Algorithmic Properties of Millstream Systems

Suna Bensch, Henrik Björklund, and Frank Drewes

Department of Computing Science, Umeå University
90187 Umeå, Sweden
{suna, henrikb, drewes}@cs.umu.se

Abstract. Millstream systems have recently been proposed as a formalization of the linguistic idea that natural language should be described as a combination of different modules related by interfaces. In this paper we investigate algorithmic properties of Millstream systems having regular tree grammars as modules and MSO logic as interface logic. We focus on the so-called completion problem: Given trees generated by a subset of the modules, can they be completed into a valid configuration of the Millstream system?

1 Introduction

Millstream systems [1] have recently been introduced as a generic mathematical framework for the description of natural language providing the possibility to formalize and reason about the relation between different linguistic levels, such as phonology, morphology, syntax and semantics. Millstream systems are motivated by contemporary linguistic theories that refrain from the idea of transformational grammars in the Chomskian tradition in which linguistic levels are hierarchically ordered. The authors in [11, 7], for example, propose to view them as autonomous modules that work simultaneously but are linked with each other through interfaces that describe the interactions and interdependencies between these linguistic levels. Both authors argue that the human way of processing language is more adequately described by such a non-hierarchical approach, as the human brain seems to store and process different linguistical levels in parallel, at the same time linking them according to certain rules in order to create a whole that is more than the sum of its parts. Matching this view, a Millstream system consists of several individual *modules* specifying tree languages $L_1, \dots, L_k$, and a logical *interface* relating the (trees yielded by the) modules. A configuration is a tuple $(t_1, \dots, t_k) \in L_1 \times \cdots \times L_k$, augmented with links as specified by the interface.

Let us consider an example that illustrates the linguistic ideas that have motivated Millstream systems. Figure 1 shows the syntactic and semantic structure, depicted as trees (a) and (b), respectively, of the sentence *John loves Sarah* and the established interface links which are depicted as dotted lines linking syntactic categories occurring in structure (a) with semantic categories occurring in structure (b). The syntactic tree (a) divides the sentence S into a nominal phrase $\mathrm{NP_S}$ and a verbal phrase VP. The nominal phrase $\mathrm{NP_S}$ consists of the lexical

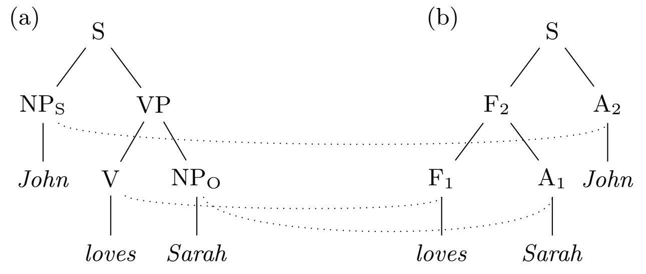

Fig. 1. Syntactical and semantic structures of *John loves Sarah*.

item *John* and the verbal phrase VP is divided into a verb V and a nominal phrase $\mathrm{NP_O}$, where V and $\mathrm{NP_O}$ consist of the lexical items *loves* and *Sarah*, respectively. The semantic tree (b) depicts that the transitive verb *loves* is of category $\mathrm{F_1}$ which represents a function that is applied to the argument $\mathrm{A_1}$ (the object) whose lexical item is *Sarah* and yields a function as its result, namely $\mathrm{F_2}$. Function $\mathrm{F_2}$ in turn is applied to the argument $\mathrm{A_2}$ (the subject) whose lexical item is *John*. Thus, the analysis of the sentence *John loves Sarah* does not only result in its syntactic and semantic trees, but also includes relationships between them, namely the links in Figure 1. The syntactic category V, for example, is linked with the semantic category F_1 which illustrates that these particular occurrences of V and F_1 correspond to each other. The syntactic subject $\mathrm{NP_S}$ and the object $\mathrm{NP_O}$ are linked with the semantic arguments A_2 and A_1, respectively, which reflects that the syntactic arguments (subject and object) of a (transitive) verb correspond to the semantic arguments of the semantic function of that verb. The conditions that such links have to fulfill are described by the interface. The reader is referred to [12] for a discussion of the syntax-semantics interface from the linguistic point of view and to [1] for more a detailed discussion of how this can be formalized in terms of Millstream systems.

In particular, Millstream systems are of interest for natural language processing, and in particular for natural language understanding and natural language generation. Simply put, the task of natural language understanding is to construct a suitable semantic representation of a sentence that has been heard (phonology) and parsed (syntax). Within the framework of Millstream systems this corresponds to the problem where we are given a syntactic tree (and possibly a phonological tree if a phonological module is involved) and the goal is to construct an appropriate semantic tree. Conversely, natural language generation can be seen as the problem to construct an appropriate syntactic (and/or phonological) tree from a given semantic tree.

