let  $\varphi$  be a  $C^2(*, 0, \{<, \neq\})$  formula of height  $c$  in normal form, over signature  $\Sigma$ . Then  $\varphi$  is finitely satisfiable if and only if there exists a path from a source node to a target node in the graph  $G_{\Sigma,c}^{\varphi}$ .*

*Proof.* First, assume that  $\varphi$  is finitely satisfiable, and let  $\mathcal{A}$  be a model of  $\varphi$ . Let  $a_1, \dots, a_k$  be all elements of  $\mathcal{A}$  ordered w.r.t.  $<^{\mathcal{A}}$ . Then  $\text{ft}^{\mathcal{A}}(a_1), \dots, \text{ft}^{\mathcal{A}}(a_k)$  is a path from a source node to a target node in  $G_{\Sigma,c}^{\varphi}$ .

Second, assume that there exists a path  $\sigma_1, \dots, \sigma_k$  from a source node to a target node in  $G_{\Sigma,c}^{\varphi}$ . We will construct a structure  $\mathcal{A}$  over elements  $a_1, \dots, a_k$  such that  $\mathcal{A} \models \varphi$ . Let the universe of  $\mathcal{A}$  consist of  $k$  distinct elements  $a_1, \dots, a_k$ . Define unary predicates in  $\mathcal{A}$  in such a way that for all  $i \in \{1, \dots, k\}$  the 1-type of  $a_i$  coincides with  $\sigma_i(\theta_{=})$ . Then define the relation  $\neq^{\mathcal{A}}$  as  $\{\langle a_i, a_{i+1} \rangle \mid 1 \leq i \leq k-1\}$  and  $<^{\mathcal{A}}$  as the transitive closure of  $\neq^{\mathcal{A}}$ .

The definition of  $\neq^{\mathcal{A}}$  guarantees that for  $\theta \in \{\theta_{-1}, \theta_{\neq}\}$  and for all  $i$  we have  $\sigma_i(\theta) = \text{ft}^{\mathcal{A}}(a_i)(\theta)$ . A simple inductive proof shows that  $\sigma_i$  coincides with  $\text{ft}^{\mathcal{A}}(a_i)$  on  $\theta_{<}$  and  $\theta_{>}$ , too. Since  $\sigma_1$  is a source node,  $\sigma_1(\theta_{<})$  is the empty multiset, which coincides with the value of  $\text{ft}^{\mathcal{A}}(a_1)$  on  $\theta_{<}$ . In the inductive step, assuming that  $\sigma_i(\theta_{<})$  and  $\text{ft}^{\mathcal{A}}(a_i)(\theta_{<})$  coincide, we show the same for  $\sigma_{i+1}$  and  $\text{ft}^{\mathcal{A}}(a_{i+1})$ : both the edge relation  $E$  and the definition of full types require that the values of  $\sigma_{i+1}$  and  $\text{ft}^{\mathcal{A}}(a_{i+1})$  on  $\theta_{<}$  are the  $c$ -multiset union of the value of  $\sigma_i$  on  $\theta_{<}$  with the singleton of the 1-type of the predecessor of  $a_i$ . The case of the value on  $\theta_{>}$  is symmetric: the induction starts in  $k$  with empty multisets and goes down to 1 adding in each step respective successors. This shows that  $\text{ft}^{\mathcal{A}}(a_i) = \sigma_i$  for all  $i \in \{1, \dots, k\}$ . Therefore all full types realised in  $\mathcal{A}$  are compatible with  $\varphi$  and by Lemma 4.5 we have  $\mathcal{A} \models \varphi$ .  $\square$

Lemma 4.6 leads us directly to the main theorem of this section. To check satisfiability of a formula in  $C^2(*, 0, \{<, \neq\})$  it is enough to guess an appropriate path in  $G_{\Sigma,c}^{\varphi}$ . Moreover, it is enough to use only exponentially many different full types in the guessed path.

**Theorem 4.7.** *The finite satisfiability problem for  $C^2(*, 0, \{<, \neq\})$  is NEXPTIME-complete.*

*Proof.* The lower bound follows from the complexity of the finite satisfiability problem for  $\text{FO}^2$  with only unary predicates. For the upper bound, an algorithm for deciding finite satisfiability of  $\text{C}^2(*, 0, \{<, \neq\})$  works as follows. It takes a  $\text{C}^2(*, 0, \{<, \neq\})$  formula  $\psi$  and converts it to a normal form  $\varphi$  (in polynomial time). It also checks (by nondeterministic guessing) if there exists a model of  $\psi$  of cardinality at most  $c$ , where  $c$  is the height of  $\varphi$ . Then the algorithm guesses a path from a source node to a target node in  $G_{\Sigma, c}^{\varphi}$  where  $\Sigma$  is the signature of  $\varphi$ . This requires in particular verification of the fact that all nodes are compatible with  $\varphi$ .

All this can be accomplished in time polynomial in the size of the graph. This size is potentially doubly exponential in  $|\varphi|$ : the number of all 1-types over  $\Sigma$  is exponential in  $|\varphi|$ , so the number of sets of 1-types, and, in consequence, the number of full types, is doubly exponential. The potential  $2\text{NEXPTIME}$  complexity of the algorithm can be lowered to  $\text{NEXPTIME}$  using the observation that the  $\theta_{<}$  and  $\theta_{>}$  components of full types behave in a monotone way along any path connecting any source node with any target node. The  $\theta_{<}$  component may only increase and  $\theta_{>}$  only decrease along any such path. Thus multisets on a path form an ascending resp. descending chain, and by Fact 4.1 there are only  $O(|\Pi(\Sigma)| * c)$  such multisets occurring along the path. Therefore it is enough to guess only exponentially many (in  $|\varphi|$ ) different full types.  $\square$

## 5. HARDNESS OF $\text{C}^2(*, 1, \{<\})$

In this section we show that the finite satisfiability problem for  $\text{C}^2(*, 1, \{<\})$ , with a binary relation  $s$ , is at least as hard as non-emptiness of multicounter automata. Similar reductions from emptiness of multicounter automata in this context can be found e.g. in [1] and [22]. Here, for a given multicounter automaton  $M$ , we first construct a  $\text{C}^2(*, 1, \{<, \neq\})$  formula  $\varphi_M$  which has a finite model if and only if  $M$  is non-empty. Then, using the idea from Section 3, we argue that the reduction can be modified to work for  $\text{C}^2(*, 1, \{<\})$ .

We adopt a notion of multicounter automata (MCA for short) similar to one in [1] or [22], but with empty input alphabet and simplified counter manipulation. The exposition below closely follows the one from the long version of [22]. Intuitively, an MCA is a finite state automaton without input but equipped with a finite set of counters which can be incremented and decremented, but not tested for zero. More formally, a multicounter automaton  $M$  is a tuple  $\langle Q, C, R, \delta, q_I, F \rangle$ , where the set  $Q$  of states, the initial state  $q_I \in Q$  and the set  $F \subseteq Q$  of final states are as in usual finite state automata,  $C$  is a finite set (the *counters*) and  $R$  is a subset of  $C$ . The transition relation  $\delta$  is a subset of

$$Q \times \{inc(c), dec(c), skip \mid c \in C\} \times Q.$$

An MCA is called *reduced* if it does not have skip transitions and  $R = C$  (in this case we just omit the  $R$  component of tuple  $M$ ).

A *configuration* of a multicounter automaton  $M$  is a pair  $\langle p, \vec{n} \rangle$  where  $p$  is a state and  $\vec{n} \in \mathbb{N}^C$  gives a value  $\vec{n}(c)$  for each counter  $c$  in  $C$ . Transitions with  $inc(c)$  and  $skip$  can always be applied, whereas transitions with  $dec(c)$  can only be applied to configurations with  $\vec{n}(c) > 0$ . Applying a transition  $\langle p, inc(c), q \rangle$  to a configuration  $\langle p, \vec{n} \rangle$  yields a configuration  $\langle q, \vec{n}_0 \rangle$  where  $\vec{n}_0$  is obtained from  $\vec{n}$  by incrementing its  $c$ -th component and keeping values of all other components unchanged. Analogously, applying (an applicable) transition  $\langle p, dec(c), q \rangle$  to a configuration  $\langle p, \vec{n} \rangle$  yields a configuration  $\langle q, \vec{n}_0 \rangle$  where  $\vec{n}_0$  is obtained from  $\vec{n}$  by decrementing its  $c$ -th component. Transitions with  $skip$  do not change value

of any counter in  $C$ . A *run* is an interleaving sequence of configurations and transitions  $conf_1, trans_1, \dots, trans_{k-1}, conf_k$  such that  $trans_i$  applied to  $conf_i$  gives  $conf_{i+1}$ , for  $1 \leq i < k$ . A run is *accepting*, if it starts in configuration  $\langle q_I, \vec{0} \rangle$  and ends in some configuration  $\langle q_F, \vec{n}_F \rangle$  with  $q_F \in F$  and  $\vec{n}_F(c) = 0$  for every  $c \in R$  (note that for counters  $c \in C \setminus R$  the value  $\vec{n}_F(c)$  may be arbitrary). The emptiness problem for multicounter automata is the question whether a given automaton  $M$  has an accepting run. It is well known that this problem (for both MCA and reduced MCA) is decidable, as it is polynomial-time equivalent to the reachability problem in Vector Addition Systems/Petri Nets[18, 23].

**Definition 5.1.** Let  $M = \langle Q, C, \delta, q_I, F \rangle$  be a reduced MCA. Let  $\Sigma = \{q \mid q \in Q\} \cup \{inc_c, dec_c \mid c \in C\} \cup \{\min, \max, <, \#1, s\}$  where predicates  $q, inc_c, dec_c, \min$  and  $\max$  are unary and  $<, \#1$  and  $s$  are binary. Define  $\varphi_M$  as the conjunction of the following  $\Sigma$ -formulas.

- (1)  $\exists^{=1}x. \min(x) \wedge \exists^{=1}x. \max(x)$
- (2)  $\forall x \forall y. (\min(x) \rightarrow (x < y \vee x = y)) \wedge (\max(x) \rightarrow (y < x \vee y = x))$
- (3)  $\forall x. \left( \bigvee_{q \in Q} q(x) \right) \wedge \bigwedge_{q \in Q} \left( q(x) \rightarrow \bigwedge_{q' \in Q \setminus \{q\}} \neg q'(x) \right)$
- (4)  $\forall x. (\min(x) \rightarrow q_I(x)) \wedge \left( \max(x) \rightarrow \bigvee_{q_F \in F} q_F(x) \right)$
- (5)  $\forall x \forall y. \#1(x, y) \rightarrow \bigvee_{(q, inc(c), q') \in \delta} (q(x) \wedge inc_c(x) \wedge q'(y)) \vee \bigvee_{(q, dec(c), q') \in \delta} (q(x) \wedge dec_c(x) \wedge q'(y))$
- (6)  $\forall x. (\neg \max(x)) \rightarrow \bigvee_{c \in C} (inc_c(x) \vee dec_c(x))$
- (7)  $\forall x. \bigwedge_{c \in C} \left( inc_c(x) \rightarrow \neg dec_c(x) \wedge \bigwedge_{c' \in C \setminus \{c\}} (\neg dec_{c'}(x) \wedge \neg inc_{c'}(x)) \right)$
- (8)  $\forall x. \bigwedge_{c \in C} \left( dec_c(x) \rightarrow \neg inc_c(x) \wedge \bigwedge_{c' \in C \setminus \{c\}} (\neg inc_{c'}(x) \wedge \neg dec_{c'}(x)) \right)$
- (9)  $\forall x. \max(x) \rightarrow \bigwedge_{c \in C} (\neg inc_c(x) \wedge \neg dec_c(x))$
- (10)  $\forall x \forall y. s(x, y) \rightarrow \bigvee_{c \in C} (inc_c(x) \wedge dec_c(y))$
- (11)  $\forall x \forall y. s(x, y) \rightarrow x < y$
- (12)  $\forall x. (\max(x) \vee \exists^{=1}y. (s(x, y) \vee s(y, x)))$

We will interpret  $\varphi_M$  as a  $C^2(*, 1, \{<, \#1\})$  formula. Models of  $\varphi_M$  encode accepting runs of MCA  $M$ . The first two conjuncts of  $\varphi_M$  define the meaning of the auxiliary predicates  $\min$  and  $\max$ ; they hold for the least (resp. the greatest) element of a model. Each element of the model corresponds to precisely one state  $q \in Q$ , as specified by Conjunct 3. Thus the model is just a sequence of states. The first of them must be the starting state  $q_I$  and the last must be a final state  $q_F \in F$ , as defined by Conjunct 4. Every two consecutive elements of the model form a transition. A state in which the transition is fired is marked by predicate of the form  $inc_c$  or  $dec_c$  denoting a counter to increment or decrement; this is specified by Conjunct 5. Every state, with the exception of the last one, must be labelled by precisely one predicate of the form  $inc_c$  or  $dec_c$ , as expressed by Conjuncts 6–8. The last element is not labelled by any of these predicates (Conjunct 9), as no transition is fired there. Since the values of all counters in starting and final state is 0 and no counter may fall below 0, each incrementation of a counter  $c$  must be eventually followed by its decrementation, and conversely, each decrementation of  $c$  must be preceded by its incrementation. We use the relation  $s$  to match these increments and decrements, as stated in Conjunct 10. Conjunct 11 states that decrementation of a counter indeed follows its incrementation. Since each state, except the final one, is a starting state of some transition, it either corresponds to incrementation or decrementation of some counter. Therefore it emits or accepts precisely

one edge labelled  $s$ , as stated by Conjunct 12 of  $\varphi_M$ . Formally, we have the following lemma and a corollary that results from it.

**Lemma 5.2.** *Let  $M = \langle Q, C, \delta, q_I, F \rangle$  be a reduced multicounter automaton and let  $\varphi_M$  be the  $C^2(*, 1, \{<, \# \})$  formula constructed in Definition 5.1. Formula  $\varphi_M$  is finitely satisfiable if and only if  $M$  is non-empty.*

*Proof.* First, assume that  $M$  is non-empty. Then  $M$  has an accepting run of the form  $conf_1, trans_1, \dots, trans_{k-1}, conf_k$ , where  $conf_i = \langle q_i, \vec{n}_i \rangle$  for  $i \in \{1, \dots, k\}$ , and  $trans_i = \langle q_i, inc(c_i), q_{i+1} \rangle$  or  $trans_i = \langle q_i, dec(c_i), q_{i+1} \rangle$ , for  $i \in \{1, \dots, k-1\}$ . We will construct a structure  $\mathcal{A}$  on elements  $e_1, \dots, e_k$  and show that  $\mathcal{A}$  models  $\varphi_M$ . Define  $(<)^{\mathcal{A}} = \{ \langle e_i, e_j \rangle \mid i, j \in \{1, \dots, k\}, i < j \}$  and  $(\#)^{\mathcal{A}} = \{ \langle e_i, e_{i+1} \rangle \mid i \in \{1, \dots, k-1\} \}$ . Label  $e_1$  by predicate  $\min$  and  $e_k$  by  $\max$ . This leads to satisfaction of Conjuncts 1 and 2 of  $\varphi_M$ . Label each element  $e_i$  by  $q_i$ , for  $i \in \{1, \dots, k\}$ . This leads to satisfaction of Conjunct 3. Conjunct 4 says that  $e_1$  should be labelled by  $q_I$  and  $e_k$  by some  $q_F$  with  $q_F \in F$ . Since run  $r$  is accepting, indeed we have  $q_I = q_1$  and  $q_k \in F$  and thus Conjunct 4 is satisfied by  $\mathcal{A}$ . For each  $i \in \{1, \dots, k-1\}$  label the element  $e_i$  by  $inc_{c_i}$  if  $trans_i = \langle q_i, inc(c_i), q_{i+1} \rangle$  or by  $dec_{c_i}$  if  $trans_i = \langle q_i, dec(c_i), q_{i+1} \rangle$ . This leads to satisfaction of Conjuncts 5–9. Finally we define relation  $s^{\mathcal{A}}$  in such a way that it connects  $e_i$  with  $e_j$  if  $trans_j$  decrements the same counter that is incremented by  $trans_i$  and the value of the counter after firing  $trans_j$  is for the first time equal to the value before firing  $trans_i$ . Formally,

$$\begin{aligned} s^{\mathcal{A}}(e_i, e_j) \quad \text{iff} \quad & i < j, \\ & trans_i = \langle q_i, inc(c_i), q_{i+1} \rangle, \\ & trans_j = \langle q_j, dec(c_j), q_{j+1} \rangle, \\ & c_i = c_j, \text{ and} \\ & j = \min\{l \mid l > i \wedge \vec{n}_{l+1}(c_i) = \vec{n}_i(c_i)\}. \end{aligned}$$

Conjuncts 10 and 11 of  $\varphi_M$  are satisfied by construction. In the last configuration all counters have value 0, so each incrementation of a counter has a later matching decrementation of the same counter. Similarly, in the first configuration all counters have value 0, so each decrementation of a counter has an earlier matching incrementation of the same counter. Therefore Conjunct 12 is also satisfied.

