ACADEMIC PRESS

Available at www.ComputerScienceWeb.com POWERED BY SCIENCE @ DIRECT•

Journal of Computer and System Sciences 66 (2003) 775–808

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

http://www.elsevier.com/locate/jcss

Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width[☆]

Georg Gottlob,[a,*] Nicola Leone,[b] and Francesco Scarcello[c]

[a] *Inst. für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria*
[b] *Department of Mathematics, University of Calabria, I-87030 Rende, Italy*
[c] *DEIS, University of Calabria, I-87030 Rende, Italy*

Received 24 September 2001; revised 15 June 2002

Abstract

In a previous paper (J. Comput. System Sci. 64 (2002) 519), the authors introduced the notion of hypertree decomposition and the corresponding concept of hypertree width and showed that the conjunctive queries whose hypergraphs have bounded hypertree width can be evaluated in polynomial time. Bounded hypertree width generalizes the notions of acyclicity and bounded treewidth and corresponds to larger classes of tractable queries. In the present paper, we provide natural characterizations of hypergraphs and queries having bounded hypertree width in terms of game-theory and logic. First we define the Robber and Marshals game, and prove that a hypergraph H has hypertree width at most k if and only if k marshals have a winning strategy on H, allowing them to trap a robber who moves along the hyperedges. This game is akin the well-known Robber and Cops game (which characterizes bounded treewidth), except that marshals are more powerful than cops: They can control entire hyperedges instead of just vertices. Kolaitis and Vardi (J. Comput. System Sci. 61 (2000) 302) recently gave an elegant characterization of the conjunctive queries having treewidth $<k$ in terms of the k-variable fragment of a certain logic L ($=$ existential-conjunctive fragment of positive FO). We use the Robber and Marshals game to derive a surprisingly simple and equally elegant characterization of the class $HW[k]$ of queries of hypertree width at most k in terms of guarded logic. In particular, we show that $HW[k] = GF_k(\mathrm{L})$, where $GF_k(\mathrm{L})$ denotes the k-guarded fragment of L. In this fragment, conjunctions of k atoms rather than just single atoms are allowed to act as guards. Note that, for the particular case $k = 1$, our results provide new characterizations of the class of acyclic queries. We extend the notion of bounded hypertree width to nonrecursive stratified

[☆] A preliminary version of this paper appeared in the Proceedings of the 20th ACM Symposium on Principles of Database Systems (PODS'O1), Santa Barbara, May 2001.

[*] Corresponding author.

E-mail addresses: gottlob@dbai.tuwien.ac.at (G. Gottlob), leone@unical.it (N. Leone), scarcello@unical.it (F. Scarcello).

doi:10.1016/S0022-0000(03)00030-8

Datalog and show that the k-guarded fragment $GF_k(\mathrm{FO})$ of first-order logic has the same expressive power as nonrecursive stratified Datalog of hypertree width at most k.

1. Introduction and overview of results

Conjunctive queries (CQs) have been studied for a long time in database theory. This class of queries, equivalent in expressive power to the class of Select-Project-Join queries, is probably the most fundamental and most thoroughly analyzed class of database queries.

There is currently a renewed interest in complexity questions related to CQs. Specifically, the *combined complexity* [31] of certain classes of CQs, is a hot topic of recent database research. The combined complexity is the complexity of evaluating a query on a database measured both in terms of the size of the query and the size of the database. While this problem is known to be NP-complete in general [5], and in PTIME for the restricted class of acyclic queries [15,33], several recent papers [6,10,14,18,20,21,23] identify and analyze very large parameterized classes of conjunctive queries whose evaluation is tractable. The reason for this quest is twofold. Firstly, it is well-known that the problem of *conjunctive query containment* is essentially the same as the problem of CQ evaluation [5]. Conjunctive query containment is of central importance in *view-based query processing* [1,30] which arises, e.g., in the context of data warehousing. Secondly, conjunctive query evaluation is essentially the same problem as *constraint satisfaction*, one of the major problems studied in the field of AI, and there has been a lot of recent interaction between the areas of query optimization and constraint satisfaction (see Vardi's survey paper [32]).

In this paper, we adopt the logical representation of a relational database [2,29], where data tuples are identified with logical ground atoms, and conjunctive queries are represented as Datalog rules. In particular, a *Boolean* conjunctive query (BCQ) is represented by a rule whose head is variable-free, i.e., propositional. We are interested in polynomially solvable classes of conjunctive queries that are determined by structural properties of the query hypergraph or query graph. The *hypergraph* $\mathcal{H}(Q)$ associated with a conjunctive query Q is defined as $\mathcal{H}(Q) = (V, H)$, where the set V of vertices consists of all variables occurring in the body of Q, while the set H of hyperedges contains, for each atom A in the rule body, the set $var(A)$ of all variables occurring in A.

Example 1. Consider the following query Q_1:

$$\begin{aligned} Q_1 : ans \leftarrow\; & a(S,X,T,R) \wedge b(S,Y,U,P) \wedge f(R,P,V) \wedge \\ & g(X,Y) \wedge c(T,U,Z) \wedge d(W,X,Z) \wedge e(Y,Z) \end{aligned}$$

Fig. 1(A) shows its associated hypergraph.

The *query graph* $\mathcal{G}(Q)$ of a conjunctive query Q (also referred to as its *Gaifman graph*) is the primal graph of the hypergraph $\mathcal{H}(Q)$, i.e., the graph whose vertices are those of $\mathcal{H}(Q)$ and whose edges connect a pair of distinct variables $X, Y \in V$ iff $\{X, Y\} \subseteq h$ for some $h \in H$. If each

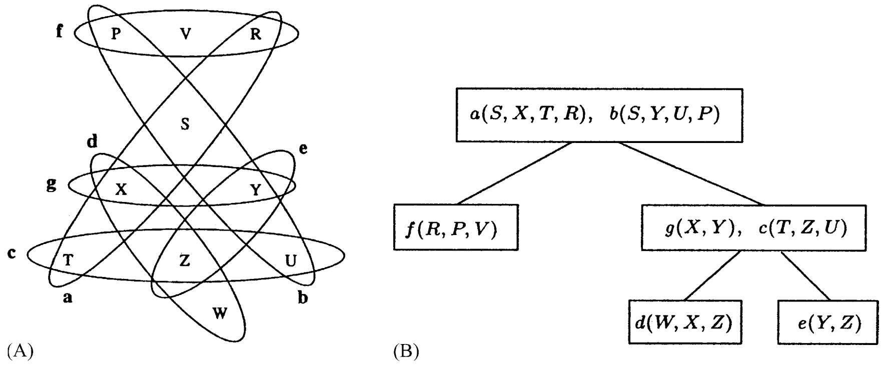


Fig. 1. (A) Hypergraph $\mathcal{H}(Q_1)$, and (B) a width 2 hypertree decomposition of $\mathcal{H}(Q_1)$ in Example 1.

atom in the body of a query contains exactly two variables, then the hypergraph associated with the query is identical to its primal graph.

A *structural query decomposition method*[1] is a method of appropriately transforming a conjunctive query into an equivalent tree query (i.e., acyclic query given in form of a join tree) by organizing its atoms into a polynomial number of clusters, and suitably arranging the clusters as a tree (see Fig. 1(B)). Each cluster contains a number of atoms. After performing the join of the relations corresponding to the atoms jointly contained in each cluster, we obtain a join tree of an acyclic query which is equivalent to the original query. The resulting query can be answered in output-polynomial time by Yannakakis's well-known algorithm [33]. In case of a Boolean query, it can be answered in polynomial time.

The tree of atom-clusters produced by a structural query decomposition method on a given query Q is referred to as the *decomposition* of Q. Fig. 1(B) shows a possible decomposition of query Q_1 of Example 1.

The efficiency of a structural decomposition method essentially depends on the maximum size of the produced clusters, measured (according to the chosen decomposition method) either in terms of the number of variables or in terms of the number of atoms. For a given decomposition, this size is referred to as the *width* of the decomposition. For example, if we adopt the number of atoms as width, then the width of the decomposition shown in Fig. 1(B) is 2. Intuitively, the complexity of transforming a given query into an equivalent tree query is exponential in the width. In fact, for each of the (polynomially many) clusters, the complexity of performing the join of all relations is bounded by the width.