In abstract terms, the situations just described are identical. In both cases, a Millstream system is given, the input is a partial configuration consisting of some of the required trees, and the goal is to complete the configuration by adding the missing trees and the links between the trees. In this paper, we study whether and how this problem, called the completion problem, can be solved for so-called regular MSO Millstream systems, i. e. systems in which the modules are regular tree grammars (or, equivalently, finite tree automata) and the interface conditions are expressed in monadic second-order (MSO) logic. We prove that the emptiness problem (where no tree of the configuration is known) is undecidable, but the completion problem is decidable if no direct links exist between the unknown trees. Finally, motivated by the observation that the completion problem is decidable if the configurations are of bounded tree width, we establish sufficient conditions under which this is the case. Moreover, structures of bounded tree-width seem to be of particular interest for natural language processing. For example, Kornai and Tuza [8] argue that bounded path-width is related to the bounded capacity of the short-term memory and its influence on human generation and understanding of language.

The rest of this paper is organized as follows. The next section contains the definition of Millstream systems and other basic notions. In Section 3, the undecidability of the emptiness problem is shown. Section 4 contains the proof that the completion problem is decidable in those cases where there are no direct links between unknown parts of the configuration, and in Section 5 sufficient conditions for bounding the tree width of configurations are studied. To obey the page limit, some proofs have been moved into the appendix.

2 Definitions and Preliminaries

The set of natural numbers is denoted by $\mathbb{N}$, and $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. For $k \in \mathbb{N}$, we let $[k] = \{1, \ldots, k\}$. For a set S, the set of all nonempty finite sequences (or strings) over S is denoted by S^+; if the empty sequence ϵ is included, we write S^*.

A *ranked alphabet* is a finite set Σ of pairs (f, k), where f is a symbol and $k \in \mathbb{N}$ is its *rank*. We denote (f, k) by $f^{(k)}$, or simply by f if k is understood.

We define trees over Σ in one of the standard ways, by identifying the nodes of a tree t with sequences of natural numbers. Thus, T_Σ consists of all mappings $t \colon V(t) \to \Sigma$ (called trees), such that the set $V(t)$ of *nodes* of t is a finite and non-empty prefix-closed subset of $\mathbb{N}_+^*$, and, for every node $v \in V(t)$, if $t(v) = f^{(k)}$, then $\{i \in \mathbb{N} \mid vi \in V(t)\} = [k]$. In other words, the *children* of v are $v1, \ldots, vk$ (and v is their *parent*). For $v, u \in V(t)$ we write $v \leq u$ if v is a prefix of u and $v < u$ if v is a proper prefix. A *Σ-tree generator* is any kind of device G that specifies a tree language $L(G) \subseteq \mathrm{T}_\Sigma$. We assume familiarity with generators of regular tree languages, e.g., finite tree automata, regular tree grammars, and monadic second-order logic (MSO) (see, e.g., [6]).

For a tree $t \in \mathrm{T}_\Sigma$, the subtree of t rooted at v (defined in the usual way) is denoted by t/v. We denote a tree t as $f[t_1, \ldots, t_k]$ if $t(\epsilon) = f^{(k)}$ and $t/i = t_i$ for $i \in [k]$, omitting the brackets if $k = 0$.

For a tuple $T \in \mathrm{T}_\Sigma^k$, we let $V(T)$ denote the set $\{(i, v) \mid i \in [k] \text{ and } v \in V(t_i)\}$. Thus, $V(T)$ is the disjoint union of the sets $V(t_i)$. Furthermore, we let $V(T, i)$ denote its ith component, i.e., $V(T, i) = \{i\} \times V(t_i)$ for all $i \in [k]$.

We now define a logical representation of trees, which is fairly standard (see, e.g., [9]). For this, let Λ be any type of predicate logic that allows us to make use of n-ary predicate symbols $P^{(n)}$, and let F_Λ denote the set of all formulas in Λ without free variables (i.e., the set of sentences of Λ). For $k \in \mathbb{N}$, we say that a predicate symbol $P^{(n)}$ is k-typed if it comes with an associated type $(i_1, \dots, i_n) \in [k]^n$. We write $P\colon i_1 \times \dots \times i_n$ to specify the type of P.

Given a (finite) set $\mathcal{P}$ of predicate symbols, a logical structure $\langle D; (\psi_P)_{P\in\mathcal{P}}\rangle$ consists of a set D called the domain and, for each $P^{(n)} \in \mathcal{P}$, a predicate $\psi_P \subseteq D^n$. If an existing structure Z is enriched with additional predicates $(\psi_P)_{P\in\mathcal{P}'}$ (where $\mathcal{P} \cap \mathcal{P}' = \emptyset$), we denote the resulting structure by $\langle Z; (\psi_P)_{P\in\mathcal{P}'}\rangle$. In this paper, we will only consider structures with finite domains.