Second, assume that  $\varphi_M$  is finitely satisfiable and let  $\mathcal{A}$  be a model of  $\varphi_M$ . We will show that MCA  $M$  is non-empty. Assume that elements of  $\mathcal{A}$  sequenced in order  $(<)^{\mathcal{A}}$  are  $e_1, \dots, e_k$ . By Conjunct 3 of  $\varphi_M$  there exist a sequence of  $M$  states  $q_1, \dots, q_k$  such that  $q_i^{\mathcal{A}}(e_i)$  hold for every  $i \in \{1, \dots, k\}$ . By Conjuncts 6–9 each  $e_i$  with  $i < k$  is labelled by precisely one predicate of the form  $inc_c^{\mathcal{A}}(e_i)$  or  $dec_c^{\mathcal{A}}(e_i)$ . For  $i \in \{1, \dots, k\}$  let

$$conf_i = \langle q_i, \vec{n}_i \rangle, \text{ where for every } c \in C$$

$$\vec{n}_i(c) = |\{e_j \in \mathcal{A} \mid j < i \text{ and relation } s^{\mathcal{A}}(e_j, e_l) \text{ holds for some } l \geq i\}|.$$

For  $i \in \{1, \dots, k-1\}$  let

$$trans_i = \langle q_i, inc(c), q_{i+1} \rangle \text{ provided that } inc_c^{\mathcal{A}}(e_i) \text{ holds, or}$$

$$trans_i = \langle q_i, dec(c), q_{i+1} \rangle \text{ provided that } dec_c^{\mathcal{A}}(e_i) \text{ holds for some } c \in C.$$

The accepting run  $r$  of  $M$  is

$$conf_1, trans_1, \dots, trans_{k-1}, conf_k.$$

We will show that  $r$  is an accepting run. First we prove that it is a run of  $M$ . Assume that for some  $i < k$  sequence  $conf_1, trans_1, \dots, trans_{i-1}, conf_i$  forms a run. We will show that  $trans_i$  is applicable to configuration  $conf_i$ . This is clear when  $trans_i = \langle q_i, inc(c), q_{i+1} \rangle$ , for some  $c \in C$ . Otherwise  $trans_i = \langle q_i, dec(c), q_{i+1} \rangle$ . Therefore  $dec_c^A(e_i)$  holds. Since  $i < k$ , by Conjunct 12 of  $\varphi_M$ , element  $e_i$  emits or accepts precisely one  $s$  edge. Because it corresponds to decrementation of  $c$ , by Conjunct 10 it must accept  $s$  edge from some element  $e_j$ . By Conjunct 11 we have  $j < i$ . Therefore, by definition of  $\vec{n}_i$  we have  $\vec{n}_i(c) > 0$ . Thus  $trans_i$  is applicable to  $conf_i$  and  $conf_1, trans_1, \dots, trans_{i-1}, conf_i, trans_i, conf_{i+1}$  indeed forms a run.

Clearly,  $q_1 = q_I$  and  $q_k \in F$ , by Conjunct 4. Moreover,  $\vec{n}_1 = \vec{0}$  and  $\vec{n}_k = \vec{0}$ , thus  $conf_1$  and  $conf_k$  are starting and resp. final configuration. Thus run  $r$  is accepting and  $M$  is a non-empty MCA.  $\square$

**Corollary 5.3.** *The finite satisfiability problem for  $C^2(*, 1, \{<, \neq\})$  is at least as hard as the emptiness problem for multicounter automata.*

The result above can be strengthened by observing that  $\neq$  symbol is not necessary. Without this symbol in the signature we may still encode accepting runs of multicounter automata in  $C^2(*, 1, \{<\})$  by complicating slightly the definition of the predicate  $s$ . In the new encoding its purpose is not only to provide correspondence between increase and decrease of a counter but also to encode the successor relationship between elements of a structure. This, however, requires a slight change in a way we encode accepting runs. In Definition 5.1 a run  $conf_1, trans_1, \dots, trans_{i-1}, conf_i, trans_{k-1}, conf_k$  is represented as a structure with  $k$  elements, where the  $i$ -th element (for  $i < k$ ) represents both the state of  $conf_i$  and the counter manipulation defined by transition  $trans_i$ . In the modified encoding we separate elements representing states and transitions. In a  $C^2(*, 1, \{<\})$  formula we say that every element representing a state (with the exception of first and last one) has precisely one  $s$  edge to- and from- an element representing transition. Similarly, every transition element has precisely one  $s$  edge to- and from- a state element. Since we still require that  $s(x, y)$  implies  $x < y$ , it follows that a state element  $x$  (resp. a transition element) is connected to a transition element  $y$  (resp. a state element) by an  $s$  edge if and only if  $y$  is the successor of  $x$  w.r.t.  $<$  (we have seen a formula that specifies a similar property in Section 3). The formula does not constrain in any way  $s$  edges between two transition elements. Thus we may expand the formula with conjuncts that use  $s$  edges to match an increase of a counter in a transition with a decrease of the same counter in some following transition. This leads to the following corollary.

**Theorem 5.4.** *The finite satisfiability problem for  $C^2(*, 1, \{<\})$  is at least as hard as the emptiness problem for multicounter automata.*

Note that by Remark 3.4 we have the following corollary.

**Corollary 5.5.** *The finite satisfiability problem for  $C^2(*, 2, \{<\})$ , where  $<$  is interpreted as an acyclic relation, is at least as hard as the emptiness problem for multicounter automata.*

## 6. SATISFIABILITY OF $C^2(*, *, \{<, \neq\})$

In this section we show that the finite satisfiability problem of  $C^2(*, *, \{<, \neq\})$  is decidable. Here again the algorithm consists of three parts: we bring the input formula to a normal

form and then we check its local and global consistency. Local consistency is checked using *frames* that store information about possible types in a structure. Global consistency is checked using multicounter automata. Compared to the previous section, both types and automata are much more complicated.

**6.1. Normal form of  $C^2$  formulas.** For a natural number  $n$  denote by  $\underline{n}$  the set  $\{1, \dots, n\}$ . We will assume that input  $C^2(*, *, \{<, \neq\})$  formula  $\varphi$  is in a normal form

$$\varphi = \forall x \forall y. (\alpha(x, y) \vee x = y) \wedge \bigwedge_{h \in \underline{m}} \forall x \exists^=1 y. (f_h(x, y) \wedge x \neq y) \quad (6.1)$$

where  $\alpha$  is a quantifier-free formula with unary and binary predicate symbols and  $f_1, \dots, f_m$  are binary predicates. By a routine adaptation of transformation in [10] we may convert each  $C^2$  formula to an exponentially larger  $C^2$  formula  $\varphi'$  in normal form, such that  $\varphi$  and  $\varphi'$  are equisatisfiable (on structures of cardinality  $> 1$ ).

**Remark 6.1.** In Section 4 we use a notion of normal forms for  $C^2$  formulas with only polynomial blowup. Here, for simplicity of presentation, we decided to employ the one with exponential blowup. Since there is no elementary upper bound on the complexity of the problem to which we reduce our logic, the construction in the present section would not benefit from the usage of a more succinct normal form.

**6.2. 2-types and message types.** To be able to reason about local consistency of a formula we introduce several (standard in finite-model theory) notions of types, often following notations from [29]. Intuitively, a 2-type defined below is a generalisation of an order formula from Section 4. It contains enough information to check whether the conjunct  $\alpha$  from a formula in normal form (6.1) is locally true and if a given element is a witness for another element w.r.t. a conjunct  $\forall x \exists^=1 y. (f_h(x, y) \wedge x \neq y)$ . Star types defined in Section 6.4 generalise full types from Section 4 and contain enough information to check if an element has the right number of witnesses.

Fix a finite signature  $\Sigma$ . A *2-type* is a maximal consistent conjunction of atomic and negated atomic formulas over  $\Sigma$  involving only the variables  $x$  and  $y$  and satisfying three additional restrictions: first, it contains  $\neg(x = y)$ ; second, whenever it contains  $\neq(x, y)$  or  $\neq(y, x)$ , it also contains respectively  $x < y$  or  $y < x$ ; and third, it contains either  $x < y$  or  $y < x$ , but not both. We will identify a 1-type  $\pi$  (resp. a 2-type  $\tau$ ) with the set of positive atomic formulas occurring in  $\pi$  (resp. in  $\tau$ ). Each 2-type  $\tau(x, y)$  uniquely determines two 1-types of  $x$  and  $y$ , respectively, that we denote  $\text{tp}_1(\tau)$  and  $\text{tp}_2(\tau)$ . For a 2-type  $\tau$ , the 2-type obtained by swapping the variables  $x$  and  $y$  is denoted  $\tau^{-1}$ . The symbol  $\mathcal{T}(\Sigma)$  denotes the set of 2-types over  $\Sigma$ .

For a structure  $\mathcal{A}$  over the signature  $\Sigma$  and an element  $e \in \mathcal{A}$ ,  $\text{tp}^{\mathcal{A}}(e)$  denotes the unique 1-type  $\pi \in \Pi(\Sigma)$  such that  $\mathcal{A} \models \pi(e)$ . Similarly, for  $e_1, e_2 \in \mathcal{A}$ ,  $\text{tp}^{\mathcal{A}}(e_1, e_2)$  is the unique 2-type  $\tau \in \mathcal{T}(\Sigma)$  such that  $\mathcal{A} \models \tau(e_1, e_2)$ . If  $\mathcal{A} \models \tau(e_1, e_2)$ , we say that  $e_1$  *emits* the type  $\tau$  and  $e_2$  *accepts* it and that  $\tau$  *originates* in  $e_1$ . A 1-type  $\pi$  (resp. 2-type  $\tau$ ) is *realised* in  $\mathcal{A}$  if  $\pi = \text{tp}^{\mathcal{A}}(e)$  (resp.  $\tau = \text{tp}^{\mathcal{A}}(e_1, e_2)$ ) for some  $e \in \mathcal{A}$  (resp.  $e_1, e_2 \in \mathcal{A}$ , with  $e_1 \neq e_2$ ). The symbols  $\Pi(\mathcal{A})$  and  $\mathcal{T}(\mathcal{A})$  denote respectively the set of 1-types and the set of 2-types over  $\Sigma$  realised in  $\mathcal{A}$ . A 1-type  $\kappa \in \Pi(\Sigma)$  that has only one realisation in a structure  $\mathcal{A}$  is said to be a *royal 1-type* in  $\mathcal{A}$ . If an element  $e$  of  $\mathcal{A}$  realises a royal 1-type then it is said to be a *king* in  $\mathcal{A}$ . Any structure may have multiple kings.

If  $\Sigma$  is a relational signature and  $\bar{f} = f_1, \dots, f_m$  is a sequence of distinct binary predicates in  $\Sigma$ , then the pair  $\langle \Sigma, \bar{f} \rangle$  is called a *classified signature*. Let  $\langle \Sigma, \bar{f} \rangle$  be a classified signature and let  $\tau(x, y)$  be a 2-type over  $\Sigma$ . We say that  $\tau$  is a *message type* over  $\langle \Sigma, \bar{f} \rangle$  if  $f(x, y) \in \tau(x, y)$  for some predicate  $f$  in  $\bar{f}$ . Predicates in  $\bar{f}$  will be called *message predicates*. Given a structure  $\mathcal{A}$  over the signature  $\langle \Sigma, \bar{f} \rangle$  and an element  $a \in \mathcal{A}$ , we want to capture message types connecting  $a$  to other elements of  $\mathcal{A}$  and all 2-types connecting  $a$  to kings of  $\mathcal{A}$ . We first define the set of all these 2-types. If  $K$  is a set of royal 1-types from  $\mathcal{A}$ , then denote by  $\tau(K, \Sigma, \bar{f})$  the set of all 2-types  $\mu$ , such that  $\mu$  is a message type over  $\langle \Sigma, \bar{f} \rangle$  or  $\text{tp}_2(\mu) \in K$ . A 2-type from  $\tau(K, \Sigma, \bar{f})$  is called an *essential type*. If  $\tau$  is an essential type (resp. a message type) such that  $\tau^{-1}$  is also an essential type (resp. a message type) then we say that  $\tau$  is an *invertible essential type* (resp. *invertible message type*). On the other hand, if  $\tau$  is a 2-type such that neither  $\tau$  nor  $\tau^{-1}$  is an essential type, then we say that  $\tau$  is a *silent type*.

Given a structure  $\mathcal{A}$  over a classified signature  $\langle \Sigma, \bar{f} \rangle$  and a message type  $\tau$ , if  $\mathcal{A} \models \tau(e_1, e_2)$  then  $e_2$  is called a *witness for  $e_1$*  in  $\mathcal{A}$ . It follows that if  $\tau$  is an invertible message type then also  $e_1$  is a witness for  $e_2$ . This is because  $\mathcal{A} \models \tau^{-1}(e_2, e_1)$  and  $\tau^{-1}$  is an (invertible) message type. From now on we assume that the classified signature of  $\varphi$  is  $\langle \Sigma, \bar{f} \rangle$ , that is,  $\Sigma$  contains unary and binary predicates from  $\varphi$  and  $\bar{f} = f_1, \dots, f_m$  is the sequence of binary predicates occurring under existential quantifiers in  $\varphi$ .

Since we consider predicates of arity at most 2, a structure  $\mathcal{A}$  can be seen as a complete directed graph, where nodes are labelled by 1-types and edges are labelled by 2-types. Thus to define such a structure it is enough to define 1-types of its elements and 2-types of all pairs of elements provided that the projections of 2-types onto 1-types coincide with these 1-types and that for each pair  $\langle e_1, e_2 \rangle$  of elements connected by a 2-type  $\mu$  the pair  $\langle e_2, e_1 \rangle$  is connected by the 2-type  $\mu^{-1}$ . In the rest of the paper we use the notions of nodes and elements interchangeably.

**6.3. Normal structures.** Now, to simplify the reasoning, we restrict the class of models that we consider. Intuitively, besides some simple sanity conditions, we want to avoid non-royal 1-types with small number of occurrences. In other words, each 1-type either should be royal or it should have many occurrences in a structure. Here “many” means that we can always find a representative of this 1-type that is not a witness in any conjunct of the form  $\forall x \exists^{=1} y \dots$  and thus can be connected to any other non-royal element with a silent 2-type.

**Definition 6.2.** A finite structure  $\mathcal{A} \in \mathcal{O}(\Sigma, <, \uplus)$  over a classified signature  $\langle \Sigma, \bar{f} \rangle$  is *normal* if

- (1) both the smallest and the largest elements w.r.t.  $<^{\mathcal{A}}$  are kings in  $\mathcal{A}$ ,
- (2) for every two non-royal elements  $e_1, e_2 \in \mathcal{A}$  satisfying  $e_1 <^{\mathcal{A}} e_2$  there exist two elements  $e'_1, e'_2 \in \mathcal{A}$  such that  $e'_1 <^{\mathcal{A}} e'_2$ ,  $\text{tp}^{\mathcal{A}}(e_1) = \text{tp}^{\mathcal{A}}(e'_1)$ ,  $\text{tp}^{\mathcal{A}}(e_2) = \text{tp}^{\mathcal{A}}(e'_2)$ , and  $\text{tp}^{\mathcal{A}}(e'_1, e'_2)$  is a silent 2-type,
- (3) for every node  $e \in \mathcal{A}$  and  $f \in \bar{f}$  we have  $|\{e' \in \mathcal{A} \mid \mathcal{A} \models f(e, e')\}| = 1$ , and
- (4) for every  $e_1, e_2 \in \mathcal{A}$  if  $\uplus^{\mathcal{A}}(e_1, e_2)$  then  $\text{tp}^{\mathcal{A}}(e_1, e_2)$  is an invertible essential type.