Various structural query (or CSP) decomposition methods have been introduced in both database theory and AI. We can divide these methods in two main groups: Graph-based methods and hypergraph-based methods. The first group uses the query graph $\mathcal{G}(Q)$ for defining a decomposition,[2] while the second group relies on the query hypergraph $\mathcal{H}(Q)$.

[1] In the field of constraint satisfaction, the same notion is known as *structural CSP decomposition method*, cf. [14].

[2] Sometimes, instead of using the query graph, another graph called the *variable-atom incidence graph* [6] (or the *hidden variable encoding* [26]) is used. This is irrelevant to the results of the present paper. In particular, the comparison results shown in Fig. 2 remain the same for both graphs.

The known graph based decomposition methods are: biconnected components [11], cycle cutset [8], tree clustering [9], and tree decompositions [6,23,25]. The hypergraph-based decomposition methods are hinge decomposition [21,22], hinge decomposition with tree clustering [21], cycle hypercutset [14], and hypertree decomposition [16]. A compact description of each of these methods is given in [14], where we have compared the various methods. Fig. 2 shows the result of this comparison. This figure in addition mentions another method (ω^*) which is known to be equivalent to the tree-clustering method [9].

An arrow from method D_1 to method D_2 in Fig. 2 indicates that D_2 is strongly more general than D_1. This means that the following conditions hold: (i) whenever method D_1 produces decompositions of bounded width on a class C of input queries, also method D_2 produces bounded-width decompositions; and (ii) for some classes of hypergraphs, D_2 produces bounded-width decompositions, while D_1 produces decompositions having unbounded width. Since this relationship is transitive, also a directed path between two methods indicates the same relationship. The picture is *complete* in the sense that there is a directed path from method D_1 to method D_2 *if and only if* D_2 strongly generalizes D_1. On the other hand, whenever two methods are not related by a directed path, then they are *incomparable* to each other. (For a more precise explanation of this figure and more subtle relationships between the methods, refer to [14].) Hypertree width was also compared to the notion of *clique width* [7] in [18], where it is shown that hypertree width of a hypergraph strongly dominates the clique width of the associated incidence graph. Note that determining whether a graph or hypergraph has clique width smaller than a given constant is currently not known to be tractable. For this reason, the notion of bounded clique width is currently not considered in the *query tractability hierarchy* depicted in Fig. 2.

From Fig. 2 it is clear that the method of *treewidth* is the most general graph-based decomposition methods. This method is based on the well-known concepts of *tree decomposition* and *graph treewidth* as introduced by Robertson and Seymour [25]. Treewidth is, in a precise sense, the absolutely most general graph-based decomposition method. In fact, Grohe,

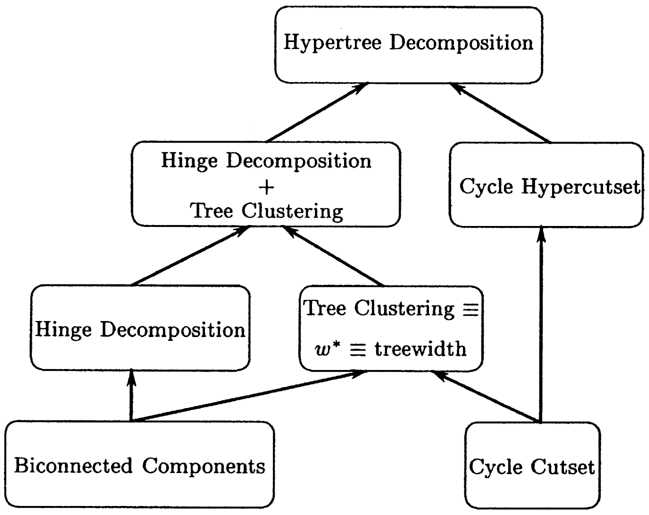


Fig. 2. Query tractability hierarchy.

Schwentick, and Segoufin have shown in a very recent paper [20] that, for any class C of graphs of unbounded treewidth, the evaluation problem for the class Q_C of all conjunctive queries whose query graph is in C is not fixed-parameter tractable, and is thus unlikely to be tractable. (See also the discussion at the end of Section 2, where we compare these results to tractability results for queries with bounded hypertree width.)

Treewidth is an extremely well-studied concept and has been characterized in various ways. We recall two very elegant and useful characterizations: the game theoretic characterization of graph treewidth by Seymour and Thomas [28], and the logical characterization of query treewidth by Kolaitis and Vardi [23].

Game theoretic characterization of treewidth: the robber and cops game [28]

The robber and cops game is played on a graph $G = (V, E)$ by a robber and k cops, where $k > 1$. The robber stands on a vertex p, and can at any time run at great speed along the paths of G. She is not permitted to run through a vertex controlled by a cop, however. Each of the k cops either stands on a vertex or is in a helicopter. The goal of the cops is to land one of them via helicopter on the vertex occupied by the robber, while the robber clearly tries to avoid her capture. Helicopters allow cops to move arbitrarily. The flying cops can see the robber, but also the robber can see the helicopters approaching and change vertex before they land. Seymour and Thomas proved that a graph has treewidth at most k if and only if $k+1$ cops have a winning strategy for this game on G. Moreover, they show that such a strategy exists if and only if the cops can capture the robber in a *monotonic* way, i.e., never returning to a vertex that a cop has previously vacated, which implies that the moving area of the robber is monotonically shrinking.

Logical characterization of treewidth

Kolaitis and Vardi [23] proved that the class of all queries having treewidth $< k$ coincides in expressive power with the class L^k, which is the class of all queries that can be formulated in the k-variable fragment of first-order logic (FO) whose connectives are restricted to existential quantification and conjunction (i.e., $\neg$, $\vee$, and $\forall$ are disallowed). The logic L is the simplest and most basic logic for database querying that allows for nested quantification; in [23], this class is called $\exists FO_{\wedge,+}$.

The game-theoretic characterization of treewidth was profitably used as a tool for proving several results on treewidth. Moreover, it provides a nice intuitive understanding of the notion of treewidth. The logical characterization allows us to precisely assess the expressive power of the bounded treewidth queries.

More recently, Flum et al. [10] generalized the logical characterization of bounded treewidth by lifting it from the level of conjunctive queries to the level of nonrecursive stratified Datalog. In particular, they defined, for each integer k, a version of nonrecursive stratified Datalog where all rule bodies correspond to queries of treewidth $\leqslant k+1$, and showed that this formalism is equivalent in expressive power to FO^k, i.e., to k-variable first-order logic.

Let us now turn to the hypergraph-based methods. As clearly stated in Fig. 2, the method of *hypertree width* is more general than all other currently known methods. This method was introduced in [16] as a generalization of the method of *query width* by Chekuri and Rajaraman [6], which is not considered in Fig. 2 because queries of k-bounded query width are not efficiently recognizable [16] for each constant $k \geqslant 4$. The queries of hypertree width at most k are polynomially recognizable for every constant k and, moreover, the hypertree decomposition method generalizes the query decomposition method [16].

Precise definitions of the concepts of hypertree decomposition and hypertree width are given in Section 2, and tools for computing and drawing hypertree decompositions are available at the hypertree decomposition homepage [27]. Let us just give an informal explanation here. In principle, a hypertree decomposition of a conjunctive query is a clustering of the query atoms as in Fig. 1(B), where the following classical connectedness condition for join-trees holds: If X is a variable occurring in the query, then the set of clusters in which X occurs spans a connected subtree of the decomposition tree. However, a hypertree decomposition may deviate in two ways from this principle: (1) An atom already used in some cluster may be reused in some other cluster; (2) If an atom is reused, some of its variables may not be required to fulfill the connectedness condition. Thus, each cluster v has a set of associated atoms $\lambda(v)$ and a set of effective variables $\chi(v)$ that are subject to the connectedness condition, while all variables that appear in the atoms of $\lambda(v)$ but are not included in $\chi(v)$ are "ineffective" for v and do not count w.r.t. the connectedness condition. Example 2 in Section 2 shows a hypertree decomposition with reused atoms.