A tuple $T = (t_1, \dots, t_k) \in \mathrm{T}^k_\Sigma$ of trees will be represented by the structure

$$|T| = \langle V(T); (V_i)_{i\in[k]}, (\mathrm{lab}_g)_{g\in\Sigma}, (\downarrow_i)_{i\in[r]}\rangle,$$

using the following predicates $V_i^{(1)}$ $(i \in [k])$, $\mathrm{lab}_g^{(1)}$ $(g \in \Sigma)$ and $\downarrow_i^{(2)}$ $(i \in [r])$:

- For every $i \in [k]$, $V_i = V(T, i)$. Thus, $V_i(d)$ expresses that d belongs to t_i.
- For every $g \in \Sigma$, $\mathrm{lab}_g = \{(i, v) \in V(T) \mid i \in [k] \text{ and } t_i(v) = g\}$. Thus, $\mathrm{lab}_g(d)$ expresses that the label of d is g.
- For every $j \in [r]$, $\downarrow_j = \{((i, v), (i, vj)) \mid i \in [k] \text{ and } v, vj \in V(t_i)\}$. Thus, $\downarrow_j(d, d')$ expresses that d' is the jth child of d in one of the trees $t_1, \dots, t_k$. In the following, we write $d \downarrow_j d'$ instead of $\downarrow_j(d, d')$.

Note that, in the definition of $|T|$, we have blurred the distinction between predicate symbols and their interpretation as predicates, because this interpretation is fixed. In the following, especially in intuitive explanations, we shall sometimes also identify the logical structure $|T|$ with the tuple T it represents.

To define Millstream systems, we first formalize our notion of interfaces. The idea is that a tuple $T = (t_1, \dots, t_k)$ of trees, represented as $|T|$, is augmented with additional *interface links* that are subject to logical conditions.

Definition 1 (Interface). *Let Σ be a ranked alphabet. An* interface *on T^k_Σ ($k \in \mathbb{N}$) is a pair $INT = (\mathcal{I}, \Phi)$, such that*

- *$\mathcal{I}$ is a finite set of k-typed predicate symbols called* interface symbols*, and*
- *Φ is a finite set of formulas in F_Λ that may, in addition to the fixed vocabulary of Λ, contain the predicate symbols in $\mathcal{I}$ and those occurring in the structures $|T|$ (where $T \in \mathrm{T}^k_\Sigma$). These formulas are called* interface conditions.

A configuration *(w.r.t. INT) is a structure $C = \langle |T|; (\psi_I)_{I\in\mathcal{I}}\rangle$ with $T \subseteq \mathrm{T}^k_\Sigma$ such that $\psi_I \subseteq V(T, i_1) \times \dots \times V(T, i_l)$ for each $I\colon i_1 \times \dots \times i_l$ in $\mathcal{I}$, and C satisfies the interface conditions in Φ (if each $I \in \mathcal{I}$ is interpreted as ψ_I).*

For $I \in \mathcal{I}$ and nodes $v_1, \dots, v_l$, we say that a configuration C as above contains the link $I(v_1, \dots, v_l)$ if $(v_1, \dots, v_l) \in \psi_I$.

Definition 2 (Millstream system). *Let Σ be a ranked alphabet and $k \in \mathbb{N}$. A* Millstream system *(MS, for short) is a system $MS = (M_1, \dots, M_k; INT)$ consisting of Σ-tree generators $M_1, \dots, M_k$, called the* modules *of MS, and an interface INT on T^k_Σ. The language $L(MS)$ generated by MS is the set of all configurations $\langle |T|; (\psi_I)_{I\in\mathcal{I}}\rangle$ such that $T \in L(M_1) \times \dots \times L(M_k)$.*

In the remainder of the paper, we mainly consider *regular MSO Millstream systems.* This is the special case of Millstream systems where the modules are regular tree grammars (or equivalent devices) and Λ is monadic second-order predicate logic. Similarly, we talk about regular FO Millstream systems if only first-order predicate logic is considered. If we do not want to restrict the type of modules considered, we speak of Λ Millstream systems. Recall that, in fact, the regular tree languages are exactly those which can be specified by MSO formulas over trees, which means that each component language $L(M_i)$ of a regular MSO Millstream system can, if convenient, be assumed to be equal to T_Σ.

3 Emptiness for Millstream Systems is Undecidable

In this section we show that it is undecidable whether the language of a given Millstream system is empty. More precisely, we show the following.

Theorem 3. *Emptiness for regular FO Millstream systems is undecidable.*

Proof sketch. The proof is by reduction from Post's correspondence problem (PCP), which is well known to be undecidable [10]. An instance of PCP over a finite alphabet Σ is a set $I = \{(u_1, w_1), \ldots, (u_n, w_n)\}$ of pairs of words over Σ, i.e., $u_i, w_i \in \Sigma^*$ for $i = 1, \ldots, n$. The question is whether there exists an integer $m \in \mathbb{N}$ and a sequence of indices $i_1, i_2, \ldots, i_m \in \{1, \ldots, n\}$ such that

$$u_{i_1} u_{i_2} \cdots u_{i_m} = w_{i_1} w_{i_2} \cdots w_{i_m}. \tag{1}$$

For an instance $I = \{(u_1, w_1), \ldots, (u_n, w_n)\}$ of PCP over $\Sigma = \{a, b\}$, we show how to construct a regular FO Millstream system $MS_I = (M_1, M_2; INT)$ such that $L(MS_I) \neq \emptyset$ if and only if I has a solution.