Lemma 6.5 below says that when dealing with models of  $C^2(*, *, \{<, \uplus\})$  formulas, we may restrict to normal structures. To prove it we will need some technical lemmas. If  $\mathcal{A}$  is a model of a formula in  $C^2(*, *, \{<, \uplus\})$  and  $S$  is a set containing some elements of  $\mathcal{A}$ , then

by  $\min_{\mathcal{A}}(S)$  we denote the smallest element of  $S$  w.r.t. the linear order  $<^{\mathcal{A}}$ . Similarly, by  $\max_{\mathcal{A}}(S)$  we denote the largest element of  $S$  w.r.t.  $<^{\mathcal{A}}$ .

Let  $\mathcal{A}$  be a structure over  $\langle \Sigma, \bar{f} \rangle$  and  $\pi$  be a 1-type realised in  $\mathcal{A}$ . Let  $\{e_i\}_{i=1}^k$  be the sequence of all elements of  $\mathcal{A}$  that realise  $\pi$ , and such that  $e_i <^{\mathcal{A}} e_{i+1}$  for  $i \in \{1, \dots, k-1\}$ . Define  $S(\mathcal{A}, \pi) = \{e_i \mid i \in \{1, \dots, \min(k, 2m+1)\}\}$  and  $L(\mathcal{A}, \pi) = \{e_i \mid i \in \{\max(1, k+1-(2m+1)), \dots, k\}\}$ . Intuitively,  $S(\mathcal{A}, \pi)$  consists of  $\min(k, 2m+1)$  smallest (w.r.t.  $<^{\mathcal{A}}$ ) nodes that realise 1-type  $\pi$ , and  $L(\mathcal{A}, \pi)$  consists of  $\min(k, 2m+1)$  largest nodes that realise  $\pi$ . Recall that here  $m$  is the number of predicates in  $\bar{f}$ . If  $\pi, \pi'$  are two 1-types realised in  $\mathcal{A}$  then we say that a pair  $\langle \pi, \pi' \rangle$  is *correct* w.r.t. structure  $\mathcal{A}$  if  $|S(\mathcal{A}, \pi)| = 2m+1$ ,  $|L(\mathcal{A}, \pi')| = 2m+1$  and  $\max_{\mathcal{A}}(S(\mathcal{A}, \pi)) <^{\mathcal{A}} \min_{\mathcal{A}}(L(\mathcal{A}, \pi'))$ . A pair  $\langle \pi, \pi' \rangle$  is *incorrect* w.r.t.  $\mathcal{A}$  if it is not correct w.r.t.  $\mathcal{A}$ .

**Lemma 6.3.** *Let  $\mathcal{A}$  be a structure over  $\langle \Sigma, \bar{f} \rangle$ . Let  $\pi$  and  $\pi'$  be 1-types such that the pair  $\langle \pi, \pi' \rangle$  is correct w.r.t.  $\mathcal{A}$ . Then, for some distinct elements  $e, e' \in \mathcal{A}$  the 2-type  $\tau$  connecting  $e$  to  $e'$  is silent,  $\text{tp}_1(\tau) = \pi$ ,  $\text{tp}_2(\tau) = \pi'$  and  $(x < y) \in \tau$ .*

*Proof.* Since the pair  $\langle \pi, \pi' \rangle$  is correct, there exist two sequences of  $\mathcal{A}$ 's elements  $\{e_i\}_{i=1}^{2m+1}$  and  $\{e'_i\}_{i=1}^{2m+1}$  such that  $\max_{\mathcal{A}}(\{e_i\}_{i=1}^{2m+1}) <^{\mathcal{A}} \min_{\mathcal{A}}(\{e'_i\}_{i=1}^{2m+1})$ ,  $\text{tp}^{\mathcal{A}}(e_i) = \pi$  and  $\text{tp}^{\mathcal{A}}(e'_i) = \pi'$  for  $i \in 1, \dots, (2m+1)$ .

Consider the substructure of  $\mathcal{A}$  generated by (directed) edges connecting elements of  $\{e_i\}_{i=1}^{2m+1}$  to elements of  $\{e'_i\}_{i=1}^{2m+1}$ . The number of these edges is  $(2m+1)^2$ . Since each element needs at most  $m$  witnesses (and none of the types  $\pi, \pi'$  is royal), there are at most  $2(2m+1)m$  edges among them that are labelled with essential types. Since  $2(2m+1)m < (2m+1)^2$ , at least one of these edges is labelled with a silent type. Take such an edge. It connects some element  $e \in \{e_i\}_{i=1}^{2m+1}$  to some  $e' \in \{e'_i\}_{i=1}^{2m+1}$ . Let  $\tau = \text{tp}^{\mathcal{A}}(e, e')$ . Clearly,  $\tau$  is a silent 2-type,  $\text{tp}_1(\tau) = \pi$ ,  $\text{tp}_2(\tau) = \pi'$  and  $(x < y) \in \tau$ .  $\square$

In every finite structure, which we are dealing with, the largest (w.r.t.  $<$ ) node is always present. Thus predicate  $\neq$  cannot be among message predicates of any normal structure as it would violate Condition 3 of Definition 6.2. For similar reasons  $<$  cannot be one of these predicates. Therefore, to satisfy Conditions 3 and 4, we need the following lemma.

**Lemma 6.4.** *Let  $\mathcal{A}$  be a structure over a classified signature  $\langle \Sigma, \bar{f} \rangle$ . By interpreting two additional binary message-predicates we can expand  $\mathcal{A}$  to a structure  $\mathcal{B}$  such that for every  $e_1, e_2 \in \mathcal{B}$  if  $\neq^{\mathcal{B}}(e_1, e_2)$  then  $\text{tp}^{\mathcal{B}}(e_1, e_2)$  is an invertible essential type.*

*Proof.* Let  $e_1, \dots, e_k$  be all nodes of  $\mathcal{A}$  sequenced in order  $<^{\mathcal{A}}$ . Let  $f_{\text{back}}$  and  $f_{\text{forth}}$  be two fresh binary predicates. We add them to  $\bar{f}$  and expand the structure  $\mathcal{A}$  by putting  $f_{\text{forth}}^{\mathcal{B}} = \{\langle e_i, e_{i+1} \rangle \mid i < k\} \cup \{\langle e_k, e_1 \rangle\}$  and  $f_{\text{back}}^{\mathcal{B}} = \{\langle e_{i+1}, e_i \rangle \mid i < k\} \cup \{\langle e_1, e_k \rangle\}$ . Then for all  $i \in \underline{k-1}$  we have  $\text{tp}^{\mathcal{B}}(e_i, e_{i+1})$  is an invertible essential type.  $\square$

**Lemma 6.5.** *Let  $\varphi$  be a  $\text{C}^2(*, *, \{<, \neq\})$  formula in normal form, over a classified signature  $\langle \Sigma, \bar{f} \rangle$ . If  $\varphi$  is finitely satisfiable then there exists a signature  $\langle \Sigma', \bar{f}' \rangle$  such that  $\Sigma \subseteq \Sigma'$  and  $\bar{f} \subseteq \bar{f}'$  and a finite normal  $\langle \Sigma', \bar{f}' \rangle$ -structure  $\mathcal{B}$  such that  $\mathcal{B} \models \varphi$ . Moreover,  $|\Sigma'|$  is polynomial in  $|\Sigma|$  and  $|\bar{f}'| = |\bar{f}| + 2$ .*

*Proof.* Let  $\mathcal{A}$  be a model of  $\varphi$ . By Lemma 6.4 we may assume that Condition 4 of Definition 6.2 is satisfied. Condition 3 of this definition is also true, as  $\varphi$  is in normal form. The following algorithm constructs  $\mathcal{B}$  from  $\mathcal{A}$  in a sequence of steps numbered by a natural number  $j$ . In each step the algorithm interprets fresh unary predicates and does not change the interpretation of binary predicates thus not violating Conditions 3 and 4.

Initially  $j = 0$ , structure  $\mathcal{B}_0$  is  $\mathcal{A}$  where the smallest and the largest node is turned into king by interpreting two fresh unary predicates, and  $Pairs = \{\langle \pi, \pi' \rangle \mid \pi \text{ and } \pi' \text{ are non-royal 1-types realised in } \mathcal{B}_0\}$ .

- (1) while there exists a pair  $\langle \pi, \pi' \rangle \in Pairs$  which is incorrect w.r.t.  $\mathcal{B}_j$ , execute steps (2)–(3)
- (2) construct structure  $\mathcal{B}_{j+1}$  by labelling all nodes of  $\mathcal{B}_j$  from  $S(\mathcal{B}_j, \pi) \cup L(\mathcal{B}_j, \pi')$  by fresh unary predicates to turn them into kings of  $\mathcal{B}_{j+1}$ ,
- (3) remove pair  $\langle \pi, \pi' \rangle$  from  $Pairs$ , increment  $j$ ,
- (4) put  $\mathcal{B} = \mathcal{B}_j$  and terminate the algorithm

Clearly, the above algorithm terminates after at most  $|Pairs|$  steps, i. e., after  $O(2^{2^{|\Sigma|}})$  steps. At each step the algorithm creates at most  $2(2m + 1)$  new kings, so the total number of kings created by the algorithm is  $2(2m + 1)|Pairs|$ , which is  $O(2(2m + 1)2^{2^{|\Sigma|}})$ . The number of fresh unary predicates required to distinguish that number of kings is logarithmic w.r.t. that latter number. Therefore  $|\Sigma'|$  is polynomial in  $|\Sigma|$ . Since  $\mathcal{A}$  models  $\varphi$ , also  $\mathcal{B}$  models  $\varphi$ .

We now show that  $\mathcal{B}$  is normal. Condition 1 in Definition 6.2 is satisfied, because it is satisfied by  $\mathcal{B}_0$  and all kings of  $\mathcal{B}_0$  are left untouched during the construction of  $\mathcal{B}$  from  $\mathcal{B}_0$ .

In order to show that Condition 2 in Definition 6.2 is also satisfied, take non-royal elements  $e_1, e_2 \in \mathcal{B}$  such that  $e_1 <^{\mathcal{B}} e_2$ . Let  $\pi = \text{tp}^{\mathcal{B}}(e_1)$  and  $\pi' = \text{tp}^{\mathcal{B}}(e_2)$ . Since  $\pi$  and  $\pi'$  are non-royal 1-types realised in  $\mathcal{B}$ , they are also non-royal 1-types realised in  $\mathcal{A}$ . Thus, at the beginning of the algorithm we have  $\langle \pi, \pi' \rangle \in Pairs$ . Moreover, tuple  $\langle \pi, \pi' \rangle$  was never selected at step (1) of the algorithm. Therefore pair  $\langle \pi, \pi' \rangle$  is correct w.r.t.  $\mathcal{B}$ . By Lemma 6.3 we obtain the desired conclusion. Thus  $\mathcal{B}$  is a normal structure.  $\square$

**6.4. Star types.** Given a structure  $\mathcal{A}$  over a signature  $\langle \Sigma, \bar{f} \rangle$  and an element  $a \in \mathcal{A}$ , we want to capture essential 2-types emitted from  $a$  to other elements of  $\mathcal{A}$ . For this reason we introduce *star types*. Intuitively a star type of an element  $a$  in a structure describes a small neighbourhood of  $a$  and contains enough information to check whether  $a$  has the right number of witnesses.

**Definition 6.6** (Star type in  $\mathcal{A}$ ). Let  $\mathcal{A}$  be a normal structure over  $\langle \Sigma, \bar{f} \rangle$ , and let  $a$  be an element of  $\mathcal{A}$ . Let  $K = \{\kappa \mid \kappa \text{ is a royal type in } \mathcal{A}\}$ . The *star type* of  $a$  in  $\mathcal{A}$ , denoted  $\text{st}^{\mathcal{A}}(a)$  is a pair  $\sigma = \langle \pi, \mathcal{T} \rangle$  where  $\pi = \text{tp}^{\mathcal{A}}(a)$  and  $\mathcal{T}$  is the set of essential types originating in  $a$ :

$$\mathcal{T} = \{\mu \in \tau(K, \Sigma, \bar{f}) \mid \text{tp}^{\mathcal{A}}(a, b) = \mu \text{ for some } b \in \mathcal{A}\}.$$

We denote the type  $\pi$  by  $\pi(\sigma)$ . We say that a 2-type  $\mu$  *occurs* in  $\sigma$ , written  $\mu \in \sigma$ , if  $\mu \in \mathcal{T}$ . We write  $\sigma - \mu$  for the star-type  $\sigma' = \langle \pi, \mathcal{T} \setminus \{\mu\} \rangle$ . When  $S$  is a set of 2-types we write  $\sigma \setminus S$  to denote the star type  $\langle \pi, \mathcal{T} \setminus S \rangle$ .

Observe that in the definition above  $\sigma$  satisfies the conditions

- (1) for all  $\mu_1, \mu_2 \in \sigma$  if  $f(x, y) \in \mu_1$  and  $f(x, y) \in \mu_2$  for some  $f \in \bar{f}$  then  $\mu_1 = \mu_2$ ,
- (2)  $\mu \in \sigma$  implies  $\text{tp}_1(\mu) = \pi$  for all  $\mu \in \tau(K, \Sigma, \bar{f})$ ,
- (3)  $|\{\mu \in \sigma \mid \text{tp}_2(\mu) = \kappa\}| = 1$  for all  $\kappa \in K$  such that  $\kappa \neq \pi$ ,
- (4)  $|\{\mu \in \sigma \mid \text{tp}_2(\mu) = \pi\}| = 0$  if  $\pi \in K$ .

The first of these conditions is obvious, as normal structures emit precisely one edge  $\tau$  with  $f(x, y) \in \tau$  for  $f \in \bar{f}$ . The second one says that all 2-types originating in  $a$  have the same 1-type of the origin, namely the 1-type of  $a$ . The third one says that for all kings  $k$  (in  $\mathcal{A}$ ) the element  $a$  is connected with  $k$  by exactly one 2-type (provided that  $k \neq a$ ). The last

condition says that if  $a$  is a king then it is not connected with itself by any 2-type (recall that 2-types connect different elements).

**Definition 6.7.** A *star type* over the set of 2-types  $\tau(K, \Sigma, \bar{f})$  is any pair of the form  $\langle \pi, \mathcal{T} \rangle$  satisfying Conditions 1–4 above. A structure  $\mathcal{A}$  is said to realise a star type  $\sigma$  if  $\text{st}^{\mathcal{A}}(a) = \sigma$  for some  $a \in \mathcal{A}$ .

For a given set of star types ST, by  $\pi(\text{ST})$  we denote the set of 1-types  $\{\pi(\sigma) \mid \sigma \in \text{ST}\}$ . The set of 2-types occurring in star types from ST, that is the set  $\{\mu \mid \exists \sigma \in \text{ST} \ \mu \in \sigma\}$  is denoted by  $\tau(\text{ST})$ , and the set  $\{\langle \pi, \mathcal{T}' \rangle \mid \langle \pi, \mathcal{T} \rangle \in \text{ST} \text{ for some } \mathcal{T} \text{ satisfying } \mathcal{T}' \subseteq \mathcal{T}\}$  by  $\text{partial}(\text{ST})$ . Elements of  $\text{partial}(\text{ST})$  are called *partial star types*. A partial star type is said to be *empty* if it is of the form  $\langle \pi, \emptyset \rangle$ .