At a first glance, the definition of hypertree width may seem to be rather ad hoc and intuitively hard to grasp. In particular, until recently, it was not clear to the authors and to many of their colleagues, whether the class $HW[k]$ of conjunctive queries of hypertree width $\leqslant k$ corresponds to a robust class that can be described in natural (i.e., less formal) terms and whether $HW[k]$ can be compared to other well-known classes of queries in terms of expressive power. In view of the nice characterizations of treewidth, these somewhat vague questions give rise to the following more punctual questions, that are the main topics of the present paper:

- Is there a simple game-theoretic characterization of $HW[k]$?
- Is there a logic for $HW[k]$?
- Can the notion of hypertree width be generalized to the context of nonrecursive stratified Datalog, and if so, does this provide us with an interesting characterization of some large fragment of first-order logic?

We show in this paper that all three questions have an affirmative answer.

Game theoretic characterization of hypertree width: the robber and marshals game

We describe a new game, the *robber and marshals game* (*R&Ms game*). A marshal is more powerful than a cop. While a cop can control a single variable only, i.e., a vertex of the query hypergraph, a marshal controls an entire hyperedge (corresponding to a query atom). In the R&Ms game, the robber moves on variables just as in the robber and cops game, but now marshals instead of cops are chasing her. During a move of the marshals from the set of hyperedges E to the set of hyperedges E', the robber cannot pass through the vertices in $B = (\cup E) \cap (\cup E')$, where, for a set of hyperedges F, $\cup F$ denotes the union of all hyperedges in F. Intuitively, the vertices in B are those not released by the marshals during the move. As in the monotonic robber and cops game, it is required that the marshals capture the robber by monotonically shrinking the moving space of the robber. The game is won by the marshals if they corner the robber somewhere in the hypergraph. We show that the class $HW[k]$ exactly corresponds to the class of all queries Q such that k marshals have a winning strategy on the hypergraph $\mathscr{H}(Q)$.

Logical characterization of hypertree width

We characterize $HW[k]$ in terms of a guarded logic. Recall that L denotes the existential conjunctive fragment of positive FO. We show that $HW[k] = GF_k(\mathrm{L})$, where $GF_k(\mathrm{L})$ denotes the k-guarded fragment of L. Here, up to k atoms may jointly act as a guard. Note that for each k, the k-guarded fragment is a subfragment of the *loosely guarded fragment* as defined (in the context of full FO) by Van Benthem [4], where an arbitrary number of atoms may jointly act as guards, see also [19].

For the particular case $k = 1$, our characterization gives us a new characterization of the acyclic queries stating that the acyclic queries are precisely those expressible in the guarded fragment of L. In order to prove these results, we play the robber and marshals game on the appropriate query hypergraphs. Thus, our proof of the logical characterization of $HW[k]$ already takes profit of the game-theoretic characterization of $HW[k]$.

Generalization

We generalize the notion of bounded hypertree width from the context of conjunctive queries to the context of nonrecursive stratified Datalog programs (NRS-DATALOG). We define the class *NRS-DATALOG*$[k]$ as restriction of NRS-DATALOG, where each rule-body has hypertree-with $\leqslant k$ and satisfies some additional syntactical conditions. We then show that *NRS-DATALOG*$[k]$ has the same expressive power as k-guarded FO, i.e., the generalized version of guarded FO, where a guard may consist of a conjunction of up to k atoms.

The rest of this paper is organized as follows. Section 2 contains the basic definitions. Section 3 describes the robber and marshals game and characterizes $HW[k]$ in terms of this game. Section 4 deals with the logical characterization of $HW[k]$. Section 5 describes the generalization to nonrecursive stratified Datalog. The paper is concluded with the description of a number of interesting open problems in Section 6.

2. Preliminaries and basic definitions

Since we always deal with hypergraphs corresponding to conjunctive queries, the vertices of any hypergraph $\mathscr{H} = (V, H)$ can be viewed as the variables of some conjunctive query. Thus, we will often use the term *variable* as a synonym for vertex, when referring to elements of V, and we assume that every variable in V occurs in at least one edge of $\mathscr{H}$. Moreover, for the hypergraph $\mathscr{H} = (V, H)$, $var(\mathscr{H})$ and $edges(\mathscr{H})$ denote the sets V and H, respectively. For any set of edges $H' \subseteq edges(\mathscr{H})$, let $var(H') = \bigcup_{h \in H'} h$. More generally, for any formal object α, $var(\alpha)$ designates the set of variables occurring in α.

We assume w.l.o.g. that all hypergraphs under consideration are *connected*, i.e., their primal graph consists of a single connected component. All our definitions and results easily extend to general hypergraphs.

A *hypertree for a hypergraph* $\mathcal{H}$ is a triple $\langle T, \chi, \lambda \rangle$, where $T = (N, E)$ is a rooted tree, and χ and λ are labelling functions associating to each vertex $p \in N$ sets $\chi(p) \subseteq var(\mathcal{H})$ and $\lambda(p) \subseteq edges(\mathcal{H})$. If $T' = (N', E')$ is a subtree of T, we define $\chi(T') = \bigcup_{v \in N'} \chi(v)$. We denote the set of vertices N of T by $vertices(T)$, and the root of T by $root(T)$. Moreover, for any $p \in N$, T_p denotes the subtree of T rooted at p.

Definition 1. A *hypertree decomposition* of a hypergraph $\mathcal{H}$ is a hypertree $HD = \langle T, \chi, \lambda \rangle$ for $\mathcal{H}$ which satisfies all the following conditions:

1. for each edge $h \in edges(\mathcal{H})$, there exists a decomposition vertex $p \in vertices(T)$ such that $var(h) \subseteq \chi(p)$ (we say that p *covers* h);
2. for each $Y \in var(\mathcal{H})$, the set $\{p \in vertices(T) \mid Y \in \chi(p)\}$ induces a (connected) subtree of T;
3. for each $p \in vertices(T)$, $\chi(p) \subseteq var(\lambda(p))$;
4. for each $p \in vertices(T)$, $var(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p)$.

Note that the inclusion in condition 4 is actually an equality, because condition 3 implies the reverse inclusion.

The *width* of a hypertree decomposition $\langle T, \chi, \lambda \rangle$ is defined as $max_{p \in vertices(T)} |\lambda(p)|$. The *hypertree width* $hw(\mathcal{H})$ of $\mathcal{H}$ is the minimum width over all its hypertree decompositions.

An edge $h \in edges(\mathcal{H})$ is *strongly covered* in HD if there exists $p \in vertices(T)$ such that $var(h) \subseteq \chi(p)$ and $h \in \lambda(p)$. In this case, we say that p strongly covers h. A hypertree decomposition HD of hypergraph $\mathcal{H}$ is a *complete decomposition* of $\mathcal{H}$ if every edge of $\mathcal{H}$ is strongly covered in HD.

Proposition 1 (Gottlob et al. [16]). *For each hypergraph* $\mathcal{H}$ *having* $hw(\mathcal{H}) = k$, *there exists a complete* k*-width hypertree decomposition of* $\mathcal{H}$.

The acyclic hypergraphs are precisely those hypergraphs having hypertree width one [16].

Let k be a fixed positive integer. We say that a hypergraph $\mathcal{H}$ has k-bounded hypertree-width if $hw(\mathcal{H}) \leqslant k$.

In [16], it is proven that deciding k-bounded hypertree width is in LOGCFL, and thus it is a quite easy and highly parallelizable task. Moreover, from the results in [17] it follows that even computing a k-width hypertree decomposition (if any) is feasible in polynomial time and parallelizable, since this problem belongs to the functional version $\mathrm{L}^{\mathrm{LOGCFL}}$ of LOGCFL. A polynomial time algorithm for computing hypertree decompositions is described in [13], and its implementation is available on the Web [27].

For a query Q, we define the hypertree width $hw(Q)$ of Q as the hypertree width of its associated hypergraph $\mathcal{H}(Q)$, and we say that Q has k-bounded hypertree-width if $hw(Q) \leqslant k$.