The modules M_1 and M_2 generate monadic trees over the ranked alphabet $\Gamma = \{a^{(1)}, b^{(1)}, 1^{(1)}, 2^{(1)}, \ldots, n^{(1)}, \Diamond^{(0)}\}$. Thus, we may view $L(M_1)$ and $L(M_2)$ as regular string languages. The idea is that these languages contain all possible indexed left- and right-hand sides of solutions to Equation (1). To be precise, we let $L(M_1) = (1 \cdot u_1 + \cdots + n \cdot u_n)^* \Diamond$ and $L(M_2) = (1 \cdot w_1 + \cdots + n \cdot w_n)^* \Diamond$.

The interface INT, has two interface symbols, $Link_1$ and $Link_2$, and is used to ensure that only pairs of trees that satisfy Equation 1 can be produced. For instance, the following three formulas are used to ensure that the sequence of indices in the tree produced by M_1 is the same as in the tree produced by M_2:

$$\begin{aligned}
\phi_1 &\equiv \forall x \forall y : (root_1(x) \wedge root_2(y)) \rightarrow Link_1(x, y) \\
\phi_2 &\equiv \forall x \forall y : Link_1(x, y) \rightarrow (index(x) \wedge SameLabel(x, y)) \\
\phi_3 &\equiv \forall x \forall y : Link_1(x, y) \rightarrow \\
&\qquad \exists x' \exists y' : NextIndex_1(x, x') \wedge NextIndex_2(y, y') \wedge \\
&\qquad (Link_1(x', y') \vee (\mathrm{lab}_\Diamond(x') \wedge \mathrm{lab}_\Diamond(y')))
\end{aligned}$$

Here, $SameLabel(x, y)$ and $NextIndex_1(x, x')$ are abbreviations of formulas expressing that x and y have the same label and that x' is the first node below x that has a label in $[n]$, respectively. The full interface definition is given in the Appendix. $\square$

4 The Completion Problem

In this section, the completion problem for regular MSO Millstream systems is studied. Let us first define this problem. Given a Millstream system $MS = (M_1, \ldots, M_k; INT)$ over a ranked alphabet Σ and a set $K \subseteq [k]$, the (uniform) *K-completion problem for MS* is defined as follows:

Instance A family $\kappa = (\kappa_i)_{i \in K}$ of trees $\kappa_i \in \mathrm{T}_\Sigma$.

Question Is there a *completion* of κ, i.e., a configuration $\langle |(t_1, \ldots, t_k)|, \Psi \rangle$ in $L(MS)$ such that $t_i = \kappa_i$ for all $i \in K$?

Intuitively, the trees κ_i, $i \in K$, are the "known trees" of an otherwise unknown configuration in $L(MS)$, which is sought. Note that there can be zero, one, finitely or infinitely many such configurations. Thus, one may also wish to compute a representation of the set of all completions of κ. If we talk about the problem at a general level, we simply call it the completion problem.

The completion problem is of obvious linguistic relevance, as discussed in the introduction. There are two extreme cases of the completion problem. The first is when $K = [k]$, asking whether k given trees can be linked consistently. The second is when $K = \emptyset$, asking whether $L(MS)$ is nonempty. The first is trivially decidable by enumeration, provided that a logic is used for which it can be decided whether a given configuration satisfies a given formula. The second was studied in the previous section, which yields the following corollary of Theorem 3.

Corollary 4. *The completion problem for regular FO Millstream systems is undecidable.*

Our next goal is to identify conditions under which the completion problem becomes decidable. For this, we now define a normal form of MSO Millstream systems, called typed MSO Millstream system, that turns out to be useful. Intuitively, in a typed MSO Millstream system with k modules, every variable is associated with an index $i \in [k]$, indicating that this variable is meant to range only over (sets of) nodes in $V(t_i)$. In the definition, we let $V_i(X)$ abbreviate $\forall x \colon (x \in X \to V_i(x))$.

Definition 5 (Typed MSO Millstream system). *Let $MS = (M_1, \ldots, M_k;$ $INT)$ be an MSO Millstream system. An interface condition φ of MS is* typed *if each quantified subformula of φ is of one of the forms $\exists \xi \colon (V_i(\xi) \wedge \varphi')$ and $\forall \xi \colon (V_i(\xi) \to \varphi')$, where ξ is an individual or set variable and $i \in [k]$. We abbreviate such formulas by $\exists \xi^{(i)} \colon \varphi'$ and $\forall \xi^{(i)} \colon \varphi'$, respectively, and call i the type of ξ. MS is* typed *if each of its interface conditions is typed.*

The following lemma states that MSO Millstream systems can, without loss of generality, be assumed to be typed. The rather straightforward proof, which can be found in the appendix, is based on a recursive construction that replaces every variable ξ by k variables $\xi_1^{(1)}, \ldots, \xi_k^{(k)}$.

Lemma 6. *Every MSO Millstream system can effectively be turned into a typed MSO Millstream system MS', such that $L(MS') = L(MS)$.*

In the following, we assume that variables of type i occur only in the "right places" in typed formulas. For example, if $\mathcal{I}$ contains $I\colon 2 \times 1$ and $I(x,y)$ occurs in a typed formula, then x and y are of types 2 and 1, respectively. This is no restriction, because $I(x,y)$ would necessarily be false, otherwise.