**6.5. Frames.** We now introduce finite and small structures called frames. Frames contain, among others, the information about all 2-types and all star types that may occur in a structure. This is enough to check whether the universal subformula  $\alpha$  of a formula in normal form (6.1) is true and if all elements have the right number of witnesses. Intuitively, a correctly guessed frame confirms that a formula is locally consistent (see Definition 6.9).

**Definition 6.8** (Frame). Let  $\langle \Sigma, \bar{f} \rangle$  be a classified signature,  $K$  be a set of 1-types over  $\Sigma$ , let ST be a set of star types over  $\tau(K, \Sigma, \bar{f})$  and let  $\Xi$  be a set of silent 2-types over  $\langle \Sigma, \bar{f} \rangle$ . The tuple  $\langle K, \text{ST}, \Xi, \Sigma, \bar{f} \rangle$  is called a *frame* if the following conditions are satisfied

- (1) for each 2-type  $\tau \in \tau(\text{ST}) \cup \Xi$  if  $\neg \mathbb{H}(x, y) \in \tau$  or  $\neg \mathbb{H}(y, x) \in \tau$  then  $\tau$  is an invertible essential type,
- (2) there exists exactly one star type  $\sigma_{\text{first}} \in \text{ST}$  such that for all  $\tau \in \sigma_{\text{first}}$  we have  $\neg \mathbb{H}(y, x) \notin \tau$ ,
- (3) there exists exactly one star type  $\sigma_{\text{last}} \in \text{ST}$  such that for all  $\tau \in \sigma_{\text{last}}$  we have  $\neg \mathbb{H}(x, y) \notin \tau$ ,
- (4) for each  $\kappa \in K$  there exists exactly one  $\sigma \in \text{ST}$  such that  $\pi(\sigma) = \kappa$ , and
- (5) for each star type  $\sigma \in \text{ST}$  and each 2-type  $\mu$ , if  $\mu \in \sigma$  then  $\text{tp}_2(\mu) \in \pi(\text{ST})$ .

Frames are intended to describe local configurations in normal structures  $\mathcal{A}$ . The set  $K$  contains all royal 1-types of  $\mathcal{A}$ , ST contains all star types of  $\mathcal{A}$  and the set  $\Xi$  contains all silent 2-types realised in  $\mathcal{A}$ . Condition 1 says that every node in  $\mathcal{A}$  is connected to its successor and predecessor by invertible essential types. Conditions 2 and 3 say that there are unique star types for the first and the last node in  $\mathcal{A}$ . Note that Conditions 1–3 are true in every normal structure. Condition 4 says that each king has exactly one star type. Condition 5 ensures that if a neighbour of a node in a structure has some 1-type  $\pi$ , then there exists a star type  $\sigma \in \text{ST}$  such that  $\pi \in \pi(\text{ST})$ . The last two conditions hold in every relational structure.

Intuitively, we want to check finite satisfiability of a  $\text{C}^2$  formula  $\varphi$  by guessing a right frame. “Right” means here that two conditions must be satisfied. First, the frame should be locally consistent with  $\varphi$ . This means that every 2-type occurring in the frame entails the subformulas of  $\varphi$  of the form  $\forall x \forall y \dots$ , and that the number of witnesses in every star type is correct. This is formalised in the following definition. Second, the frame should be globally consistent in the sense that there exists a structure that fits this frame – this is formalised in Definition 6.10.

**Definition 6.9** ( $\mathcal{F} \models \varphi$ ). Consider a frame  $\mathcal{F} = \langle K, \text{ST}, \Xi, \Sigma, \bar{f} \rangle$  and a  $\text{C}^2$   $(*, *, \{<, \mathbb{H}\})$  formula  $\varphi$  in normal form (6.1) over  $\langle \Sigma, \bar{f} \rangle$ . We say that  $\mathcal{F}$  *satisfies*  $\varphi$ , in symbols  $\mathcal{F} \models \varphi$ , if

- for each 2-type  $\mu \in \Xi \cup \tau(\text{ST})$ , the formula  $\alpha$  is a consequence of  $\mu$  and of  $\mu^{-1}$ , that is  $\models \mu \rightarrow \alpha$  and  $\models \mu^{-1} \rightarrow \alpha$ , where  $\mu$  is seen as conjunction of literals, and
- for each  $\sigma \in \text{ST}$  and  $h \in \underline{m}$  we have  $|\{\mu \in \sigma \mid f_h(x, y) \in \mu\}| = 1$ .

**Definition 6.10.** Let  $\langle K, \text{ST}, \Xi, \Sigma, \bar{f}, c \rangle$  be a frame, and  $\mathcal{A}$  a structure over  $\langle \Sigma, \bar{f} \rangle$ . We say that  $\mathcal{A}$  fits the frame  $\mathcal{F}$  if

- the set of royal 1-types realised in structure  $\mathcal{A}$  is  $K$ , and
- the set of all silent types realised in  $\mathcal{A}$  is a subset of  $\Xi$ , and
- the set of star types of  $\mathcal{A}$  is a subset of  $\text{ST}$ , in symbols  $\text{st}^{\mathcal{A}}(\mathcal{A}) \subseteq \text{ST}$ .

The following proposition reduces the finite satisfiability problem of  $\mathcal{C}^2(*, *, \{<, \neq\})$  to the problem of existence of a structure in  $\mathcal{O}(\Sigma, <, \neq)$  that fits a given frame. This is the reduction that splits the problem into local and global consistency.

**Proposition 6.11.** Let  $\varphi$  be a  $\mathcal{C}^2(*, *, \{<, \neq\})$  formula in normal form over a classified signature  $\langle \Sigma, \bar{f} \rangle$ , where  $\{<, \neq\} \subseteq \Sigma$ . Let  $\mathcal{A}$  be a structure in  $\mathcal{O}(\Sigma, <, \neq)$ .

- (1) If  $\mathcal{A}$  is normal and  $\mathcal{A} \models \varphi$  then there exists a frame  $\mathcal{F}$ , such that  $\mathcal{A}$  fits  $\mathcal{F}$  and  $\mathcal{F} \models \varphi$ .
- (2) If there exists a frame  $\mathcal{F}$  such that  $\mathcal{A}$  fits  $\mathcal{F}$  and  $\mathcal{F} \models \varphi$  then  $\mathcal{A} \models \varphi$ .

*Proof.* For the proof of the first statement, assume that  $\mathcal{A}$  is normal and  $\mathcal{A} \models \varphi$ . Let  $K$  be the set of royal 1-types realised in  $\mathcal{A}$ , let  $\text{ST}$  be the set of star types of  $\mathcal{A}$  and let  $\Xi$  be the set of all silent types realised in  $\mathcal{A}$ . The facts that the tuple  $\mathcal{F} = \langle K, \text{ST}, \Xi, \Sigma, \bar{f}, c \rangle$  forms a frame,  $\mathcal{F} \models \varphi$  and  $\mathcal{A}$  fits  $\mathcal{F}$  are immediate, once Definitions 6.8, 6.9 and 6.10 are unravelled.

For the proof of the second statement, let  $\mathcal{F}$  be a frame such that  $\mathcal{F} \models \varphi$  and let  $\mathcal{A}$  be a structure such that  $\mathcal{A}$  fits  $\mathcal{F}$ . Since  $\varphi$  is in normal form, it is of the form (6.1). Let  $\mu$  be any 2-type realised in  $\mathcal{A}$ . Then either  $\mu$  is a silent type or it occurs in some star type realised in  $\mathcal{A}$ . In any case, by Definition 6.10 we have that  $\mu \in \Xi \cup \tau(\text{ST})$ . By Definition 6.9 it follows that  $\models \mu \rightarrow \alpha$ . So  $\mathcal{A} \models \forall x \forall y. (\alpha \vee x = y)$ . Since  $\mathcal{A}$  fits  $\mathcal{F}$  and  $\mathcal{F} \models \varphi$ , it also follows that for each star type  $\sigma$  realised in  $\mathcal{A}$  and each  $h$  such that  $1 \leq h \leq m$  we have  $|\{\mu \in \sigma \mid f_h(x, y) \in \mu\}| = 1$ , and thus  $\mathcal{A} \models \bigwedge_{h \in \underline{m}} \forall x \exists^{=1} y. (f_h(x, y) \wedge x \neq y)$ . Hence  $\mathcal{A} \models \varphi$  as required.  $\square$

**6.6. High-level multicounter automata.** In the rest of this section we show how to decide, for a given frame  $\mathcal{F}$ , whether there exists a structure in  $\mathcal{O}(\Sigma, <, \neq)$  that fits this frame. This will be done by a reduction to the emptiness problem for multicounter automata.

We now introduce a syntactic extension to multicounter automata that we call *High-level MultiCounter Automata* (**HMCA**). The idea is to specify transitions of an automaton as programs in a higher-level imperative language with conditionals, loops and arrays, which leads to clearer exposition of reachability problems. A transition in a High-Level MCA is a sequence  $\Delta$  of actions; each action in turn may update and test finite-domain variables, and conduct conditional or loop instructions depending on results of these tests. A transition may also increment or decrement, but not test the value of, counters, which are the only infinite-domain variables of the automaton.

In the following subsections we give formal syntax and semantics of high-level multicounter automata and we prove that HMCA can be compiled to multicounter automata. If the reader is familiar with finite-state machines and how they handle finite information, it should be more or less clear how to translate HMCA to MCA. In such a case we suggest to skip Sections 6.6.2 and 6.6.3 and move directly to Section 6.7.

6.6.1. *Syntax of HMCA.* Formally, an HMCA is a tuple  $H = \langle V_{fin}, \text{Vec}_{\mathbb{N}}, \text{Type}, \Delta, \rho_I, P_F, E \rangle$ . The set  $V_{fin}$  consists of variables  $v$  to be interpreted in the finite domain  $\text{Type}(v)$ . We may think of  $V_{fin}$  as a declaration of finite-domain variables of the program. The set  $\text{Vec}_{\mathbb{N}}$  corresponds to a declaration of arrays, it consists of variables  $\vec{vec}$  to be interpreted as vectors of natural numbers indexed by elements of some finite set  $\mathbb{A}$ , where  $\text{Type}(\vec{vec}) = \mathbb{A} \rightarrow \mathbb{N}$ . We will refer to the index set  $\mathbb{A}$  as the domain  $\text{Dom}(\vec{vec})$ . The  $\text{Type}$  function assigns to every variable in  $V_{fin} \cup \text{Vec}_{\mathbb{N}}$  its type. The sequence of actions  $\Delta$  is the actual program built from actions defined below. The starting state of  $H$  is  $\rho_I$ , the set of accepting states is  $P_F$ . The set  $E$  is a subset of  $\{\langle \vec{v}, a \rangle \mid \vec{v} \in \text{Vec}_{\mathbb{N}}, a \in \text{Dom}(\vec{v})\}$ , and is used in the acceptance condition explained later.

We now define a set of actions  $\alpha$  that constitute transitions in **HMCA**. The simplest action is an *assignment* of the form  $v := \text{Expr}$ , where  $v$  is a variable of some domain  $\mathbb{A} = \text{Type}(v)$ , and  $\text{Expr}$  is an expression built from variables from  $V_{fin}$ , constants from appropriate domains and operators. An *operator* is any effectively computable function, e. g.,  $\cup, \cap, \setminus$  are operators of domain  $(2^{\mathbb{B}})^2 \rightarrow 2^{\mathbb{B}}$  for any domain  $\mathbb{B}$ ; function  $\pi : \text{ST}(\mathcal{K}, \Sigma, \vec{f}) \rightarrow \Pi(\Sigma)$  from Definition 6.6 is also an operator, provided that our finite domain contains  $\text{ST}(\mathcal{K}, \Sigma, \vec{f})$  and  $\Pi(\Sigma)$ . We silently extend the  $\text{Type}$  function to constants by letting  $\text{Type}(a) = \mathbb{A}$  if  $a \in \mathbb{A}$ , and to expressions, e. g.,  $\text{Type}(s_1 \cup s_2) = 2^{\mathbb{B}}$  if  $\text{Type}(s_1) = 2^{\mathbb{B}}$  and  $\text{Type}(s_2) = 2^{\mathbb{B}}$ . We require that assignments  $v := \text{Expr}$  are well typed, i. e., that  $\text{Type}(v) = \text{Type}(\text{Expr})$ . An *atomic test* is of the form  $\text{Expr}_1 = \text{Expr}_2$  or  $\text{Expr}_3 \in \text{Expr}_4$ , where  $\text{Expr}_1, \text{Expr}_2, \text{Expr}_3, \text{Expr}_4$  are expressions such that  $\text{Type}(\text{Expr}_1) = \text{Type}(\text{Expr}_2)$  and  $\text{Type}(\text{Expr}_4) = 2^{\text{Type}(\text{Expr}_3)}$ . A *test* is an arbitrary Boolean combination of *atomic tests*. Notice that tests do not use counters. A *non-deterministic assignment* action is of the form **guess**  $v \in \text{Expr}$  **with**  $\text{Test}$ , where  $\text{Type}(\text{Expr}) = 2^{\text{Type}(v)}$  and the variable  $v$  may occur in  $\text{Test}$ . A conditional action is of the form **if**  $\text{Test}$  **then**  $\alpha^*$  **else**  $\alpha'^*$  **endif** or **if**  $\text{Test}$  **then**  $\alpha^*$  **endif**. A loop action is of the form **while**  $\text{Test}$  **do**  $\alpha^*$  **endwhile**. An *incrementing action* (resp. *decrementing action*) is of the form **inc**( $\vec{f}[\text{Expr}]$ ) (resp. **dec**( $\vec{f}[\text{Expr}]$ )), where  $\text{Expr}$  evaluates to an index of the array  $\vec{f}$ , that is,  $\vec{f} \in \text{Vec}_{\mathbb{N}}$ ,  $\text{Type}(\vec{f}) = \mathbb{A} \rightarrow \mathbb{N}$  and  $\text{Type}(\text{Expr}) = \mathbb{A}$ . The remaining action, **Reject**, simply rejects current computation.

6.6.2. *Semantics of HMCA.* Expressions and tests are evaluated in context of variable valuations. A *variable valuation* (also called a *state*) is any function  $\rho$  that assigns to every finite-domain variable  $v$  a value  $\llbracket v \rrbracket_{\rho} \in \text{Type}(v)$ . The semantics of expressions is defined inductively:  $\llbracket v \rrbracket_{\rho} = \rho(v)$  for  $v \in V_{fin}$ ,  $\llbracket \text{Expr}_1 \bowtie \text{Expr}_2 \rrbracket_{\rho} = \llbracket \text{Expr}_1 \rrbracket_{\rho} \bowtie \llbracket \text{Expr}_2 \rrbracket_{\rho}$  where  $\bowtie \in \{\cup, \cap, \setminus\}$ , and  $\llbracket f(\text{Expr}_1, \dots, \text{Expr}_k) \rrbracket_{\rho} = f(\llbracket \text{Expr}_1 \rrbracket_{\rho}, \dots, \llbracket \text{Expr}_k \rrbracket_{\rho})$  where  $f$  is an operator. In a similar way we define semantics of tests.

A *counter valuation* is any function  $\vartheta$  that assigns (a sequence of) natural numbers to (arrays of) counters. A *configuration* of **HMCA**  $H$  is a pair  $\langle \rho, \vartheta \rangle$  where  $\rho$  is a variable valuation (i. e., a state) and  $\vartheta$  is a counter valuation. Actions transform configurations. Most actions work only on variable valuations; the exceptions are incrementing and decrementing of counters. With the exception of the decrementing action, the semantics of actions is self-explanatory;  $\text{dec}(c)$  decrements the counter  $c$  if it is strictly positive and otherwise (if it is 0) it rejects the current computation.