Example 2. Consider the following conjunctive query Q_2:

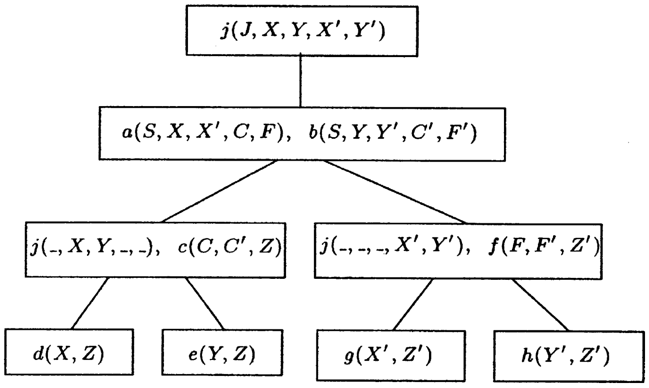
$$\begin{aligned} ans \leftarrow & a(S,X,X',C,F) \wedge b(S,Y,Y',C',F') \wedge c(C,C',Z) \\ & \wedge d(X,Z) \wedge e(Y,Z) \wedge f(F,F',Z') \wedge g(X',Z') \\ & \wedge h(Y',Z') \wedge j(J,X,Y,X',Y') \end{aligned}$$

Fig. 3 shows a (complete) hypertree decomposition HD_2 of $\mathcal{H}(Q_2)$. In figure, the cluster of atoms enclosed in a vertex p represents the label $\lambda(p)$ of p. The set of variables appearing in (the atoms of) p represents the label $\chi(p)$ of p; the anonymous variable '-' is placed in the arguments whose corresponding variables are in $var(\lambda(p)) - \chi(p)$. Thus, the presence of an anonymous variable in an atom A reveals that atom A is reused with a subset of its variables. For instance, let us name p the child of the root. Then, we have $\lambda(p) = \{a,b\}$ and $\chi(p) = \{S,X,X',C,F,Y,Y',C',F'\}$. For the left child q of p, we have $\lambda(q) = \{j,c\}$ and $\chi(q) = \{X,Y,C,C',Z\}$. The atom with predicate j, appearing in the root, is then reused in two further clusters of the decomposition (together with c and f, respectively).

Decomposition HD_2 has width 2, as 2 is the maximum cardinality of the clusters of the decomposition. Since Q_2 is a cyclic query and only acyclic queries have hypertree width 1, it follows that $hw(Q_2) = 2$.

This query highlights the importance of reusing query atoms (actually, the edges corresponding to them) in the clusters of hypertree decompositions, whose allowance is a key feature of this method. The reuse of atom with predicate j (appearing in three clusters of the decomposition) with a subset of its variables is strongly needed to get such a low-width hypertree decomposition. Indeed, the decomposition method based on the query width [6], which forbids such a reuse of atoms, is less efficient than the hypertree-decomposition method on this query, as there exists no query decomposition of width 2 for Q_2 (the query width of Q_2 is 3) [16].

Let $\mathcal{H}$ be a hypergraph, let $V \subseteq var(\mathcal{H})$ be a set of variables, and $X, Y \in var(\mathcal{H})$. Then X is $[V]$-adjacent to Y if there exists an edge $h \in edges(\mathcal{H})$ such that $\{X,Y\} \subseteq h - V$. A $[V]$-path π from X to Y is a sequence $X = X_0, \ldots, X_\ell = Y$ of variables such that X_i is $[V]$-adjacent to X_{i+1}, for each $i \in [0 \ldots \ell - 1]$. A set $W \subseteq var(\mathcal{H})$ of variables is $[V]$-connected if, for all $X, Y \in W$, there is a $[V]$-path from X to Y.



j(J, X, Y, X', Y')
a(S, X, X', C, F), b(S, Y, Y', C', F')
j(_, X, Y, _, _), c(C, C', Z)
j(_, _, _, X', Y'), f(F, F', Z')
d(X, Z)
e(Y, Z)
g(X', Z')
h(Y', Z')

Fig. 3. A 2-width hypertree decomposition of $\mathcal{H}(Q_2)$.

A $[V]$-*component* is a maximal $[V]$-connected nonempty set of variables $W \subseteq var(\mathcal{H}) - V$. For any $[V]$-component C, let $edges(C)$ denote the edges of $\mathcal{H}$ having some variable belonging to C, i.e., $edges(C) = \{h \in edges(\mathcal{H}) \mid h \cap C \neq \emptyset\}$. Note that, by the maximality of components, $var(edges(C)) - C \subseteq V$ immediately follows.

Remark 1. Before concluding this section, we would like to discuss an apparent (but not real) contrast of the tractability of queries of bounded hypertree width with recent results in [20]. Grohe et al. [20] show that, for any class C of graphs, the evaluation problem for the class Q_C of all conjunctive queries whose query graph is in C is fixed-parameter tractable if and only if C has bounded treewidth. This means that if C has unbounded treewidth, Q_C is intractable unless some unexpected collapse of fixed-parameter complexity classes happens. On the other hand, many classes of queries of bounded hypertree width have unbounded treewidth but are still tractable. The simplest example is the class $ACYCLIC$ of acyclic Boolean queries (i.e., those of hypertree width one), which is generally considered one of the most important classes in the database literature. It is trivial to see that $ACYCLIC$ has unbounded treewidth, but it is well-known that queries in this class can be answered in polynomial time [33]. At a first glance, this seems to be in contradiction with the above mentioned result of [20]. However, it is easy to see that the class of acyclic queries cannot be represented as a class Q_C encompassing *all* queries whose query graphs lie in some class C. In other terms, there exists no class C of graphs such that $ACYCLIC = Q_C$. Therefore, the results of [20] simply do not apply to this class, nor do they apply to many other classes of bounded hypertree-width queries. The reason is that acyclicity and bounded hypertree width rely on properties of hypergraphs associated with CQs, rather than on the less informative query graphs.

Note that if one assumes a *fixed* database schema, then a class of queries has bounded treewidth if and only if it has bounded hypertree width. The notion of bounded hypertree width is thus of particular relevance in cases where the database schema is part of the input, i.e., when combined complexity in its general setting is considered. But even in case of a fixed database schema hypertree decompositions may be extremely useful because the hypertree width of a query can be drastically smaller than its treewidth and thus lead to much better evaluation algorithms.

3. The robber and marshals game

An informal description of the R&Ms game was given in the Introduction. Here we formally define the R&Ms game played on an hypergraph $\mathcal{H} = \langle V, H \rangle$.

A *k-configuration* is a pair (M, r), where M is a set of cardinality at most k of edges, and r is a variable. Intuitively, M is the set of edges occupied by the k marshals and r is the variable where the robber stands on. Note that the set M may contain less than k edges. This case occurs when two marshals stand on the same edge of the hypergraph (or at the beginning of the game, when there are no marshals at all). A k-configuration (M, r) is a *capture k-configuration* if $r \in var(M)$, i.e., the robber has been captured by the marshals. In this section, we always consider at most k marshals. Thus, whenever it does not lead to confusion, we drop the k, e.g., we will write directly "configuration", rather than "k-configuration."

If (M, r) is a capture configuration, then the *escape space* of r with respect to M, denoted by $Escape(M, r)$, is the empty set; otherwise, it is the $[var(M)]$-*component* of $\mathcal{H}$ that contains r. Indeed, at any time, the robber can run arbitrarily fast to any variable $[var(M)]$-*connected* to r.

The initial configuration of a game is $(\emptyset, r_0)$, where r_0 is any arbitrarily chosen variable of the hypergraph. Since there are no marshals on the hypergraph and we assumed $\mathcal{H}$ to be connected, the escape space for the initial configuration is the set of all variables V.

Let (M_i, r_i) be the configuration at the ith step. If this is not a capture configuration, the marshals move to a new set of edges M_{i+1}. While the marshals are moving, the robber can move to any variable r_{i+1} which is $[var(M_i) \cap var(M_{i+1})]$-*connected* to r_i. Note that r_i does not belong to $var(M_i)$, hence at least r_i is $[var(M_i) \cap var(M_{i+1})]$-*connected* to itself. The pair $\langle (M_i, r_i), (M_{i+1}, r_{i+1}) \rangle$ is a *feasible step*. If $Escape(M_{i+1}, r_{i+1}) \not\subset Escape(M_i, r_i)$, we say that an *escape step* occurred, and the robber has won. Otherwise, the game continues with the next step. Thus, in every successful game for the marshals, the escape spaces for the robber monotonically decrease at each step. In this case, we say that the (escape space) *narrowing property* holds.

Example 3. Let us play the robber-and-marshals game on the hypergraph $\mathcal{H}(Q_1)$ of query Q_1 in Example 1. Let S be the variable chosen by the robber as her first position. Fig. 4 shows the initial configuration of the R&M game on $\mathcal{H}(Q_1)$. Note that, since at this point there is no marshal on the hypergraph, the escape space of the robber is the set of all variables $var(\mathcal{H}(Q_1))$.