Let $C = \langle (t_1,\dots,t_k), \Psi\rangle$ be a configuration of $MS = (M_1,\dots,M_k; INT)$, where $INT = (\mathcal{I}, \Phi)$, and let $K \subseteq [k]$. We now define the configuration C/K which is obtained by removing the trees with indices in K and their nodes from the interface links, but memorizing the latter by attaching subscripts to the interface symbols, thus hard coding the information into the interface symbol itself. In particular, we make use of a new alphabet of interface symbols which depends on the trees t_i, $i \in K$. Before defining C/K formally, we introduce some convenient notation. For a finite number of indexed elements $a_1,\dots,a_n$, and a condition φ that is true or not for each of the a_i, we denote by $(a_i \mid \varphi(a_i))$ the tuple $(a_{i_1},\dots,a_{i_m})$ such that $i_1 < \dots < i_m$ and $\{i_1,\dots,i_m\} = \{i \in [n] \mid \varphi(a_i)\}$.

Now, we let $C/K = \langle (t_i \mid i \notin K), \Psi'\rangle$, where Ψ' is obtained from Ψ, as follows. For every interface symbol $I\colon i_1 \times \dots \times i_l$ in $\mathcal{I}$, every interface link $I(v_1,\dots,v_l)$ in C is replaced with $I_{u_1\dots u_m}(v'_1,\dots,v'_{l-m})$, where $(u_1,\dots,u_m) = (v_j \mid j \in [l] \text{ and } i_j \in K)$ and $(v'_1,\dots,v'_{l-m}) = (v_j \mid j \in [l] \text{ and } i_j \notin K)$. We define

$$L(MS)/K = \{C/K \mid C = \langle (t_1,\dots,t_k),\Psi\rangle \in L(MS) \text{ and } t_i = \kappa_i \text{ for all } i \in K\}.$$

Lemma 7. *Let $MS = (M_1,\dots,M_k; INT)$ with $INT = (\mathcal{I},\Phi)$ be an MSO Millstream system over Σ, and let $\kappa_i \in \mathrm{T}_\Sigma$ for all $i \in K$, where $K \subseteq [k]$. Then one can effectively construct an MSO Millstream system $MS' = (M'_1,\dots,M'_{k-|K|}; INT')$, with $(M'_1,\dots,M'_{k-|K|}) = (M_i \mid i \in [k] \setminus K)$ and $L(MS') = L(MS)/K$.*

Proof. By induction, and since the statement is trivially true for $K = \emptyset$, it suffices to prove the lemma for $|K| = 1$. Suppose without loss of generality that $K = \{k\}$ and, by Lemma 6, that MS is typed. Furthermore, assume that $i_1 \le \dots \le i_l$, for all $I\colon i_1 \times \dots \times i_l$ in $\mathcal{I}$, and let $m(I) = |\{j \in [l] \mid i_j = k\}|$. In the following, we denote κ_k by κ.

The new alphabet of interface symbols contains all $I_{v_1\dots v_{m(I)}}\colon i_1 \times \dots \times i_{l-m(I)}$, such that $I\colon i_1 \times \dots \times i_l$ is in $\mathcal{I}$ and $v_1,\dots,v_{m(I)} \in V(\kappa)$. It remains to define the interface conditions of MS'. We do this by recursively turning each individual interface condition $\varphi \in \Phi$ into an appropriate interface condition φ' for MS'. In fact, φ' will in general contain atomic subformulas (not involving interface symbols) in which variables $x^{(k)}$ or $X^{(k)}$ have been replaced with constants, i.e., nodes or sets of nodes of κ. Clearly, these constants can be removed by replacing the corresponding subformulas with either true or false, because κ is fixed.

Consider a (typed) MSO formula ϕ without free variables of type k. We define ϕ' as follows.

- If ϕ is atomic, then $\phi' = \phi$ unless $\phi = I(x_1,\dots,x_{l-m(I)},v_1,\dots,v_{m(I)})$ for some interface symbol $I\colon i_1 \times \dots \times i_l$, variables $x_1,\dots,x_{l-m(I)}$, and nodes $v_1,\dots,v_{m(I)} \in V(\kappa)$. In the latter case, $\phi' = I_{v_1\cdots v_{m(I)}}(x_1,\dots,x_{l-m(I)})$.
- If $\phi = \phi_1 \wedge \phi_2$, then $\phi' = \phi'_1 \wedge \phi'_2$, and similarly for $\phi = \phi_1 \vee \phi_2$ and $\phi = \neg\phi_1$.

- If $\phi = \forall x^{(i)} \colon \phi_1$, there are two cases. Either $i < k$, in which case $\phi' = \forall x^{(i)} \colon \phi'_1$, or $i = k$, in which case $\phi' = \bigwedge_{v \in V(\kappa)} \phi_1\langle x \leftarrow v\rangle'$. Here, $\phi_1\langle x \leftarrow v\rangle'$ is the formula obtained by substituting v for all free occurrences of x in ϕ_1.
- If $\phi = \forall X^{(i)} \colon \phi_1$, then $\phi' = \forall X^{(i)} \colon \phi'_1$ if $i < k$, and $\phi' = \bigwedge_{V \subseteq V(\kappa)} \phi_1\langle X \leftarrow V\rangle'$.
- The cases $\phi = \exists x^{(i)} \colon \phi_1$ and $\phi = \exists X^{(i)} \colon \phi_1$ are similar, the only difference being that $\bigwedge$ is replaced with $\bigvee$.