A *run* of an HMCA  $H$  is a sequence of configurations  $\langle \rho_1, \vartheta_1 \rangle, \dots, \langle \rho_k, \vartheta_k \rangle$  such that  $\langle \rho_{i+1}, \vartheta_{i+1} \rangle$  is obtained after executing transition  $\Delta$  in configuration  $\langle \rho_i, \vartheta_i \rangle$ , for all  $i \in \{1, \dots, k-1\}$ . A single transition  $(\langle \rho_i, \vartheta_i \rangle, \Delta) \rightsquigarrow \langle \rho_{i+1}, \vartheta_{i+1} \rangle$ , is formalised on Figures 1

$$\begin{array}{c}
\frac{\rho_2 = \rho_1[v \leftarrow \llbracket Expr \rrbracket_{\rho_1}]}{(\rho_1, v := Expr) \rightsquigarrow \rho_2} \\
\frac{e \in \llbracket Expr \rrbracket_{\rho_1} \quad \llbracket Test \rrbracket_{\rho_1[v \leftarrow e]} = \mathbf{true}}{(\rho_1, \mathbf{guess} \ v \in Expr \ \mathbf{with} \ Test) \rightsquigarrow \rho_1[v \leftarrow e]} \\
\frac{\llbracket Test \rrbracket_{\rho_1} = \mathbf{true} \quad (\rho_1, \alpha^*) \rightsquigarrow \rho_2}{(\rho_1, \mathbf{if} \ Test \ \mathbf{then} \ \alpha^* \ \mathbf{else} \ \alpha'^* \ \mathbf{endif}) \rightsquigarrow \rho_2} \\
\frac{\llbracket Test \rrbracket_{\rho_1} = \mathbf{false} \quad (\rho_1, \alpha'^*) \rightsquigarrow \rho_2}{(\rho_1, \mathbf{if} \ Test \ \mathbf{then} \ \alpha^* \ \mathbf{else} \ \alpha'^* \ \mathbf{endif}) \rightsquigarrow \rho_2} \\
\frac{\llbracket Test \rrbracket_{\rho_1} = \mathbf{true} \quad (\rho_1, \alpha^*) \rightsquigarrow \rho \quad (\rho, \mathbf{while} \ Test \ \mathbf{do} \ \alpha^* \ \mathbf{endwhile}) \rightsquigarrow \rho_2}{(\rho_1, \mathbf{while} \ Test \ \mathbf{do} \ \alpha^* \ \mathbf{endwhile}) \rightsquigarrow \rho_2} \\
\frac{\llbracket Test \rrbracket_{\rho_1} = \mathbf{false}}{(\rho_1, \mathbf{while} \ Test \ \mathbf{do} \ \alpha^* \ \mathbf{endwhile}) \rightsquigarrow \rho_1}
\end{array}$$

Figure 1: Semantics of actions that operate on state. For these actions  $\alpha$  we define  $(\langle \rho_1, \vartheta \rangle, \alpha) \rightsquigarrow \langle \rho_2, \vartheta \rangle$  if  $(\rho_1, \alpha) \rightsquigarrow \rho_2$ .

and 2. A run is *accepting*, if it starts in an initial configuration  $\langle \rho_I, \vartheta_0 \rangle$  with  $\rho_I$  being initial state and  $\vartheta_0$  assigning 0 to all counters, and it ends in some configuration  $\langle \rho_F, \vartheta_F \rangle$  with  $\rho_F$  being a final state and  $\vartheta_F$  assigning 0 to all counters specified in the set  $E$  of final counters:  $\vartheta_F(\vec{f})(a) = 0$  for every  $\langle f, a \rangle \in E$ . The emptiness problem for high-level multicounter automata is the question whether a given automaton  $H$  has an accepting run.

For a counter valuation  $\vartheta$ ,  $\vec{v} \in \text{Vec}_{\mathbb{N}}$  and a function  $f$  with domain  $\text{Type}(\vec{v})$  by  $\vartheta[\vec{v} \leftarrow f]$  we mean the valuation identical to  $\vartheta$ , with the exception that  $\vartheta(\vec{v}) = f$ . In a similar way, for a variable valuation  $\rho$ ,  $v \in \mathbb{V}_{fin}$  and  $a \in \text{Type}(v)$ , we define the variable valuation  $\rho[v \leftarrow a]$ .

Figures 1 and 2 present the natural semantics of actions in a formal way, separately for actions that operate on states and counters. The semantics of assignment, conditional and loop actions is the expected one. E. g., assignment  $v := Expr$  updates current state  $\rho$  by assigning to  $v$  the value  $\llbracket Expr \rrbracket_{\rho}$ ; conditional action **if**  $Test$  **then**  $\alpha^*$  **endif** executes  $\alpha^*$  provided that  $\llbracket Test \rrbracket_{\rho}$  is true, and skips to next action otherwise. A non-deterministic assignment **guess**  $v \in Expr$  **with**  $Test$  assigns to  $v$  an arbitrary value  $e \in \llbracket Expr \rrbracket_{\rho}$  such that  $Test$  holds when  $v$  becomes  $e$ , i. e., when  $\llbracket Test \rrbracket_{\rho[v \leftarrow e]}$  is true. Let  $\text{Type}(\vec{f}) = \mathbb{A} \rightarrow \mathbb{N}$  and  $e = \llbracket Expr \rrbracket_{\rho}$ . Incrementing action **inc**( $\vec{f}[Expr]$ ) transforms a configuration  $\langle \rho, \vartheta \rangle$  to  $\langle \rho, \vartheta[\vec{f} \leftarrow \vec{f}'] \rangle$ , where  $f'(a) = \vartheta(f)(a) + 1$  for  $a = e$  and  $f'(a) = \vartheta(f)(a)$  for remaining elements  $a \in \mathbb{A}$ . Decrementing action **dec**( $\vec{f}[Expr]$ ) works similarly, but it decrements the value  $\vartheta(f)(e)$ , provided that it is positive (the computation rejects if  $\vartheta(f)(e) = 0$ ). Finally, action **Reject** rejects the computation unconditionally.

6.6.3. *Compilation of HMCA.* Intuitively, a compilation of an HMCA to an MCA consists in hiding the control structures and finite-domain variables (including tests for zero on finite-domain variables) in states of the constructed MCA. Formally, we have the following proposition.

$$\begin{array}{c}
\frac{e = \llbracket Expr \rrbracket_{\rho_1} \quad \vartheta_2 = \vartheta_1[\vec{f}(e) \leftarrow \vec{f}(e) + 1]}{\left( \langle \rho_1, \vartheta_1 \rangle, \mathbf{inc}(\vec{f}(Expr)) \right) \rightsquigarrow \langle \rho_2, \vartheta_2 \rangle} \quad \frac{\langle \langle \rho_1, \vartheta_1 \rangle, \alpha \rangle \rightsquigarrow \langle \rho, \vartheta \rangle \quad \langle \langle \rho, \vartheta \rangle, \alpha^* \rangle \rightsquigarrow \langle \rho_2, \vartheta_2 \rangle}{\langle \langle \rho_1, \vartheta_1 \rangle, \alpha; \alpha^* \rangle \rightsquigarrow \langle \rho_2, \vartheta_2 \rangle} \\
\frac{e = \llbracket Expr \rrbracket_{\rho_1} \quad \vartheta_1(\vec{f})(e) > 0 \quad \vartheta_2 = \vartheta_1[\vec{f}(e) \leftarrow \vec{f}(e) - 1]}{\left( \langle \rho_1, \vartheta_1 \rangle, \mathbf{dec}(\vec{f}(Expr)) \right) \rightsquigarrow \langle \rho_2, \vartheta_2 \rangle} \quad \frac{}{\langle \langle \rho_1, \vartheta_1 \rangle, \epsilon \rangle \rightsquigarrow \langle \rho_1, \vartheta_1 \rangle}
\end{array}$$

Figure 2: Semantics of incrementing and decrementing actions and semantics of actions composition.

**Proposition 6.12.** Emptiness problem for **HMCA** is reducible to emptiness problem of multicounter automata, and is therefore decidable.

*Proof.* Let  $H = \langle V_{fin}, \text{Vec}_{\mathbb{N}}, \text{Type}, \Delta, \rho_I, P_F, E \rangle$  be an HMCA. We define an MCA  $M_H = \langle Q, C, R, \delta, q_I, F \rangle$  such that  $H$  is non-empty if and only if  $M_H$  is non-empty. Assume that  $\Delta$  consists of  $k$  actions. Let  $v_1, \dots, v_l$  be an arbitrary enumeration of  $V_{fin}$  and  $P = \text{Type}(v_1) \times \dots \times \text{Type}(v_l)$  be the set of all variable valuations. Let  $Q = \{1, \dots, k\} \times P$ . Every state  $q \in Q$  is a pair  $\langle i, \rho \rangle$ , which is intended to capture the number of an action to be executed and values of variables  $v_1, \dots, v_k$  in the state just before execution of the action. We use a shortcut  $q(v_i)$  to denote the  $i$ -th component of  $k$ -tuple  $\rho$  and  $q(1)$  to denote its first component. The set of counters  $C$  of MCA  $M_H$  is defined as  $C = \{c_{\vec{f}(e)} \mid \vec{f} \in \text{Vec}_{\mathbb{N}} \text{ and } e \in \text{Type}(\vec{f})\}$ . The starting state  $q_I$  of  $M_H$  is  $\langle 1, \rho_I \rangle$ . The set of final states  $F = \{\langle 1, \rho_F \rangle \mid \rho_F \in P_F\}$ . The set  $R = \{c_{\vec{f}(e)} \mid \langle \vec{f}, e \rangle \in E\}$ .

Assume that each action  $\alpha$  in  $\Delta$  is assigned a unique number  $i \in \mathbb{N}$ , interpreted as line number in which  $\alpha$  occurs. Below we will write  $\alpha_i$  to denote action  $\alpha$  occurring in line  $i$ . A *control flow graph* (CFG) of  $\Delta$  is a graph  $(\{i \in \mathbb{N} \mid \alpha_i \in \Delta\}, \{\rightarrow, \xrightarrow{\mathbf{true}}, \xrightarrow{\mathbf{false}}\})$ . Edges of CFG are defined as follows. For every action  $\alpha_i$ , with the exception of loop and conditional action, let  $j$  be the line number of next action to be executed. If no such action exists then we put  $j = 1$  denoting that  $\Delta$  may be reexecuted. We put  $i \rightarrow j$ . For loop or conditional actions let  $j_{\mathbf{true}}$  and  $j_{\mathbf{false}}$  be numbers of actions to be executed provided the action's test succeeds, resp. fails. As in previous case, if  $j_{\mathbf{true}}$  (resp.  $j_{\mathbf{false}}$ ) are undefined we put  $j_{\mathbf{true}} = 1$  (resp.  $j_{\mathbf{false}} = 1$ ). We put  $i \xrightarrow{\mathbf{true}} j_{\mathbf{true}}$  and  $i \xrightarrow{\mathbf{false}} j_{\mathbf{false}}$ . This completes the definition of CFG for  $\Delta$ .

For every action  $\alpha_i$  of  $\Delta$  and every valuation  $\rho$  we put the following transitions to  $\delta$ .

- (1)  $\alpha_i$  is  $v := Expr$ : put  $\langle \langle i, \rho \rangle, skip, \langle j, \rho[v \leftarrow \llbracket Expr \rrbracket_{\rho}] \rangle \rangle$ , where  $i \rightarrow j$ ;
- (2)  $\alpha_i$  is **guess**  $v \in Expr$  **with**  $Test$ : for every every  $e \in \llbracket Expr \rrbracket_{\rho}$  such that  $\llbracket Test \rrbracket_{\rho[v \leftarrow e]}$  is true put  $\langle \langle i, \rho \rangle, skip, \langle j, \rho[v \leftarrow e] \rangle \rangle$ , where  $i \rightarrow j$ ;
- (3)  $\alpha_i$  is  $\mathbf{inc}(\vec{f}(Expr))$ : let  $e = \llbracket Expr \rrbracket_{\rho}$ . Put  $\langle \langle i, \rho \rangle, inc(c_{f(e)}), \langle j, \rho \rangle \rangle$ , where  $i \rightarrow j$ ;
- (4)  $\alpha_i$  is  $\mathbf{dec}(\vec{f}(Expr))$ : let  $e = \llbracket Expr \rrbracket_{\rho}$ . Put  $\langle \langle i, \rho \rangle, dec(c_{f(e)}), \langle j, \rho \rangle \rangle$ , where  $i \rightarrow j$ ;
- (5)  $\alpha_i$  is **Reject**: put  $\langle \langle i, \rho \rangle, skip, \langle i, \rho \rangle \rangle$ ;
- (6)  $\alpha_i$  is **if**  $Test$  **then**  $\alpha_{i_{\mathbf{true}}}^*$  **else**  $\alpha_{i_{\mathbf{false}}}^*$  **endif**: if  $\llbracket Test \rrbracket_{\rho}$  is true then put  $\langle \langle i, \rho \rangle, skip, \langle i_{\mathbf{true}}, \rho \rangle \rangle$ , otherwise put  $\langle \langle i, \rho \rangle, skip, \langle i_{\mathbf{false}}, \rho \rangle \rangle$ , where  $i \xrightarrow{\mathbf{true}} j_{\mathbf{true}}$  and  $i \xrightarrow{\mathbf{false}} j_{\mathbf{false}}$ ;

- (7)  $\alpha_i$  is **if**  $Test$  **then**  $\alpha_{i_{\text{true}}}^*$  **endif** or  $\alpha_i$  is **while**  $Test$  **do**  $\alpha_{i_{\text{true}}}^*$  **endwhile**: if  $\llbracket Test \rrbracket_\rho$  is true then put  $\langle\langle i, \rho \rangle, skip, \langle i_{\text{true}}, \rho \rangle\rangle$ , otherwise put  $\langle\langle i, \rho \rangle, skip, \langle j, \rho \rangle\rangle$ , where  $i \xrightarrow{\text{true}} j_{\text{true}}$  and  $i \xrightarrow{\text{false}} j_{\text{false}}$ .

Transitions defined above correspond to semantics of actions from Figures 1 and 2. The only exception is for **Reject**, which does not have any computable effect. Instead, it is modelled as a transition of  $M_H$  that loops, and therefore cannot lead to any accepting configuration.

For a counter valuation  $\vartheta$  define  $vectorise(\vartheta)$  as vector  $\vec{n}_\vartheta$  indexed by the set  $\{c_{\vec{f}(e)} \mid \vec{f} \in \text{Vec}_{\mathbb{N}} \text{ and } e \in \text{Dom}(\vec{f})\}$  such that  $\vec{n}_\vartheta(c_{\vec{f}(e)}) = \vartheta(\vec{f})(e)$ .

The statement to be proven is: for every sequence of action  $\alpha^* \subseteq \Delta$  starting in a line  $i$  and finishing in a line  $j$ , and every configurations  $\langle\rho, \vartheta\rangle$  and  $\langle\rho', \vartheta'\rangle$  of HMCA  $H$  we have  $(\langle\rho, \vartheta\rangle, \alpha^*) \rightsquigarrow \langle\rho', \vartheta'\rangle$  if and only if MCA  $M_H$  transits from configuration  $\langle\langle i, \rho \rangle, \vec{n}_\vartheta\rangle$  to  $\langle\langle j, \rho' \rangle, \vec{n}_{\vartheta'}\rangle$ . The proof proceeds by a standard induction on the structure of  $\alpha^*$ .

From the inductive statement we conclude that  $(\langle\rho_I, \vartheta_I\rangle, \Delta) \rightsquigarrow \langle\rho_F, \vartheta_F\rangle$  if and only if  $\langle\langle 1, \rho_I \rangle, \vec{n}_{\vartheta_I}\rangle$  transits to  $\langle\langle 1, \rho_F \rangle, \vec{n}_{\vartheta_F}\rangle$ . Here  $\langle\rho_I, \vartheta_I\rangle$  is the starting configuration of  $H$  and  $\langle\rho_F, \vartheta_F\rangle$  is an accepting configuration of  $H$ . I. e.,  $\rho_I$  is the starting state of  $H$ ,  $\vartheta_I(c_{\vec{f}(e)}) = \vec{0}$  for all  $\vec{f} \in \text{Vec}_{\mathbb{N}}$  and  $e \in \text{Type}(\vec{f})$ ,  $\rho_F \in P_F$  and  $\vartheta_F$  is any counter valuation satisfying  $\vartheta_F(c_{\vec{f}(e)}) = 0$  for every  $\langle\vec{f}, e\rangle \in E$ . Therefore existence of an accepting run of HMCA  $H$  is equivalent to existence of an accepting run of MCA  $M_H$ .  $\square$

**6.7. Global consistency.** In this section we check global consistency of a formula by checking for a given (locally consistent with the formula) frame  $\mathcal{F} = \langle K, \text{ST}, \Xi, \Sigma, \vec{f} \rangle$  whether there exists a normal structure that fits  $\mathcal{F}$ . Intuitively, this is like solving a jigsaw puzzle: the frame gives us a set of shapes (star types) and we have to decide if it is possible to put pieces of these shapes together to build a complete picture (a structure). Here a piece of a given shape has some number of tabs (essential types) of two kinds: tabs corresponding to invertible essential types must be connected to a matching tab in a matching piece; tabs corresponding to non-invertible essential types must be simply connected to a matching piece.