We can easily recognize that two marshals can always capture the robber and win the game by using the following strategy: Independently of the initial position of the robber, the two marshals initially move on edges $\{a, b\}$, and thus control variables T, X, S, R, P, Y, U. After this move of the marshals, the robber may be in V, in Z or in W, as shown in Figs. 5A, 6A, and 6B, respectively.

If the robber is in V, then the marshals move on edge f, and capture the robber, as shown in Fig. 5B (note that the robber cannot escape from V during this move, as both P and R—the only possible ways to leave V—are kept under marshals' control during the move).

Otherwise, if the robber is on Z or on W (see Figs. 6A and 6B, respectively), then the marshals move on $\{g, c\}$, as shown in Fig. 7A. Since they keep the control of X, Y, T, and U during the

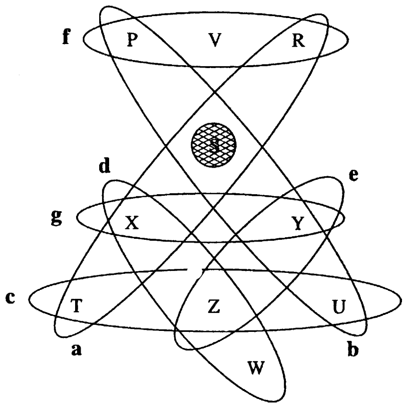


Fig. 4. The initial configuration of the R&M game on $\mathcal{H}(Q_1)$.

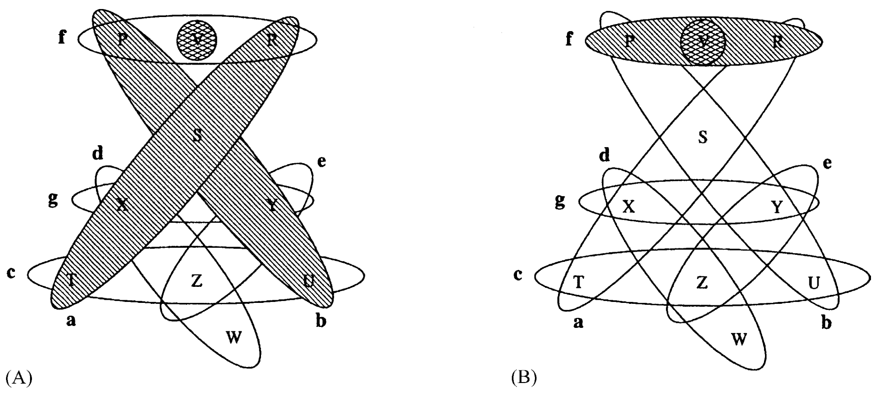


Fig. 5. (A) The second configuration if the robber goes to V; (B) move of the marshals with the robber on V (capture position).

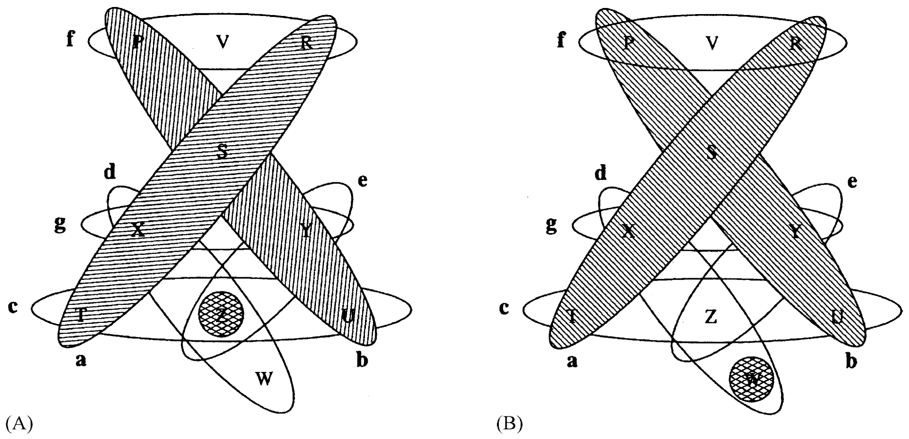


Fig. 6. The second configuration if the robber goes to Z (A); or the robber goes to W (B).

move, then the robber can escape only to variable W. Therefore, a further move on edge d allows the marshals to eventually capture the robber, as shown in Fig. 7B.

In general, a *strategy* for a player in a positional game is a function $\sigma(Hist, Pos)$ which, for a given history *Hist* of past moves and a given current position *Pos*, determines the next move of the player. In order to define strategies for the marshals in the *R&Ms* game, we assume without loss of generality that the game is history-independent. Indeed, it is easy to see that repetitions are not an issue in the rules of the R&Ms game. Moreover, observe that the robber can move freely along the edges of the hypergraph while the marshals are moving. In particular, if $\langle (M,r),(M',r') \rangle$ is a feasible step. $Escape(M,r) = Escape(M,s)$, and $Escape(M',r') = Escape(M',s')$, then $\langle (M,s),(M',s') \rangle$ is a feasible step, too. Therefore, intuitively, it makes no sense for marshals to assume that the robber is at a particular vertex within a determined escape space, when they have to decide their next move. The only relevant information for the marshals is the current escape space of the robber. Thus, we now focus on the family of strategies where the marshals choose their next move just on the basis of the escape space of the robber, i.e., those strategies

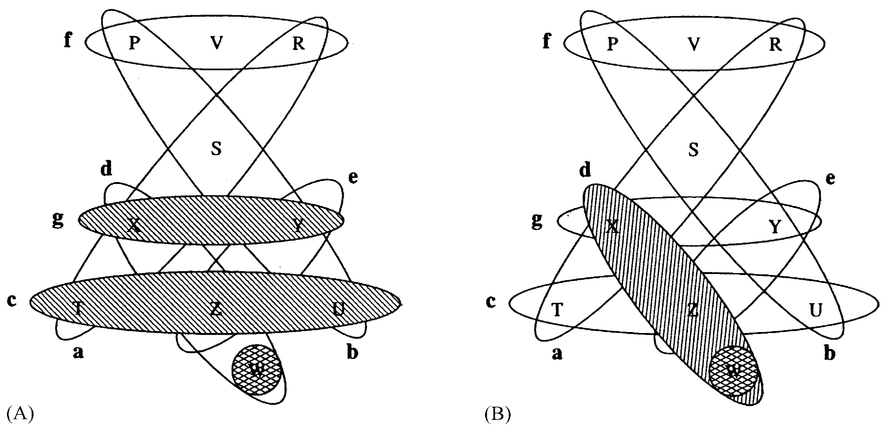


Fig. 7. (A) Move of the marshals if the robber stands on W or on Z; (B) the marshals capture the robber in W.

where marshals make a unique move for all the locations of the robber belonging to the same escape space. For the interested reader, we formally prove in the appendix that these strategies are in fact equivalent to general strategies where marshals can choose different moves for different positions of the robber within the same escape space.

We define a *compact k-configuration*, or simply a k-configuration, if no confusion arises, as a pair (M, C) such that M is a k-location, and C is an escape space for the robber w.r.t. M, i.e., it is either the empty set or a $[var(M)]$-*component* of $\mathcal{H}$.

A *strategy* may be conveniently represented as a function σ that, given a compact k-configuration (M, C), returns a k-location $\sigma(M, C)$ for the marshals.

Let C, C' and Z be sets of variables. We say that C is $[Z]$-*connected* to C' if there are variables $X \in C$ and $Y \in C'$ such that X is $[Z]$-*connected* to Y.

Given a strategy σ for k marshals, the *game tree* for σ is a rooted tree T whose vertices are compact k-configurations of the game, defined as follows. Capture configurations correspond to vertices having the form $(M, \emptyset)$, for some M, i.e., vertices where the robber has an empty escape-space. Moreover, escape steps are pairs of compact k-configurations $\langle (M, C), (\sigma(M, C), C') \rangle$ such that $C' \not\subset C$. The leaves of T are either configurations obtained through escape steps, or capture configurations. The root of T is $(\emptyset, V)$, because the (unique) escape space with respect to no marshals is the set of all the variables V (as we assumed that the hypertree is connected). Let (M, C) be a nonleaf vertex, $M' = \sigma(M, C)$, and R be the set of all the $[var(M')]$-*components* which are $[var(M) \cap var(M')]$-*connected* to C. If $R = \emptyset$, then (M, C) has the unique child $(M', \emptyset)$ (capture configuration); otherwise, (M, C) has a child (M', C') for each $C' \in R$. A strategy is *winning* if and only if all the leaves of its game tree are capture configurations.