By structural induction on ϕ, it can be shown that, for every assignment of (sets of) nodes of $t_1, \ldots, t_{k-1}$ to the free variables in ϕ and for every configuration $C = \langle(t_1, \ldots, t_{k-1}, \kappa), \Psi\rangle$, C satisfies ϕ if and only if C/K satisfies ϕ'. □

Given a Millstream system $MS = (M_1, \ldots, M_k; INT)$ with $INT = (\mathcal{I}, \Phi)$, we call a set $U \subseteq [k]$ *unlinked* (with respect to MS) if, for all interface symbols $I \colon i_1 \times \cdots i_l$ in $\mathcal{I}$, we have $|\{j \in [l] \mid i_j \in U\}| \leq 1$. In other words, U is unlinked if there are no interface symbols that could establish direct links between the nodes of the trees generated by the modules M_i, $i \in U$.

Theorem 8. *Let $MS = (M_1, \ldots, M_k; INT)$ be a regular MSO Millstream system. For all $K \subseteq [k]$, if $[k] \setminus K$ is unlinked, then a K-completion of κ can be computed for every $\kappa = (\kappa_i)_{i \in K}$, $\kappa_i \in \mathrm{T}_\Sigma$. In particular, the K-completion problem for MS is decidable.*

Proof sketch. Let $l = k - |K|$. By Lemma 7, we can effectively construct a regular MSO Millstream system $MS' = (M'_1, \ldots, M'_l; INT')$ such that $L(MS') = L(MS)/K$. Since $[k] \setminus K$ is unlinked, all interface symbols in INT' are of rank ≤ 1. Adding an additional root symbol on top of every configuration $C = \langle(t_1, \ldots, t_l), \Psi\rangle$ of MS', this shows that $L(MS')$ is essentially a regular tree language (using the fact that the regular tree languages are exactly the MSO-definable ones [14, 5]). Consequently, we can check whether $L(MS')$ is empty and compute a configuration $C \in L(MS')$ if it is not. From C, one can easily construct a configuration $C_0 = \langle(t_1, \ldots, t_k), \Psi\rangle \in L(MS)$ with $C_0/K = C$ and $t_i = \kappa_i$ for all $i \in K$, by reversing the construction of C_0/K. This completes the proof. □

Under the assumptions of the theorem and abstracting from some irrelevant details, $L(MS)/K$ is a regular tree language. Hence, even finiteness can be decided.

Corollary 9. *Given a regular Millstream system $MS = (M_1, \ldots, M_k; INT)$ and $K \subseteq [k]$ s.t. $[k] \setminus K$ is unlinked, it can be decided whether $L(MS)/K$ is finite.*

5 Configurations of Bounded Tree Width

We have seen that the emptiness problem is undecidable, but that the completion problem is decidable in certain cases. Another way to achieve positive results in situations such as those studied here is to consider structures of bounded tree width. Readers who are unfamiliar with this notion may consult, e.g., [2]. In the following, we use tree width to find restrictions under which the completion problem can be solved more efficiently. For this, we regard a configuration $\langle |T|, \Psi \rangle$ as an undirected graph on $V(T)$. More precisely, we say that Ψ contains a link $(v_1, \dots, v_l)$ and write $(v_1, \dots, v_l) \in \Psi$, if $I(v_1, \dots, v_l) =$ true for some interface symbol I. Two distinct nodes u, v are considered to be connected by an edge if one is a child of the other or $u, v \in \{v_1, \dots, v_l\}$ for some link $(v_1, \dots, v_l) \in \Psi$. We say that a set L of configurations has *bounded tree width* if there is a constant $w \in \mathbb{N}$ such that every configuration in L is of tree width at most w.

By the results of [4, 3], we have the following.

Theorem 10. *Let $MS = (M_1, \dots, M_k; INT)$ be a regular MSO Millstream system. If $L(MS)$ is of bounded tree width, then*

- *the membership problem for $L(MS)$ can be solved in linear time,*
- *the K-completion problem for MS is decidable for every $K \subseteq [k]$, and*
- *for every MSO formula φ, it can be decided whether all configurations in $L(MS)$ satisfy φ.*

Thus, Millstream systems MS for which $L(MS)$ is of bounded tree width are of particular interest. In the rest of this section, we establish two conditions that guarantee that $L(MS)$ is of bounded tree width.

Let $MS = (M_1, \dots, M_k; INT)$ be an MSO Millstream system and $C = \langle |(t_1, \dots, t_k)|, \Psi \rangle$ a configuration of MS. We say that a node $v \in t_i$ is *linked* if it occurs in a link in Ψ.