Figure 4 shows a high-level multcounter automaton  $H_{\mathcal{F}}$  that solves this problem. The automaton guesses one by one the sequence of elements of the structure as they appear in the order  $<$ . In each iteration of the transition  $\Delta$  one element of the structure is guessed.

During the construction we have to make sure that several conditions are satisfied. One of them is that all royal types from  $K$  are used exactly once. For this reason we keep track of the set of nodes visited (i. e., guessed) so far; their 1-types are stored in variable  $Visited$ . The variable is updated in line 4 and used in line 30 and in the acceptance condition. Another condition is that the constructed structure is normal, in particular it satisfies Condition 2 in Definition 6.2. Therefore every time the automaton guesses (the star type of) a new element in line 30, it guesses it from the set

$$\begin{aligned} Allowed(Visited) = & \{\sigma \in \text{ST} \mid \pi(\sigma) \in K \setminus Visited\} \cup \\ & \{\sigma \in \text{ST} \mid \forall \pi \in Visited (\pi \notin K \Rightarrow \exists \tau \in \Xi. \text{tp}_1(\tau) = \pi \wedge (x < y) \in \tau \wedge \text{tp}_2(\tau) = \pi(\sigma))\}. \end{aligned}$$

This way we formally forbid guessing a royal type more than once or violating Condition 2 in Definition 6.2. Note that here  $Allowed : 2^{\text{partial}(\text{ST})} \rightarrow 2^{\text{partial}(\text{ST})}$  is a unary operator in the language of HMCA.

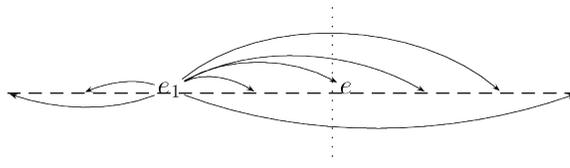


Figure 3: Difference type of  $e_1$  w.r.t.  $e$  contains the arrows that cross the dotted line.

Intuitively, during solving our jigsaw puzzle, we have to represent somehow the border of the partial picture constructed so far. This is stored in the vector  $\overrightarrow{\text{Cut}}$ . It is indexed with partial shapes (formally, partial star types) because we are not interested in parts of pieces that are already connected to the picture; we are only interested in parts of pieces that belong to the border. The cut vector says for each partial shape how many pieces of this shape belong to the border. For technical reasons this vector also counts empty shapes (which correspond to pieces that are fully connected to the picture); these counters are then ignored in the acceptance condition of the constructed automaton.

Formally, we define a difference type of a node w.r.t. to another node in a structure. In the analogy with jigsaw puzzle, the difference type of  $e_1$  w.r.t.  $e$  describes the tabs of the piece  $e_1$  that belong to the border of the constructed picture just before adding the piece  $e$ . Figure 3 shows an example of a difference type.

**Definition 6.13** (Difference type of  $e'$  w.r.t.  $e$  in a structure  $\mathcal{A}$ ). Let  $\mathcal{A} \in \mathcal{O}(\Sigma, <, \#)$  and  $e', e \in \mathcal{A}$  be elements satisfying  $e' <^{\mathcal{A}} e$ . Let  $\sigma$  be the star type of  $e'$  and let  $\{\tau_i\}_{i=1}^k$  be essential types emitted from  $e'$  and accepted by nodes  $<^{\mathcal{A}}$  than  $e$ . The partial star type  $\sigma \setminus \{\tau_i\}_{i=1}^k$  is called the *difference type of  $e'$  w.r.t.  $e$  in  $\mathcal{A}$* .

**Definition 6.14.** Let  $\mathcal{A} \in \mathcal{O}(\Sigma, <, \#)$  and  $e \in \mathcal{A}$ . The *cut* at point  $e$  in  $\mathcal{A}$  is the vector  $\overrightarrow{\text{Cut}}_e$  of natural numbers indexed by star types from  $\text{partial}(\text{ST})$  such that

$$\overrightarrow{\text{Cut}}_e[\sigma] = |\{e_1 \in \mathcal{A} \mid \text{difference type of } e_1 \text{ w.r.t. } e \text{ in } \mathcal{A} \text{ is } \sigma\}|.$$

Final states of the automaton are states where all kings from  $K$  are visited. Intuitively, final counter valuations are valuations where the border of the picture is empty, i. e., the counters corresponding to non-empty shapes must be zeroed. Note that the remaining counters, corresponding to empty shapes, store the numbers of internal pieces of the picture with that particular shape (the partial shapes of these pieces are empty because all tabs are already connected to other pieces) and contain non-zero values.

Intuitively, one iteration of the transition  $\Delta$  works as follows. Before the transition starts we take a piece  $e$  of shape  $\sigma_e$ ; at the end of the iteration the piece is connected to the picture constructed so far and the shape of the next piece is guessed. The new border of the picture is computed in two loops in lines 6–24. Before these loops, in line 5, we compute the tabs of  $e$  that must be connected to the picture. In the loop in lines 6–16 we take one by one these tabs  $\tau$  and connect them to the picture: first the shape  $\sigma_u$  of a matching piece  $e'$  in the picture is guessed and then the border is updated: in line 14 the shape of  $e'$  is removed from the border and in line 15 the updated shape is added to the new border. In order to

**Initial Configuration** Initial state is  $\rho_I$  such that  $\rho_I(\sigma_c) = \sigma_{first}$ ,  $\rho_I(Visited) = \emptyset$  and the value of  $\rho_I$  on remaining variables is arbitrary (but fixed). Initial counter valuation assigns 0 to all counters.

**Accepting Configurations** The set of accepting states  $P_F$  consists of all states  $\rho_F$  satisfying  $K \subseteq \rho_F(Visited)$ . Accepting counter valuations are defined by the set  $E = \{\langle \overrightarrow{Cut}, \sigma \rangle \mid \sigma \in partial(ST) \text{ is non-empty}\}$ .

**Transition  $\Delta$**

```

1: if  $\sigma_c = \perp$  then
2:   Reject
3: endif
4:  $Visited := Visited \cup \{\pi(\sigma_c)\}$ 
5:  $\sigma := \sigma_c^{<}$ 
6: while  $\sigma \neq \emptyset$  do
7:    $\tau := choose(\sigma)$ 
8:    $\sigma := \sigma - \tau$ 
9:   if  $\tau$  is an invertible essential type then
10:    guess  $\sigma_u \in partial(ST)$  with  $\tau^{-1} \in \sigma_u^{>}$ 
11:    else
12:     guess  $\sigma_u \in partial(ST)$  with  $\pi(\sigma_u) = tp_2(\tau)$ 
13:    endif
14:    dec( $\overrightarrow{Cut}[\sigma_u^{>}]$ )
15:    inc( $\overrightarrow{Processed}[\sigma_u^{>} - \tau^{-1}]$ )
16:  endwhile
17: guess  $anotherIteration \in \{\mathbf{true}, \mathbf{false}\}$ 
18: while  $anotherIteration$  do
19:   guess  $\sigma_g \in partial(ST)$ 
20:   guess  $\tau \in \sigma_g^{>}$  with  $tp_2(\tau) = \pi(\sigma_c)$  and ( $\tau$  is non-invertible essential type)
21:   dec( $\overrightarrow{Cut}[\sigma_g^{>}]$ )
22:   inc( $\overrightarrow{Processed}[\sigma_g^{>} - \tau]$ )
23:   guess  $anotherIteration \in \{\mathbf{true}, \mathbf{false}\}$ 
24: endwhile
25: inc( $\overrightarrow{Cut}[\sigma_c^{>}]$ )
26:  $\tau_{\#} := \tau_{\#}(\sigma_c)$ 
27: if  $\tau_{\#} = \perp$  then
28:    $\sigma_c := \perp$ 
29: else
30:   guess  $\sigma_c \in Allowed(Visited)$  with  $(\tau_{\#})^{-1} \in \sigma_c$ 
31: endif
32: guess  $anotherIteration \in \{\mathbf{true}, \mathbf{false}\}$ 
33: while  $anotherIteration$  do
34:   guess  $\sigma_g \in partial(ST)$ 
35:   dec( $\overrightarrow{Processed}[\sigma_g^{>}]$ )
36:   inc( $\overrightarrow{Cut}[\sigma_g^{>}]$ )
37:   guess  $anotherIteration \in \{\mathbf{true}, \mathbf{false}\}$ 
38: endwhile

```

Figure 4: A high-level multicounter automaton  $H_{\mathcal{F}}$  corresponding to a frame  $\mathcal{F}$ .

avoid confusion between the old and the new border, the new one is stored in a separate vector Processed. The loop in lines 18–24 connects in a similar way the tabs of the second kind (non-invertible essential types) from the picture to the piece  $e$ . Then remaining tabs of  $e$  are added to the border in line 25. Finally, the shape of the next piece is guessed and the new border is added to the old one in lines 32–38.

Formally, the set of finite-domain variables of the automaton  $H_{\mathcal{F}}$  is  $\{\sigma_c, \text{Visited}, \sigma, \tau, \sigma_u, \sigma_g, \tau_{\dashv}, \text{anotherIteration}\}$  and there are two arrays of counters Cut and Processed. The Type function is defined as follows. The type of  $\sigma_c$  is  $\text{ST} \cup \{\perp\}$  (that is, formally,  $\text{Type}(\sigma_c) = \text{ST} \cup \{\perp\}$ ); variables  $\sigma$ ,  $\sigma_u$  and  $\sigma_g$  are of type  $\text{partial}(\text{ST})$ ; variable  $\tau$  has type  $\tau(\text{ST})$  and  $\tau_{\dashv}$  has type  $\tau(\text{ST}) \cup \{\perp\}$ . The type of  $\text{Visited}$  is  $2^{\pi(\text{ST})}$  (that is, this variable ranges over sets of 1-types). Finally, the type of vectors of counters Cut and Processed is  $\text{partial}(\text{ST}) \rightarrow \mathbb{N}$ .

For a star type  $\sigma$  define  $\sigma^< = \{\tau \in \sigma \mid (y < x) \in \tau\}$  and  $\sigma^> = \{\tau \in \sigma \mid (x < y) \in \tau\}$ . Intuitively  $\sigma^<$  (resp.  $\sigma^>$ ) denotes the set of tabs that are connected to (resp. remain in the border of) the picture when adding a piece of shape  $\sigma$ . For a star type  $\sigma$  from  $\text{ST}$  define  $\tau_{\dashv}(\sigma)$  as the only 2-type  $\tau$  such that  $\dashv(x, y) \in \tau$  (intuitively, this is the tab that connects a piece of shape  $\sigma$  to the next piece in the picture), or the special value  $\perp$  if  $\sigma$  is the star type of the greatest element in a structure. We also define  $\text{choose}(\sigma)$  in line 7 to be an arbitrary 2-type  $\tau$  such that  $\tau \in \sigma$ .

In each iteration of the transition  $\Delta$  the automaton guesses one element  $e$  of the structure and assigns star type  $\sigma_c$  to it. Intuitively, at the beginning of an iteration the Cut vector stores the cut at  $e$ . Technically, it is the sum of Cut and Processed that stores the cut, but we may assume that the Processed vector is zeroed (see Lemma 6.17 below). The Processed vector stores the information about changes in cuts between the current and the next element. The value of Cut is updated in two loops in lines 6–24. During the computation some counters from Cut are decremented, and some counters from Processed — incremented. Decrementation of a counter corresponds to establishing a 2-type between  $e$  and some  $e'$  smaller than  $e$  (this is done in the loop in lines 6–16), or between  $e'$  and  $e$  (loop in lines 18–24). In order not to establish multiple 2-types between the same pair of nodes, we remove the difference type of  $e'$  w.r.t.  $e$  from Cut and store it in Processed. When the second loop (lines 18–24) finishes its execution the initial value of Cut vector for next node is the sum of the values of updated vectors Cut and Processed in line 24.

In lines 26–31 we guess the star type of the next node, or the special value  $\perp$  in case we are already at the largest node of the structure. Finally, the loop in lines 32–38 may move the content of vector Processed to Cut. We may assume (by Lemma 6.17) that the entire content is actually moved. Formally, the correspondence between a frame  $\mathcal{F}$  and HMCA  $H_{\mathcal{F}}$  is captured by the following proposition.

**Proposition 6.15.** Let  $\mathcal{F} = \langle K, \text{ST}, \Xi, \Sigma, \bar{f} \rangle$  be a frame. The automaton  $H_{\mathcal{F}}$  is non-empty if and only if there exists a structure  $\mathcal{M} \in \mathcal{O}(\Sigma, <, \dashv)$  that fits  $\mathcal{F}$ .

To prove Proposition 6.15, we first show (in Lemma 6.16) that if a structure fits a frame then the frame induces a non-empty HMCA. Then (in Lemma 6.19) we show that if there exists a frame for which the induced HMCA is non-empty, then we can construct a structure that fits the frame.

**Lemma 6.16.** Let  $\mathcal{F} = \langle K, \text{ST}, \Xi, \Sigma, \bar{f} \rangle$  be a frame. If there exists a normal structure  $\mathcal{M} \in \mathcal{O}(\Sigma, <, \dashv)$  that fits  $\mathcal{F}$  then  $H_{\mathcal{F}}$  is non-empty.

*Proof.* We will show that  $H_{\mathcal{F}}$  is non-empty by presenting its accepting run. Let  $e_1, \dots, e_k$  be nodes of  $\mathcal{M}$  sequenced in order  $<^{\mathcal{M}}$ . Define  $Visited_i = \{\text{tp}^{\mathcal{M}}(e_j) \mid j < i\}$ , for  $i \in \underline{k}$ . Recall that  $\overrightarrow{\text{Cut}}_{e_i}$  is the cut at point  $e_i$  in  $\mathcal{A}$ . The accepting run of  $H_{\mathcal{F}}$  is

$$\langle \rho_1, \vartheta_1 \rangle, \dots, \langle \rho_k, \vartheta_k \rangle, \langle \rho_{k+1}, \vartheta_{k+1} \rangle$$

where, for  $i \in \underline{k}$ ,  $\rho_i(\sigma_c) = \text{st}^{\mathcal{A}}(e_i)$ ,  $\rho_i(Visited) = Visited_i$  and the value of  $\rho_i$  on other variables is arbitrary. Moreover  $\vartheta_i(\overrightarrow{\text{Cut}}) = \overrightarrow{\text{Cut}}_{e_i}$  and  $\vartheta_i(\overrightarrow{\text{Processed}}) = \vec{0}$ .

Define the last configuration of the above sequence, i. e.,  $\langle \rho_{k+1}, \vartheta_{k+1} \rangle$ , as  $\rho_{k+1}(\sigma_c) = \perp$ ,  $\rho_{k+1}(Visited) = \{\text{tp}^{\mathcal{M}}(e_j) \mid j \leq k\}$ ,  $\vartheta_{k+1}(\overrightarrow{\text{Processed}}) = \vec{0}$  and  $\vartheta_{k+1}(\overrightarrow{\text{Cut}}) = \overrightarrow{\text{Cut}}_{\text{Acc}}$ , where

$$\overrightarrow{\text{Cut}}_{\text{Acc}}[\sigma] = \begin{cases} |\{e_i \mid 1 \leq i \leq k \text{ and } \text{tp}^{\mathcal{M}}(e_i) = \pi(\sigma)\}| & \text{if } \sigma \text{ is empty} \\ 0 & \text{otherwise.} \end{cases}$$

By inspecting Figure 4 we may observe that  $\langle \rho_{k+1}, \vartheta_{k+1} \rangle$  is indeed an accepting configuration of  $H_{\mathcal{F}}$ . Next, we show that  $\langle \rho_1, \vartheta_1 \rangle$  is the initial configuration of  $H_{\mathcal{F}}$ . Indeed, observe that  $\sigma_{\text{first}}$  is  $\text{st}^{\mathcal{A}}(e_1)$ , and  $Visited_1 = \emptyset$ . Moreover,  $\overrightarrow{\text{Cut}}_{e_1} = \vec{0}$ .