Note that, for any strategy σ, there are many configurations that cannot occur at all during the game, i.e., they do not belong to the game tree for σ. Clearly enough, the value of σ on such configurations is inessential for the game. Thus, in the rest of this paper, we will explicitly define only the relevant values of any strategy σ. For each other configuration (M, C), without loss of generality, we assume the marshals do not move from their position M, i.e., we set a $\sigma(M, C) := M$.

Example 4. The strategy informally illustrated in Example 3, is a winning strategy. Precisely, its σ function is defined as follows:

$$\sigma(\emptyset, var(\mathcal{H}(Q_1))) = \{a, b\},$$
$$\sigma(\{a, b\}, \{V\}) = \{f\},$$
$$\sigma(\{a, b\}, \{W, Z\}) = \{g, c\},$$
$$\sigma(\{g, c\}, \{W\}) = \{d\}.$$

The game tree of this strategy is shown in Fig. 8.

Observe the similarity between game trees and hypertree-decomposition trees. Fig. 9 exhibits a hypertree decomposition of hypergraph $\mathcal{H}(Q_1)$ in Example 1 where, for each vertex v, we show in the figure the set of variables $\chi(v)$ and the set of edges $\lambda(v)$. There is a visible correspondence between the game tree depicted in Fig. 8 and the hypertree decomposition of Fig. 9. Moreover, note that this hypertree decomposition is in normal form, because all the conditions of Definition 2 are satisfied. (For instance, for the root r, $\chi(r) = \{P, R, S, T, U, X, Y\}$ and there are two $[r]$-*components*, namely $C_1 = \{V\}$ and $C_2 = \{W, Z\}$. Observe that C_1 and C_2 are the $[r]$-*components* associated with the left and the right children of the root, respectively, according to condition 1 in Definition 2.) Another hypertree decomposition of $\mathcal{H}(Q_1)$, represented in a different way, is

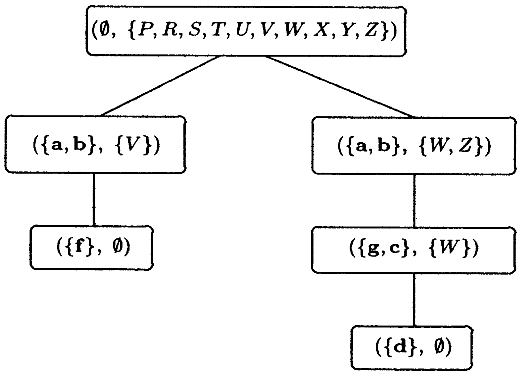


Fig. 8. The game tree of the strategy in Example 4.

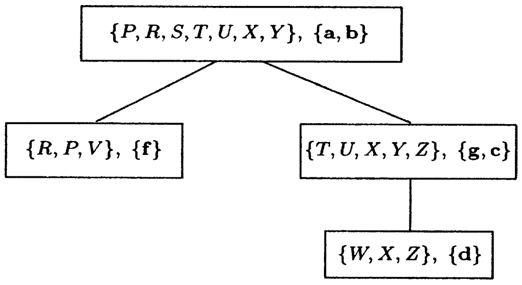


Fig. 9. A hypertree decomposition of hypergraph $\mathcal{H}(Q_1)$ in Example 1.

shown in Fig. 1b. However, this decomposition is not in normal form, because the pair of vertices composed by the rightmost leaf and its parent violate condition 2 of Definition 2.

In fact, we next prove that every game tree of a winning strategy for k marshals on a hypergraph $\mathcal{H}$ can be transformed into a legal hypertree decomposition of width k of $\mathcal{H}$ which is almost isomorphic to the game tree.

Lemma 1. *If k marshals have a winning strategy in the R&Ms game on a hypergraph $\mathcal{H}$, then $\mathcal{H}$ has hypertree width at most k.*

Proof. Assume that k marshals have a winning strategy in the R&Ms game on a hypergraph $\mathcal{H}$ and let T be the game tree for σ on $\mathcal{H}$.

From σ and T, we define a hypertree $\delta(\sigma) = \langle T', \chi, \lambda \rangle$. The tree T' of $\delta(\sigma)$ is equal to T, but it misses the last level of T. Since σ is a winning strategy, this means that T' misses all and only the capture configurations of T. Note that every game tree contains at least two vertices, because the root is not a capture configuration. Let $v = (M, C)$ be a nonleaf vertex of T, v' be the corresponding vertex in T', and $M' = \sigma(M, C)$. Then, the labels of v' are

- $\lambda(v') = M'$; and
- $\chi(v') = var(M')$, if v' is the root of T'; otherwise, $\chi(v') = var(M') \cap (\chi(u) \cup C)$, where u is the parent of v' in T'.

For instance, from the strategy in Example 4 and its game tree shown in Fig. 8, we obtain the hypertree decomposition shown in Fig. 9.

We recall from Section 2 that, for any $[V]$-component C, $edges(C)$ denotes the set of edges having some variable belonging to C and that $var(edges(C)) - C \subseteq V$ holds.

First we claim that, for any nonroot vertex $s = (M, C)$ of T,

$$var(edges(C)) - C \subseteq \chi(r'), \tag{1}$$

where r' is the vertex of T' corresponding to the parent of s in T.

To prove 1, we use structural induction on the tree T'.

Basis. This claim trivially holds for the children of the root of T', say r_0, because $\chi(r_0) = var(\lambda(r_0))$, by construction.

Induction step: Assume that (1) holds for some vertex s' of T'. Let r' be the parent of s' in T', and (M, C) the vertex of T corresponding to s'. We show that (1) holds for any child $t' = (M', C')$ of s', i.e., that $var(edges(C')) - C' \subseteq \chi(s')$. By construction, $\lambda(r') = M$ and $\lambda(s') = M'$. Since C' is a $[var(M')]$-*component*, $var(edges(C')) - C' \subseteq var(\lambda(s'))$ holds. Let $Y \in var(edges(C')) - C'$. Then, $Y \in var(edges(C))$, because $C' \subseteq C$. Therefore, either $Y \in var(edges(C)) - C$ and hence, by the induction hypothesis, $Y \in \chi(r')$, or $Y \in C$. In both cases, it follows that $Y \in \chi(s')$. Indeed, by construction, $\chi(s') = var(\lambda(s')) \cap (\chi(r') \cup C)$. This concludes the proof of the claim.

We next prove that $\delta(\sigma)$ fulfills all the properties of Definition 1 and is thus a hypertree decomposition of $\mathcal{H}$.

Property 1. $\forall h \in edges(\mathcal{H}) \exists v \in vertices(T')$ *s.t.* $h \subseteq \chi(v)$.

Let h be any edge of $\mathcal{H}$. Recall that the root of T is $(\emptyset, var(\mathcal{H}))$, i.e., the escape space for the robber is the whole set of the variables, and, trivially, $h \subseteq var(\mathcal{H})$. Then, escape spaces monotonically decrease at each step along every branch of the game tree. It follows that there exists the deepest vertex $r = (M, C)$ such that $h \cap C \neq \emptyset$ and, for each child (M', C') of $r, h \cap C' = \emptyset$. Recall that every configuration that is a child of r has the same marshals' location $M' = \sigma(M, C)$. This entails that $h \subseteq var(M')$, otherwise some $Y \in h - var(M')$ should belong to an escape space w.r.t. M' of some child of r, because any variable $X \in h \cap C$ would be $[var(M) \cap var(M')]$-*connected* to Y, through the edge h. Let $A = h - C$, and $B = h \cap C$. Note that $B \subseteq C$ and $A \subseteq var(edges(C)) - C$, since $h \in edges(C)$. Now, consider the vertex r' of T' corresponding to r. From (1), for its father t', $var(edges(C)) - C \subseteq \chi(t')$ holds. Moreover, $\lambda(r') = M'$, and we observed that $h \subseteq var(M')$. By construction, $\chi(r') = var(\lambda(r')) \cap (\chi(t') \cup C)$. Therefore, $\chi(r') \supseteq var(\lambda(r')) \cap (A \cup B)$, hence $h \subseteq \chi(r')$, and thus $\delta(\sigma)$ fulfills property 1.