Definition 11 (Simple configuration). *Let $C = \langle |(t_1, \dots, t_k)|, \Psi \rangle$ be a configuration. For nodes $x, x' \in V(t_i)$, let $gcp(x, x')$ be the* greatest common predecessor *of x and x', i.e. the largest (w.r.t. $<$) node v in t_i such that $v < x$ and $v < x'$. C is* simple *if it satisfies the following conditions:*

1. *No node in any of the trees $t_1, \dots, t_k$ is involved in more than one link, i.e., each node $v \in t_i$ occurs in at most one link in Ψ.*
2. *For every $x_1, x_2, x_3 \in t_i$ and every $y_1, y_2, y_3 \in t_j$ such that x_1 is linked to y_1, x_2 is linked to y_2, and x_3 is linked to y_3, $gcp(x_1, x_2) \geq gcp(x_1, x_3)$ if and only if $gcp(y_1, y_2) \geq gcp(y_1, y_3)$.*

The intuition behind the second condition is that the links have to respect the branching structure of the tree. Figure 2 illustrates how a configuration can violate the criterion for simplicity.

Next, we prove that every simple configuration of a Millstream system with two modules has bounded tree-width.

Theorem 12. *If $C = \langle |(t_1, t_2)|, \Psi \rangle$ is a simple configuration of $MS = (M_1, M_2; INT)$, then C has tree-width at most 2.*

Proof sketch. For the proof, we use a graph search game by Seymour and Thomas [13]. The game is played between a robber and k cops on an undirected graph. The robber stands on a vertex v of the graph and can at any time run to any

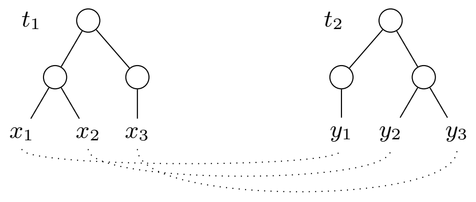

Fig. 2. The above configuration is *not* simple, since the branches of x_1 and x_2 come together *before* the branches of x_2 and x_3, while the branches of y_1 and y_2 come together *after* the branches of y_2 and y_3.

vertex u such that there is a cop-free path from v to u. Each cop is at any time either placed on a vertex or is in a helicopter. The objective of the cops is to land a cop on a vertex occupied by the robber. The robber can, however, see the helicopter approaching and has time to flee to another vertex. Thus the cops have to corner the robber, i.e., create a situation where the robber is on vertex v, all neighbors of v are occupied by cops, and there is a cop in the helicopter available to land on v and capture the robber.

A graph has tree-width k if and only if $k + 1$ is the minimal number of cops that have a winning strategy against the robber on the graph [13]. Thus our aim is to show that 3 cops can catch a robber on any simple configuration $C = \langle |(t_1, t_2)|, \Psi \rangle$. The strategy will be to place one cop in each tree t_i, make sure that the robber can never again get to any node in t_i outside the subtree of the node on which the cop is placed, and in each round move the cop one step down in one of the subtrees. Thus the robber is caught in at most $2 \cdot h$ steps, where h is the maximal height of any tree in C.

The cops will use the following strategy:

1. In the first step, two cops are placed on the two roots of t_1, t_2.
2. After m steps, one cop will be placed in each tree, say on the nodes c_1 and c_2. The robber will be in one of the subtrees $t_1/c_1, t_2/c_2$ and there will be no links from nodes within these subtrees to nodes outside them. Assume that the robber is in subtree t_1/c_1. We distinguish two cases:
   (a) If there is a child node w of c_2 such that no node of t_2 outside of t_2/w is accessible to the robber, and there is no link from any node in t_1/c_1 to c_2, then the free cop is placed on w and the cop on c_2 is removed.
   (b) Otherwise the free cop is placed on the child v of c_1 such that the robber is in t_1/v and the cop on c_1 is removed.

The proof that this strategy works is given in the appendix. □

Let us now define a second notion that can be used to bound the tree width of $L(MS)$. In this definition and in the remainder of the section, if $u, u_1, \ldots, u_n$ are nodes in a configuration of a Millstream system, we write $u > \{u_1, \ldots, u_n\}$ to express that $u > u_i$ for some $i \in [n]$. We say that a node v is a *successor* of u if v is a minimal linked node such that $u > v$.

Definition 13 (Nested Millstream system). *A Millstream system $MS = (M_1, \ldots, M_k; INT)$ is* nested *if there is a constant $h \in \mathbb{N}$ such that the following hold for every configuration $C = \langle |(t_1, \ldots, t_k)|, \Psi \rangle \in L(MS)$.*

(1) There are at most h links in Ψ containing a minimal linked node (i.e., a node that is not a successor of another node).

Furthermore, for every link $(u_1, \ldots, u_l) \in \Psi$,

(2) there are at most h links $(v_1, \ldots, v_m) \in \Psi$ such that v_i is a successor of u_j for some $i \in [m]$ and $j \in [l]$, and
(3) for each of the links $(v_1, \ldots, v_m)$ in (2) and every $i \in [m]$, $v_i > \{u_1, \ldots, u_l\}$.