To show that the specified run is indeed accepting, assume that MCA  $H_{\mathcal{F}}$  is in state  $\langle \rho_i, \vartheta_i \rangle$ , where  $i \in \underline{k}$ . We will show that it may transit to state  $\langle \rho_{i+1}, \vartheta_{i+1} \rangle$ .

Clearly, star type  $\text{st}^{\mathcal{M}}(e_i)$  is different from the special value  $\perp$ . Thus the test in line 1 fails and computation does not reject in line 2. Then, in line 4 the set of visited 1-types is updated accordingly. This way we obtain  $Visited_{i+1}$ . Next, in lines 6–24 the value of variable  $\overrightarrow{\text{Cut}}$  is transformed from  $\vartheta_i(\overrightarrow{\text{Cut}})$  to  $\vartheta_{i+1}(\overrightarrow{\text{Cut}})$  in two loops. Now we explain both loops in more detail.

The loop in lines 6–16 is responsible for verifying satisfaction of requirements imposed by  $(\text{st}^{\mathcal{M}}(e_i))^<$ , i. e., for checking that essential 2-types that  $e_i$  wants to emit to smaller nodes can be accepted. The loop handles each 2-type  $\tau \in (\text{st}^{\mathcal{M}}(e_i))^<$  in a single iteration. Let  $e'$  be the unique element of  $\mathcal{M}$  such that  $\text{tp}^{\mathcal{M}}(e_i, e') = \tau$ . There are two cases. First, if  $\tau$  is an invertible essential type then the star type  $\text{st}^{\mathcal{M}}(e')$  contains  $\tau^{-1}$ . Let  $\sigma_u$  be the difference type of  $e'$  w.r.t.  $e_i$  in  $\mathcal{M}$ . Since  $e' <^{\mathcal{M}} e_i$ , the star type  $\sigma_u$  also contains  $\tau^{-1}$ . Thus  $\sigma_u$  can be guessed in line 10. The second case is when  $\tau$  is a non-invertible essential type. Then no information about connection with  $e_i$  is stored in  $\text{st}^{\mathcal{M}}(e')$ . In line 12 we guess  $\sigma_u$ , the difference type of  $e'$  w.r.t.  $e_i$  in  $\mathcal{M}$ . In both cases, by definition of  $\sigma_u$  we have  $\overrightarrow{\text{Cut}}_{e_i}[\sigma_u] > 0$ . Thus decrementing  $\overrightarrow{\text{Cut}}[\sigma_u]$  in line 14 will not reject. To remember that some edges emitted by  $e'$  must be accepted by nodes greater than  $e_i$ , we then increment  $\overrightarrow{\text{Processed}}[\sigma_u - \tau^{-1}]$ . Note that when  $\tau$  is a non-invertible essential type then  $\sigma_u - \tau^{-1} = \sigma_u$ .

The loop in lines 18–24 is responsible for updating the arrays  $\overrightarrow{\text{Cut}}$  and  $\overrightarrow{\text{Processed}}$  to reflect connections of nodes  $e'$  smaller than  $e_i$  to  $e_i$  by non-invertible essential types. In each iteration the transition guesses a difference type of some node  $e'$  w.r.t.  $e_i$  in  $\mathcal{M}$  and assigns it to  $\sigma_g$ . Then it guesses a 2-type  $\tau$  that  $e'$  wants to emit to  $e_i$  (lines 19 and 20). To reflect that the connection indeed exists,  $\overrightarrow{\text{Cut}}[\sigma_g]$  is decremented, type  $\tau$  removed from  $\sigma_g$  and vector  $\overrightarrow{\text{Processed}}[\sigma_g - \tau]$  is incremented in lines 21 and 22. Finally, when the loop is over we increment  $\overrightarrow{\text{Cut}}[\sigma_c^>]$  to reflect that the edges emitted by  $e_i$  to greater nodes have to be matched later on, provided that  $i < k$ . When  $i = k$ , node  $e_i$  is the last node of the structure, and it must accept all essential types not accepted by smaller nodes.

In lines 26–31 we guess the star type of the next node, or the special value  $\perp$  in case  $i = k$ . The next value of  $\sigma_c$  must be such that it accepts invertible essential type  $\tau_{\perp}$  emitted by  $e_i$ . Note that  $\perp(x, y) \in \tau_{\perp}$ , which verifies that  $e_{i+1}$  is indeed the successor of  $e_i$  w.r.t.  $<^{\mathcal{M}}$ . Moreover,  $\text{st}^{\mathcal{M}}(e_{i+1}) \in \text{Allowed}(\text{Visited}_{i+1})$  as either  $e_{i+1}$  is a king in  $\mathcal{M}$  or it is not a king and can be connected to every smaller non-royal node by a silent 2-type. The latter property holds because  $\mathcal{M}$  is normal. Thus  $\text{st}^{\mathcal{M}}(e_{i+1})$  can be guessed as the new  $\sigma_c$ .

Finally, the loop in lines 32–38 may move the content of vector  $\overrightarrow{\text{Processed}}$  to  $\overrightarrow{\text{Cut}}$ . We may assume (see Lemma 6.17 below) that the entire content is actually moved. Then, vector  $\overrightarrow{\text{Cut}}$  obtains the value of  $\overrightarrow{\text{Cut}}_{e_{i+1}}$ .  $\square$

The proof of the other direction, that is the construction of a structure from a run of  $H_{\mathcal{F}}$ , is more complicated and we need some additional lemmas. First, observe that at the end of the transition of  $H_{\mathcal{F}}$  (lines 32–38) there is a loop that non-deterministically chooses a partial star-type  $\sigma_g$ , decrements  $\overrightarrow{\text{Processed}}[\sigma_g]$  and increments  $\overrightarrow{\text{Cut}}[\sigma_g]$ . W.l.o.g. we may assume that this procedure erases the entire vector  $\overrightarrow{\text{Processed}}$ , and thus  $\overrightarrow{\text{Processed}}_i = \vec{0}$  for all  $i \in \{1, \dots, k\}$ . Thus we have the following lemma.

**Lemma 6.17.** *Let  $\mathcal{F}$  be a frame such that multicounter automaton  $H_{\mathcal{F}}$  is non-empty. Then there exists an accepting run  $\langle \rho_1, \vartheta_1 \rangle, \dots, \langle \rho_k, \vartheta_k \rangle$  of  $H_{\mathcal{F}}$  such that  $\vartheta_i(\overrightarrow{\text{Processed}}) = \vec{0}$  for every  $i \in \underline{k}$ .*

*Proof.* Assume that  $H_{\mathcal{F}}$  is non-empty and let  $\langle \rho'_1, \vartheta'_1 \rangle, \dots, \langle \rho'_k, \vartheta'_k \rangle$  be its accepting run. For  $i \in \underline{k}$  let  $\rho_i = \rho'_i$ , and  $\vartheta_i(\overrightarrow{\text{Cut}}) = \vartheta'_i(\overrightarrow{\text{Cut}}) + \vartheta'_i(\overrightarrow{\text{Processed}})$ ,  $\vartheta_i(\overrightarrow{\text{Processed}}) = \vec{0}$ . We now argue that the run  $\langle \rho_1, \vartheta_1 \rangle, \dots, \langle \rho_k, \vartheta_k \rangle$  is accepting. Consider a configuration  $\langle \rho'_i, \vartheta'_i \rangle$ , for  $i \in k-1$ . By assumption  $H_{\mathcal{F}}$  may transit from  $\langle \rho'_i, \vartheta'_i \rangle$  to  $\langle \rho'_{i+1}, \vartheta'_{i+1} \rangle$ . We will show that  $H_{\mathcal{F}}$  may transit from  $\langle \rho_i, \vartheta_i \rangle$  to  $\langle \rho_{i+1}, \vartheta_{i+1} \rangle$ . Execution of  $H_{\mathcal{F}}$ 's transition can be broken into two parts. First, lines 1–31 and second, lines 32–38.

Execution of the first part on configuration  $\langle \rho'_i, \vartheta'_i \rangle$  may be mimicked on configuration  $\langle \rho_i, \vartheta_i \rangle$  for two reasons. First, the execution doesn't depend on value of vector  $\overrightarrow{\text{Processed}}$  (i.e., the first part of the execution does not use decreasing actions on  $\overrightarrow{\text{Processed}}$ ). Second we have  $\vartheta_i(\overrightarrow{\text{Cut}}) \geq \vartheta'_i(\overrightarrow{\text{Cut}})$ , and since the execution does not reject on smaller values of  $\overrightarrow{\text{Cut}}$ , it won't reject on higher values of  $\overrightarrow{\text{Cut}}$ .

In the second part of the execution (lines 32–38) we may conduct non-deterministic guesses in such a way that the entire vector  $\overrightarrow{\text{Processed}}$  is reset. Thus  $H_{\mathcal{F}}$  may transit from  $\langle \rho_i, \vartheta_i \rangle$  to  $\langle \rho_{i+1}, \vartheta_{i+1} \rangle$ . Since the selection of  $i$  was arbitrary, we conclude that all transitions in sequence  $\langle \rho_1, \vartheta_1 \rangle, \dots, \langle \rho_k, \vartheta_k \rangle$  can be conducted. Since  $\langle \rho'_1, \vartheta'_1 \rangle$  is the starting configuration of  $H_{\mathcal{F}}$ , we have  $\vartheta'_1(\overrightarrow{\text{Processed}}) = \vec{0}$ . Furthermore,  $\vartheta_1(\overrightarrow{\text{Cut}}) = \vartheta'_1(\overrightarrow{\text{Cut}})$ ,  $\vartheta_1(\overrightarrow{\text{Processed}}) = \vec{0}$  and  $\rho_1 = \rho'_1$ . Therefore  $\langle \rho_1, \vartheta_1 \rangle = \langle \rho'_1, \vartheta'_1 \rangle$  and  $\langle \rho_1, \vartheta_1 \rangle$  is the starting state of  $H_{\mathcal{F}}$ . Moreover, as  $\langle \rho'_k, \vartheta'_k \rangle$  is an accepting state, we have  $\vartheta'_k(\overrightarrow{\text{Processed}}) = \vec{0}$ , which leads to a conclusion that  $\langle \rho_k, \vartheta_k \rangle$  is also an accepting state. Therefore the specified run is accepting, and it satisfies requirements imposed by the lemma.  $\square$

By a *partial structure* we mean a (usually not complete) graph whose nodes are labelled with 1-types and edges are labelled with 2-types such that whenever an edge  $\langle e_1, e_2 \rangle$  is labelled with  $\tau$  then  $e_1$  is labelled with  $\text{tp}_1(\tau)$  and  $e_2$  is labelled with  $\text{tp}_2(\tau)$ . Intuitively, partial structures are graphs that can be extended to (full) relational structures by adding some edges. In the following lemma we construct a partial structure from a run of the

automaton  $H_{\mathcal{F}}$ . Each iteration of the transition  $\Delta$  corresponds to an extension of the structure constructed so far with edges connecting the element guessed in the iteration. Here we are interested only in edges that are labelled with essential types.

**Lemma 6.18.** *Let  $\langle \rho_1, \vartheta_1 \rangle, \dots, \langle \rho_k, \vartheta_k \rangle, \langle \rho_{k+1}, \vartheta_{k+1} \rangle$  be an accepting run of  $H_{\mathcal{F}}$  such that  $\vartheta_i(\overrightarrow{\text{Processed}}) = \vec{0}$  for every  $i \in \underline{k+1}$ . There exists a partial structure  $\mathcal{E}$  with nodes  $e_1, \dots, e_k$  such that*

- $\vartheta_i(\overrightarrow{\text{Cut}})$  is the cut at point  $e_i$  in  $\mathcal{E}$ , for  $i \in \underline{k}$ ,
- the star type of  $e_i$  in  $\mathcal{E}$  (i. e.,  $\text{st}^{\mathcal{E}}(e_i)$ ) is  $\rho_i(\sigma_c)$  for  $i \in \underline{k}$ , and
- $\mathcal{E} \models \#(e_i, e_{i+1})$ , for  $i \in \underline{k-1}$ .

*Proof.* We say that a node  $e_i$  *wants* to emit an edge of type  $\tau$  if  $\tau \in \rho_i(\sigma_c)$  and in the partial structure being constructed no edge  $\tau$  emitted from  $e_i$  is yet present. We will construct the partial structure  $\mathcal{E}$  in  $k$  steps. We start with an empty partial structure  $\mathcal{E}_0$ . At the beginning of each step  $i$ , for  $i \in \underline{k}$ , we have a partial structure  $\mathcal{E}_{i-1}$  with nodes  $e_1, \dots, e_{i-1}$  and some essential types established between these nodes. In step  $i$  we extend  $\mathcal{E}_{i-1}$  with node  $e_i$  to form  $\mathcal{E}_i$ . During the construction we will maintain the following invariants:

- During step  $i$  all essential types  $\tau_{pctr}$  emitted from  $e_i$  to nodes  $e_j$ , where  $j < i$ , are established. These essential types are defined by  $(\rho_i(\sigma_c))^{<}$ .
- For any essential type  $\tau_{brdr}$ , just before step  $i$ , vector  $\vartheta_i(\overrightarrow{\text{Cut}})$  captures the number of edges  $\tau_{brdr}$  that “want” to be emitted from nodes  $e_j$ , where  $j < i$ , and accepted by nodes  $e_{j'}$ , where  $j' \geq i$ . The number is  $\sum_{\{\sigma \mid \tau_{brdr} \in \sigma\}} \left( \vartheta_i(\overrightarrow{\text{Cut}}) \right) (\sigma)$ . Some of these edges will be accepted by  $e_i$ , and some by nodes with larger indices.

The first invariant trivially holds before step 1. Since  $\vartheta_1(\overrightarrow{\text{Cut}}) = \vec{0}$ , as  $\langle \rho_1, \vartheta_1 \rangle$  is the starting configuration of  $H_{\mathcal{F}}$ , the second invariant also holds. Now, assuming that both invariants hold and that  $\mathcal{E}_{i-1}$  is constructed, we construct the structure  $\mathcal{E}_i$ . This is done in one transition of  $H_{\mathcal{F}}$ , and after the transition the invariants are preserved.

The loop in lines 6–16 is used for preserving the first invariant, that is, for ensuring that all essential types  $\tau_{pctr} \in (\rho_i(\sigma_c))^{<}$  are established. In the loop each such type  $\tau_{pctr}$  is considered one by one.

There are two cases. If  $\tau_{pctr}$  is an invertible essential type then there must be a node  $e_j$  with  $j < i$ , that wants to emit an edge  $(\tau_{pctr})^{-1}$ . Therefore, there must be a partial star-type  $\sigma_u$  such that  $(\tau_{pctr})^{-1} \in \sigma_u$  and  $\vartheta_i(\overrightarrow{\text{Cut}})(\sigma_u) > 0$ . We guess this type in line 10. In the second case, if  $\tau_{pctr}$  is a non-invertible essential type then this fact is not reflected in the star type of a node that will accept it. Therefore, we only need to make sure that a node  $e_j$ , with  $j < i$  and with 1-type  $\text{tp}_2(\tau_{pctr})$  exists. Again,  $\vartheta_i(\overrightarrow{\text{Cut}})$  contains sufficient data to verify this. In line 12 a star-type  $\sigma_u$  of such a node  $e_j$  is guessed. Next, in both cases, line 14 verifies that the guess is correct and reflects that an edge between  $e_i$  and  $e_j$  is established. The partial star type  $\sigma_u - (\tau_{pctr})^{-1}$  may be non-empty, i. e., there may be other essential types that want to be emitted from  $e_j$ . We store this information by increasing  $\vartheta_i(\overrightarrow{\text{Processed}})(\sigma_u - (\tau_{pctr})^{-1})$  in line 15. Note that we use a separate vector  $\overrightarrow{\text{Processed}}$  as we must avoid assigning more than one essential type between the same pair of nodes  $e_i$  and  $e_j$ . For this reason we index  $\overrightarrow{\text{Cut}}$  and  $\overrightarrow{\text{Processed}}$  by partial star types instead of essential types.