Property 2. *For each variable $Y \in var(\mathcal{H})$, the set $\{v \in vertices(T') \mid Y \in \chi(v)\}$ induces a connected subtree of T'.*

Assume that property 2 does not hold. Then, there exists a variable $Y \in var(\mathcal{H})$ and two vertices v_1 and v_2 of T' such that $Y \in (\chi(v_1) \cap \chi(v_2))$ but the unique path from v_1 to v_2 in T' contains a vertex s such that $Y \notin \chi(s)$. Note that, by the construction above, Y belongs to $\chi(v')$, for some vertex v' of T', only if either Y is in the χ label of the father of v', or Y belongs to the escape space C of the vertex v of T corresponding to v'. We say that C is the escape space associated with v' (via v). W.l.o.g assume that v_1 is adjacent to s and that v_2 is a descendant of s in T', i.e., $v_2 \in vertices(T'_s)$. There are two possibilities to consider:

- v_1 *is a child of* s and v_2 belongs to the subtree T'_p of another child p of s.
  Since $Y \notin \chi(s)$, from the above discussion and the narrowing property of strategies, it follows that Y belongs to both the escape space associated with v_1 and the escape space associated with p. This is a contradiction, because the escape spaces associated with the children of any vertex are pairwise disjoint.
- s *is a child of* v_1 and v_2 belongs to the subtree T'_s of T' rooted at s.
  Again, by our construction and the discussion above, $Y \notin \chi(s)$ and $Y \in \chi(T'_s)$, entails that Y belongs to the escape space associated with s. This contradicts the narrowing property of strategies. Indeed, $Y \in \chi(v_1)$ means that the marshals are controlling this variable and thus Y does not belong to the escape space associated with v_1. However, this set should include the escape space associated with s, as $s \in T'_{v_1}$.

Property 3. $\forall p \in vertices(T')$, $\chi(p) \subseteq var(\lambda(p))$.

Follows from the definition of the χ labelling in our construction.

Property 4. $\forall p \in vertices(T')$, $var(\lambda(p)) \cap \chi(T'_p) \subseteq \chi(p)$.

Assume a vertex p in T' does not fulfill this property, and let (M, C) the label of its corresponding vertex in T. Since $\chi(p) \subseteq \chi(T'_p)$, this means that there is a vertex s in the subtree T'_p that witnesses this violation, i.e., such that $var(\lambda(p)) \cap \chi(s) \nsubseteq \chi(p)$. Without loss of generality, assume there is no other witness of this violation that is closer to p than s, and let r be the parent of s in T'_p. Thus, there is a variable $Y \in var(\lambda(p)) \cap \chi(s)$ such that $Y \notin \chi(p)$ and $Y \notin \chi(r)$. Let (M', C') be the vertex corresponding to s in T. By construction, $\chi(s) = var(\lambda(s)) \cap (\chi(r) \cup C')$. Since $Y \notin \chi(r)$, it follows that $Y \in C'$. However, T is a winning strategy, then $C' \subset C$, and hence we get $Y \in C$ that, together with $Y \in \lambda(p)$, entails $Y \in \chi(p)$, a contradiction. □

The converse also holds, i.e., each k-width hypertree decomposition in a certain form, formally defined below, gives rise to a winning strategy σ for k marshals whose game tree is very similar to the decomposition tree. To prove this result, we first need some additional notation and a normal form for hypertree decompositions.

Let $HD = \langle T, \chi, \lambda \rangle$ be a hypertree for $\mathscr{H}$. For any vertex v of T, we will often use v as a synonym of $\chi(v)$. In particular, $[v]$-*component* denotes $[X(v)]$-*component*; the term $[v]$-path is a synonym of $[\chi(v)]$-path; and so on.

Definition 2 (Gottlob et al. [16]). A hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$ of a hypergraph $\mathscr{H}$ is in *normal form* (NF) if, for each vertex $r \in vertices(T)$, and for each child s of r, all the following conditions hold:

1. there is (exactly) one $[r]$-*component* C_s such that $\chi(T_s) = C_s \cup (\chi(s) \cap \chi(r))$;
2. $\chi(s) \cap C_s \neq \emptyset$, where C_s is the $[r]$-component satisfying condition 1;
3. $var(\lambda(s)) \cap \chi(r) \subseteq \chi(s)$.

Proposition 2 (Gottlob et al. [16]). *For each k-width hypertree decomposition of a hypergraph $\mathscr{H}$ there exists a k-width hypertree decomposition of $\mathscr{H}$ in normal form.*

Intuitively, each subtree rooted at a child node s of some node r of a normal form decomposition tree serves to decompose precisely one $[r]$-*component*. We next introduce a useful notation for such components. If $HD = \langle T, \chi, \lambda \rangle$ is an NF hypertree decomposition of a conjunctive query Q, we can associate a set $treecomp(s) \subseteq var(Q)$ with each vertex s of T as follows.

- If $s = root(T)$, then $treecomp(s) = var(Q)$;
- otherwise, let r be the father of s in T; then, $treecomp(s)$ is the (unique) $[r]$-component C_s such that $\chi(T_s) = C_s \cup (\chi(s) \cap \chi(r))$.

Note that, since $s \in vertices(T_s)$, also $\chi(T_s) = C_s \cup \chi(s)$ holds.

We recall from [16] two important properties of NF hypertree decompositions.

Lemma 2 (Gottlob et al. [16]). *Let $\langle T, \chi, \lambda \rangle$ be an NF hypertree decomposition of a hypergraph $\mathscr{H}$, and r be a vertex of T. Then, $C = treecomp(s)$ for some child s of r if and only if C is an $[r]$-component of $\mathscr{H}$ and $C \subseteq treecomp(r)$.*

Lemma 3 (Gottlob et al. [16]). *Let $\langle T, \chi, \lambda \rangle$ be an NF hypertree decomposition of a hypergraph $\mathcal{H}$, s a vertex of T, and C a set of variables such that $C \subseteq treecomp(s)$. Then, C is an $[s]$-component if and only if C is a $[var(\lambda(s))]$-component.*

Lemma 4. *If a hypergraph $\mathcal{H}$ has hypertree width k, then k marshals have a winning strategy in the R&Ms game on $\mathcal{H}$.*

Proof. Let $HD = \langle T, \chi, \lambda \rangle$ be a k-width hypertree decomposition of $\mathcal{H}$. By Proposition 2, we may assume without loss of generality that $\mathcal{H}$ is in normal form. From HD we build a winning strategy σ as follows. If s is the root of T, define $\sigma(\emptyset, var(\mathcal{H})) = \lambda(s)$. Otherwise, let v be the parent of s in T and $C = treecomp(s)$. Then, define $\sigma(\lambda(v), C) = \lambda(s)$. Note that σ has been defined only on configurations of a particular form; but we will see that the other configurations are inessential, as they do not arise in the game tree of σ.

For instance, from the hypertree decomposition in normal form shown in Fig. 9, we obtain the strategy in Example 4.

Let T' be the game tree of σ. We next show that T' has precisely the same tree shape as T, but it has one further level of vertices than T, i.e., T is isomorphic to the subgraph T'_{nl} of T' which does not contain the leaves of T' (e.g., see Figs. 9 and 8.)

Claim A. *There exists a transformation γ from T to T' such that, if s is a vertex of T with parent v, then*

A1. $\gamma(s) = (\lambda(v), treecomp(s))$,
A2. *$\gamma(v)$ is the parent of $\gamma(s)$ in T', and*
A3. *if s is not a leaf of T, then s and $\gamma(s)$ have precisely the same number of children.*

Proof. We proceed by induction on the depth of the vertices of T. We inductively define γ and prove properties A1–A3:

Basis of the induction (*depth* 1): Let v and v' be the root of T and the root of T', respectively. The image $\gamma(v)$ of v under γ is obviously set to v'. By definition of game tree, $v' = (\emptyset, V)$, where $V = var(\mathcal{H})$. Since $\sigma(v') = \lambda(v)$, v' has precisely one child $s' = (\lambda(v), C)$ for each $[var(\lambda(v))]$-*component* C (as every $[var(\lambda(v))]$-*component* is $[\emptyset]$-*connected* to V). On the other hand, by Lemma 2, v has exactly one child s for each $[\chi(v)]$-*component*. Therefore, v has one child s for each $[var(\lambda(v))]$-*component*, as $\chi(v) = var(\lambda(v))$ for the root v of an hypertree decomposition (it follows immediately from item 4 of Definition 1). Therefore, v and $\gamma(v)$ have, respectively, a child s and a child s' for each $[var(\lambda(v))]$-*component*. Thus, we set $\gamma(s) = (\lambda(v), C)$, where C is the $[var(\lambda(v))]$-*component* $treecomp(s)$. Properties A1–A3 are evidently satisfied.