For a given configuration $C = \langle |T|, \Psi \rangle$ and distinct nodes $u, u' \in V(T)$, we say that u *is linked with* u' if there is a link $(v_1, \ldots, v_l) \in \Psi$ such that $u, u' \in \{v_1, \ldots, v_l\}$. A *$(u, u')$-path in C* is a sequence $u_0 \cdots u_n \in V(T)^*$ such that $u_0 = u$, $u_n = u'$, and for every $i \in [n]$, one of u_{i-1}, u_i is a child of the other or u_{i-1} is linked with u_i. Such a path is said to *use* each of the nodes $u_0, \ldots, u_n$. The proof of the next lemma can be found in the appendix.

Lemma 14. *Let $C = \langle |T|, \Psi \rangle \in L(MS)$, where MS is a nested Millstream system, and let $(v_1, \ldots, v_l) \in \Psi$ and $u, u' \in V(T)$. If $u > \{v_1, \ldots, v_l\}$ and there is a (u, u')-path in C that does not use any of $v_1, \ldots, v_l$, then $u' > \{v_1, \ldots, v_l\}$.*

Theorem 15. *$L(MS)$ is of bounded tree width for every nested Millstream system MS.*

Proof. Let h be the constant in Definition 13, and let $r \geq 1$ be the maximum rank of interface symbols of $MS = (M_1, \ldots, M_k; INT)$. Without loss of generality, we may assume that all configurations in $L(MS)$ contain a link $(v_1, \ldots, v_k)$ such that $v_1, \ldots, v_k$ are the roots of the trees in the configuration. This assumption removes the need for condition (1) in Definition 13, as it becomes an instance of (2).

Using the cops-and-robbers game, we show that the configurations in $L(MS)$ are of tree width at most $\hat{r} = (h+1)r + 2$. The winning strategy for the cops works as follows.

Maintained invariant: During the game, the cops will always completely occupy at least one link $(u_1, \ldots, u_l)$, in the sense that cops are placed on each of $u_1, \ldots, u_l$. This link is called the *guarding link*. Moreover, the strategy will guarantee that the robber sits on a node $v > \{u_1, \ldots, u_l\}$. Consequently, the robber cannot move to any node $v' \not> \{u_1, \ldots, u_l\}$ (by Lemma 14).

Initially, cops are placed on the roots of the trees, making sure that the invariant holds. Now, the following is repeated, where $(u_1, \ldots, u_l)$ is the guarding link and v denotes the current position of the robber at any instant in time:

We use the still available cops to occupy all links $(u'_1, \ldots, u'_m)$ such that u'_i is a successor of u_j for some $i \in [m]$ and $j \in [l]$. By Definition 13(2), this requires at most hr cops in addition to those occupying the guarding link. (Note that, as the cops still occupy the guarding link, the invariant still holds.)

Now, let U be the set of all nodes $u > \{u_1, \dots, u_l\}$ such that there is no successor u'_i of u_i with $u > u'_i$. (In other words, U is the set of descendants of $u_1, \dots, u_l$ that can be reached on a path not using one of the newly occupied nodes.) There are two cases.

If $v \in U$, then the robber can only move within the tree that is given by the connected component of U that v belongs to. (All nodes of links he could reach for travelling along them are occupied by cops.) Thus, the two remaining cops can be used to corner the robber, while keeping the at most $h+1$ links occupied.

If $v \notin U$, then we have $v > \{u'_1, \dots, u'_m\}$, for (at least) one of the newly occupied links $(u'_1, \dots, u'_m)$. We choose this link as the new guarding link, making all other cops available again, and continue. Note that, by Definition 13(3), the sum of the sizes of the subtrees rooted at the nodes of the guarding link has become strictly smaller. Thus, the robber will eventually be caught. □

References

1. S. Bensch and F. Drewes. Millstream systems. Report UMINF 09.21, Umeå University, 2009. Available at `http://www8.cs.umu.se/research/uminf/index.cgi?year=2009&number=21`.
2. H.L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11(1-2):1–22, 1993.
3. B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 193–242. Elsevier and MIT Press, 1990.
4. B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.
5. J. Doner. Tree acceptors and some of their applications. *J. Commput. Syst. Sci.*, 4(5):406–451, 1970.
6. Gécseg F. and Steinby M. Tree languages. In G. Rozenberg and Salomaa A., editors, *Handbook of Formal Languages, Volume 3, Beyond Words*, pages 1–68. Springer, Berlin, 1997.
7. R. Jackendoff. *Foundations of Language: Brain, Meaning, Grammar, Evolution*. Oxford University Press, Oxford, 2002.
8. A. Kornai and Z. Tuza. Narrowness, path-width, and their application in natural language processing. *Discrete Applied Mathematics*, 36:87–92, 1992.
9. L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.
10. E. Post. A variant of a recursively unsolvable problem. *Bulletin of the AMS*, 52:264–268, 1946.
11. J. Sadock. *Autolexical Syntax - A Theory of Parallel Grammatical Representations*. The University of Chicago Press, Chicago & London, 1991.
12. U. Sauerland and A. von Stechow. Syntax-semantics interface. In N. Smelser and P. Baltes, editors, *International Encyclopedia of the Social and Behavioural Sciences*, pages 15412 – 15418. Oxford Pergamon, 2001.
13. P.D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58:22–33, 1993.
14. J.W. Thatcher and J.B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.