To preserve the second invariant, we look at loops in lines 18–24 and 32–38. The loop in lines 18–24 is used for verifying that some essential types that nodes  $e_j$  (for  $j < i$ ) want

to emit to larger nodes, are accepted by  $e_i$ . Note that the number of iterations made by the loop is not determined, unlike the case of the loop considered previously. In lines 19 and 20 a partial star-type  $\sigma_g$  and an essential type  $\tau_{brdr} \in \sigma_g$  are guessed. Note that  $\tau_{brdr}$  cannot be invertible, as such types were considered in the previous loop. After verifying the correctness of the guess by decrementing an appropriate counter, we store the information about remaining essential type that  $e_j$  wants to emit by incrementing  $\vartheta_i(\overrightarrow{\text{Processed}})(\sigma_g - (\tau_{brdr}))$ .

When the execution reaches line 25, all essential types between  $e_i$  and smaller nodes are established. Since  $\rho_i(\sigma_c)$  is  $(\rho_i(\sigma_c))^{<} \cup (\rho_i(\sigma_c))^{>}$ , node  $e_i$  still wants to emit essential types from  $(\rho_i(\sigma_c))^{>}$ . To reflect this the action in line 25 increases an appropriate counter.

In lines 26–31 we guess the star type of the next node, or the special value  $\perp$  in case  $i = k$ . If  $i < k$  then the next value of  $\sigma_c$  must be such that it accepts invertible essential type  $\tau_{\perp}$  emitted by  $e_i$ . This ensures that nodes  $e_1, \dots, e_k$  are guessed in order.

Finally, the loop in lines 32–38 may move the content of vector  $\overrightarrow{\text{Processed}}$  to  $\overrightarrow{\text{Cut}}$ , and by assumption of the lemma it indeed moves the entire content of  $\overrightarrow{\text{Processed}}$  to  $\overrightarrow{\text{Cut}}$ , ensuring that the second invariant is satisfied just before step  $i + 1$ .

Put  $\mathcal{E} = \mathcal{E}_k$  and consider the partial structure  $\mathcal{E}$ . Since  $\langle \rho_{k+1}, \vartheta_{k+1} \rangle$  is an accepting configuration, we have  $\vartheta_{k+1}(\overrightarrow{\text{Cut}})(\sigma) = 0$ , for every non-empty star type  $\sigma$ . Thus  $\sum_{\{\sigma | \tau_{brdr} \in \sigma\}} (\vartheta_{k+1}(\overrightarrow{\text{Cut}}))(\sigma) = 0$ , for every essential type  $\tau_{brdr}$ . By the second invariant we conclude that for all  $i \in \underline{k}$ , all essential types  $\tau \in (\rho_i(\sigma_c))^{>}$  are emitted from  $e_i$  and accepted by some  $e_j$ , where  $j > i$  and  $j \leq k$ . This means that  $\vartheta_i(\overrightarrow{\text{Cut}})$  is the cut at point  $e_i$  in  $\mathcal{E}$  (first condition of the lemma). As guaranteed by first invariant, in step  $i$  we emitted from  $e_i$  all types  $\tau \in (\rho_i(\sigma_c))^{<}$ . Since  $\rho_i(\sigma_c)$  is  $(\rho_i(\sigma_c))^{<} \cup (\rho_i(\sigma_c))^{>}$ , the node  $e_i$  in  $\mathcal{E}_k$  does not want to emit any essential type. This means that  $e_i$  realises the star type  $\text{st}^{\mathcal{E}}(e_i) = \rho_i(\sigma_c)$  (second condition of the lemma). Finally, as already observed, in lines 26–31 the transition of  $H_{\mathcal{F}}$  ensured that nodes of  $\mathcal{E}$  are guessed in order (third condition of the lemma).  $\square$

In previous lemma we constructed a partial structure with all edges labelled with essential types. This structure is still not a complete graph because edges labelled with silent types are missing. In the following lemma we add these edges.

**Lemma 6.19.** *Let  $\mathcal{F} = \langle K, \text{ST}, \Xi, \Sigma, \bar{f} \rangle$  be a frame. If  $H_{\mathcal{F}}$  is non-empty then there exists a structure  $\mathcal{M} \in \mathcal{O}(\Sigma, <, \perp)$  that fits  $\mathcal{F}$ .*

*Proof.* Assume that  $H_{\mathcal{F}}$  is non-empty. By Lemma 6.17 there exists an accepting run of  $H_{\mathcal{F}}$ :  $\langle \rho_1, \vartheta_1 \rangle, \dots, \langle \rho_k, \vartheta_k \rangle, \langle \rho_{k+1}, \vartheta_{k+1} \rangle$  where  $\vartheta_i(\overrightarrow{\text{Processed}}) = \vec{0}$  for every  $i \in \underline{k+1}$ . By Lemma 6.18 there exists a partial structure  $\mathcal{E}$  whose nodes sequenced in order are  $e_1, \dots, e_k$  and  $\text{st}^{\mathcal{E}}(e_i) = \rho_i(\sigma_c)$ , for every  $i \in \underline{k}$ . Note that  $\text{Type}(\sigma_c) = \text{ST}$  and therefore  $\text{st}^{\mathcal{E}}(e_i) \in \text{ST}$ . Moreover, every royal type from  $K$  is realised in  $\mathcal{E}$ , as the acceptance condition of  $H_{\mathcal{F}}$  requires that  $\rho_{k+1}(\text{Visited})$  contains the set of kings. We now show that we may supplement  $\mathcal{E}$  to a full relational structure  $\mathcal{M} \in \mathcal{O}(\Sigma, <, \perp)$  by silent types from  $\Xi$ . This will be enough to show that  $\mathcal{M}$  is a structure that fits  $\mathcal{F}$ .

Take any pair of elements  $\langle e_i, e_j \rangle$  such that no type is established for this pair in  $\mathcal{E}$ . This means that neither  $e_i$  nor  $e_j$  is a king. W.l.o.g. assume that  $i < j$ . Let  $\pi_i = \text{tp}^{\mathcal{E}}(e_i)$  and  $\pi_j = \text{tp}^{\mathcal{E}}(e_j)$ . Consider the execution of  $H_{\mathcal{F}}$  in step  $j - 1$ . In line 30 of the transition the star type of  $e_j$  is guessed in such a way that all already visited non-royal 1-types can be connected by a silent type from  $\Xi$  to the 1-type  $\pi(\sigma_c)$ , i. e., to  $\pi_j$  (cf. Condition 2 in Definition 6.2). Since  $\pi_i$  is one of such visited types, we may find a type  $\tau \in \Xi$  such that

$\text{tp}_1(\tau) = \pi_i$ ,  $\text{tp}_2(\tau) = \pi_j$  and  $(x < y) \in \tau$ . We then establish the type  $\tau$  between  $e_i$  and  $e_j$ , and we continue the procedure for remaining pairs of nodes. In this way we construct the structure  $\mathcal{M}$ . It is clear from the construction that  $\mathcal{M}$  fits  $\mathcal{F}$  and that  $\mathcal{M} \in \mathcal{O}(\Sigma, <, \#)$ .  $\square$

### 6.8. Main theorem.

**Theorem 6.20.** *The finite satisfiability problem for  $C^2(*, *, \{<, \#\})$  is decidable.*

*Proof.* We may assume that the input  $C^2(*, *, \{<, \#\})$  formula  $\varphi$  is in normal form (otherwise it can be brought to the normal form). A non-deterministic decision procedure for the finite satisfiability problem guesses a frame  $\mathcal{F}$  such that  $\mathcal{F} \models \varphi$  and checks if HMCA  $H_{\mathcal{F}}$  is non-empty. If so, then by Proposition 6.15 we obtain a structure  $\mathcal{M} \in \mathcal{O}(\Sigma, <, \#)$  that fits  $\mathcal{F}$ . Because  $\mathcal{M}$  fits  $\mathcal{F}$  and  $\mathcal{F} \models \varphi$ , by Proposition 6.11 we conclude that  $\mathcal{M} \models \varphi$ . This shows that  $\varphi$  is finitely satisfiable, so our procedure is sound. On the other hand, if  $\varphi$  is finitely satisfiable then, by Lemma 6.5, it has a model  $\mathcal{M}$  which is a normal structure. Again, by Proposition 6.11 there exists a frame  $\mathcal{F}$  such that  $\mathcal{M}$  fits  $\mathcal{F}$  and  $\mathcal{F} \models \varphi$ . By Proposition 6.15 we conclude that  $H_{\mathcal{F}}$  is non-empty, so the procedure is complete.

Note that the size of  $\mathcal{F}$  is at most doubly exponential in the size of the formula's signature  $\langle \Sigma, \bar{f} \rangle$ , so there are finitely many frames that can be guessed. Furthermore, the emptiness problem of HMCA  $H_{\mathcal{F}}$  is decidable, as stated in Proposition 6.12.  $\square$

The decidability result for  $C^2(*, *, \{<\})$  can be applied to decide  $C^2$  with one acyclic relation. Every acyclic relation can be extended to a linear order by topological ordering, and conversely, every relation contained in a linear order is acyclic. Thus we have the following corollary.

**Corollary 6.21.** *The finite satisfiability problem for  $C^2(*, *, \{<\})$ , where  $<$  is interpreted as an acyclic relation, is decidable.*

*Proof.* A formula  $\varphi$  is finitely satisfiable in  $C^2(*, *, \{<\})$  if and only if the formula  $\varphi \wedge \forall x \forall y \ x < y \Rightarrow x < y$  is finitely satisfiable in  $C^2(*, *, \{<\})$ .  $\square$

## 7. CONCLUSION

We have shown several complexity results for finite satisfiability of two-variable logics with counting quantifiers and linear orders. In particular we proved NEXPTIME-completeness of the problem for  $C^2(*, 0, \{<, \#\})$ , VAS-completeness for  $C^2(*, *, \{<\})$  and undecidability for  $C^2(*, 2, \{<_1, <_2\})$ . The results for  $C^2(*, *, \{<\})$  extend to  $C^2$  with one acyclic relation. There are still some unsolved cases, including  $C^2(*, 0, \{<_1, <_2\})$  and  $C^2(*, 1, \{<_1, <_2\})$ .

There are lots of open problems in the area. One of them is general satisfiability. None of the logics considered here has the finite model property. Our techniques rely on finiteness of the underlying structure, so they cannot be directly applied to general satisfiability on possibly infinite structures. Among possible directions for future work one can choose combination of  $C^2$  with other interpreted binary relations like preorders [22] or transitive relations [33]. Another possibility is to consider  $C^2$  with closure operations on some relations, like equivalence closure [15] or deterministic transitive closure [3].

## REFERENCES

- [1] Mikolaj Bojanczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27, 2011.
- [2] Witold Charatonik, Emanuel Kieronski, and Filip Mazowiecki. Satisfiability of the two-variable fragment of first-order logic over trees. *CoRR*, abs/1304.7204, 2013.
- [3] Witold Charatonik, Emanuel Kieroński, and Filip Mazowiecki. Decidability of weak logics with deterministic transitive closure. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, page 29, 2014.
- [4] Witold Charatonik and Piotr Witkowski. Two-variable logic with counting and trees. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 73–82, 2013.
- [5] Witold Charatonik and Piotr Witkowski. Two-variable logic with counting and a linear order. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, pages 631–647, 2015.
- [6] Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.
- [7] Diego Figueira. Satisfiability for two-variable logic with two successor relations on finite linear orders. *Computing Research Repository*, abs/1204.2495, 2012.
- [8] E. Grädel, M. Otto, and E. Rosen. Undecidability results on two-variable logics. *Archive for Mathematical Logic*, 38:213–354, 1999.
- [9] Erich Grädel, Phokion Kolaitis, and Moshe Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.
- [10] Erich Grädel and Martin Otto. On logics with two variables. *Theor. Comput. Sci.*, 224(1-2):73–113, 1999.
- [11] Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *LICS*, pages 306–317. IEEE Computer Society, 1997.
- [12] N. Immerman, A. Rabinovich, T. Reps, M. Sagiv, and G. Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In *Proceedings of the 18th Annual Conference of the European Association for Computer Science Logic (CSL'04)*, pages 160–174, 2004.
- [13] Emanuel Kieroński. Decidability issues for two-variable logics with several linear orders. In Marc Bezem, editor, *CSL*, volume 12 of *LIPICs*, pages 337–351. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.
- [14] Emanuel Kieroński and Jakub Michaliszyn. Two-variable universal logic with transitive closure. In Patrick Cégielski and Arnaud Durand, editors, *CSL*, volume 16 of *LIPICs*, pages 396–410. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.
- [15] Emanuel Kieroński, Jakub Michaliszyn, Ian Pratt-Hartmann, and Lidia Tendera. Two-variable first-order logic with equivalence closure. In *LICS*, pages 431–440. IEEE, 2012.
- [16] Emanuel Kieroński and Martin Otto. Small substructures and decidability issues for first-order logic with two variables. In *LICS*, pages 448–457. IEEE Computer Society, 2005.
- [17] Emanuel Kieroński and Lidia Tendera. On finite satisfiability of two-variable first-order logic with equivalence relations. In *LICS*, pages 123–132. IEEE Computer Society, 2009.
- [18] S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 267–281, 1982.
- [19] Jérôme Leroux. The general vector addition system reachability problem by presburger inductive invariants. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science*, pages 4–13, 2009.
- [20] R. J. Lipton. The reachability problem requires exponential space. 62, New Haven, Connecticut: Yale University, Department of Computer Science, Research, Jan, 1976.
- [21] Amaldev Manuel. Two variables and two successors. In Petr Hlinený and Antonín Kucera, editors, *MFCs*, volume 6281 of *Lecture Notes in Computer Science*, pages 513–524. Springer, 2010.
- [22] Amaldev Manuel and Thomas Zeume. Two-variable logic on 2-dimensional structures. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPICs*, pages 484–499. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2013. Long version of the paper available from the authors.

- [23] Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.
- [24] Michael Mortimer. On languages with two variables. *Mathematical Logic Quarterly*, 21(1):135–140, 1975.
- [25] B. O. Nash. Reachability problems in vector addition systems. *The American Mathematical Monthly*, 80(3):pp. 292–295, 1973.
- [26] Martin Otto. Two variable first-order logic over ordered domains. *J. Symb. Log.*, 66(2):685–702, 2001.
- [27] Leszek Pacholski, Wiesław Szwał, and Lidia Tendera. Complexity of two-variable logic with counting. In *LICS*, pages 318–327. IEEE, 1997.
- [28] Ian Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, 14(3):369–395, 2005.
- [29] Ian Pratt-Hartmann. The two-variable fragment with counting revisited. In Anuj Dawar and Ruy J. G. B. de Queiroz, editors, *WoLLIC*, volume 6188 of *Lecture Notes in Computer Science*, pages 42–54. Springer, 2010.
- [30] Ian Pratt-Hartmann. Logics with counting and equivalence. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, page 76, 2014.
- [31] Thomas Schwentick and Thomas Zeume. Two-variable logic with two order relations - (extended abstract). In Anuj Dawar and Helmut Veith, editors, *CSL*, volume 6247 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 2010.
- [32] Dana Scott. A decision method for validity of sentences in two variables. *Journal of Symbolic Logic*, 27:477, 1962.
- [33] Wiesław Szwał and Lidia Tendera.  $\text{FO}^2$  with one transitive relation is decidable. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 317–328, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.