Induction step: Assume that properties A1–A3 hold for depth n. We prove that they hold for depth $n+1$ as well.

Let v be a vertex of T of depth n, and $v' = \gamma(v)$. By the induction hypothesis, $\gamma(v) = (\lambda(p), treecomp(v))$, where p is the parent of v. If v is a leaf of T, then we are done. Otherwise, from Lemma 2, v has exactly a child s_i for each $[\chi(v)]$-*component* C contained in $treecomp(v)$, where C is also a $[var(\lambda(v))]$-*component*, by Lemma 3.

On the other hand, from the definition of σ and from the definition of game-tree, the vertex $\gamma(v)$ has exactly a child $s_i' = (\lambda(v), C_i)$ for each $[var(\lambda(v))]$-*component* C_i which is $[var(\lambda(p)) \cap var(\lambda(v))]$-*connected* to the set of variables $treecomp(v)$. Clearly, every $[var(\lambda(v))]$-*component* contained in $treecomp(v)$ is also $[var(\lambda(p)) \cap var(\lambda(v))]$-*connected* to $treecomp(v)$ (as it contains some variable in $treecomp(v)$ which is not in $var(\lambda(v))$). We next show that also the converse holds, i.e., every $[var(\lambda(v))]$-*component* which is $[var(\lambda(p)) \cap var(\lambda(v))]$-*connected* to $treecomp(v)$ is contained in $treecomp(v)$.

By contradiction, suppose that the converse does not hold. Then, there is a $[var(\lambda(v))]$-*component* C which is $[var(\lambda(p)) \cap var(\lambda(v))]$-*connected* to $treecomp(v)$, but it is not contained in $treecomp(v)$. The edge h connecting C to $treecomp(v)$ must necessarily contain a variable X which is in $treecomp(v)$ and a variable Y of C which is either in $\lambda(p) - \lambda(v)$ or in $\lambda(v) - \lambda(p)$. Since C is a $[var(\lambda(v))]$-*component*, Y cannot be in $\lambda(v)$, and it therefore belongs to $\lambda(p) - \lambda(v)$. From property 1 of Definition 1, there is a vertex r of HD such that $h \subseteq \chi(r)$. If r belongs to the subtree of HD rooted at v, then it violates the connectedness condition (property 2 of Definition 1) for variable Y; otherwise, it violates the connectedness condition for variable X (a contradiction).

Therefore, v and $\gamma(v)$ have, respectively, a child s_i and a child s_i', for each $[var(\lambda(v))]$-*component* which is contained in $treecomp(v)$. We set $\gamma(s_i) = (\lambda(v), C)$, where C is the $[var(\lambda(v))]$-*component* $treecomp(s_i)$. Properties A1–A3 are satisfied. $\square$

By virtue of Claim A, the tree shape of the game-tree T' is equal to the shape of T, up to the leaves of T. However, if s' is the image $\gamma(s)$ of a leaf s of T, then Claim A does not say anything about its possible children. We next show that the leaves of T' are children of the images under γ of leaves of T.

Let l be a leaf of T with parent v, and let $\gamma(l) = (\lambda(v), treecomp(l))$ be its image. Since l is a leaf of T, $treecomp(l)$ is contained in $var(\lambda(l))$. Therefore, the child of $\gamma(l)$ in T' is $(\lambda(l), \emptyset)$, and it is a leaf of T'.

Hence, every leaf of T' is a capture configuration, and σ is a winning strategy for k marshals on $\mathcal{H}$. $\square$

Lemmas 1 and 4 imply the following theorem, which states the relationship between the R&Ms game played on a hypergraph and its hypertree decompositions.

Theorem 1. *A hypergraph $\mathcal{H}$ has hypertree width at most k if and only if k marshals have a winning strategy in the R&Ms game on $\mathcal{H}$.*

Since deciding whether a hypergraph has bounded hypertree width is in LOGCFL [16], we immediately get the following upper bound on the complexity of playing the R&Ms game.

Corollary 1. *Let k be a fixed constant. Deciding whether k marshals have a winning strategy in the R&Ms game on a hypergraph $\mathcal{H}$ is in* LOGCFL.

It is worthwhile noting that, from the equivalence between acyclic and hypertree-width-one hypergraphs, we get a nice game characterization of acyclic hypergraphs.

Corollary 2. *A hypergraph $\mathcal{H}$ is acyclic if and only if one marshal has a winning strategy in the R&Ms game on $\mathcal{H}$.*

4. Logical characterization

In this section we give a logical characterization of queries of bounded hypertree width. We show that, for each constant k, a simple and appealing fragment $GF_k(L)$ of FO precisely captures the class $HW[k]$ of queries whose hypertree width is bounded by k. For simplicity, we limit our attention here to *Boolean* queries and, accordingly, to *closed* first-order formulas. However, our characterization extends to non-Boolean queries and to formulas with free variables.

Definition 3. We denote by L the fragment of FO sentences that are built from query atoms, existential quantifiers, and conjunctions. Moreover, we denote by ACYCLIC the class of all acyclic conjunctive queries, and by $HW[k]$ the class of all conjunctive queries having hypertree width at most k.

Note that Kolaitis and Vardi [23] denoted the fragment L by $\exists FO_{\wedge,+}$. They pointed out that the expressive power of this fragment is the same as the expressive power of conjunctive queries. We next generalize the well-known notion of guarded formulas to the case where, rather than just one guard, formulas may have up to k guards, for some $k>0$.

Definition 4. Let $\mathcal{L}$ be a logic which is either FO, or a restriction thereof. Then, the set of the k-guarded formulas of $\mathcal{L}$ is the smallest set of formulas $G_k(\mathcal{L})$ such that

- every atom of $\mathcal{L}$ belongs to $G_k(\mathcal{L})$;
- let $\odot$ be any binary connective of $\mathcal{L}$ and let $\varphi_1, \varphi_2 \in G_k(\mathcal{L})$. Then, $\varphi_1 \odot \varphi_2 \in G_k(\mathcal{L})$;
- if $\neg$ is a connective of $\mathcal{L}$ and $\varphi \in G_k(\mathcal{L})$, then $\neg\varphi \in G_k(\mathcal{L})$;
- let $\alpha_1, \dots, \alpha_i$ be atoms, with $1 \leqslant i \leqslant k$, and let φ be a formula in $G_k(\mathcal{L})$. If $free(\varphi) \subseteq var(\alpha_1) \cup \cdots \cup var(\alpha_i)$, then, for each tuple of variables $\bar{y}$, the formula $\varphi' : \exists\bar{y}(\alpha_1 \wedge \cdots \wedge \alpha_i \wedge \varphi)$ belongs to $G_k(\mathcal{L})$. Moreover, the set of atoms $\{\alpha_1, \dots, \alpha_i\}$ is referred to as the guard of φ' and is denoted by $guard(\varphi')$.[3]

The k-guarded fragment $GF_k(\mathcal{L})$ consists of all k-guarded sentences of $\mathcal{L}$. For $k = 1$, we also write $GF(\mathcal{L})$ and simply refer to it as the guarded fragment of $\mathcal{L}$.

Example 5. Consider the following formula Φ, equivalent to query Q_1 of Section 1:

$$\begin{aligned}\Phi = {} & \exists S, X, T, R, Y, U, P(a(S, X, T, R) \wedge b(S, Y, U, P) \wedge (\exists V f(R, P, V)) \\ & \wedge (\exists Z(g(X, Y) \wedge c(T, U, Z) \wedge (\exists W d(W, X, Z)) \wedge e(Y, Z))))\end{aligned}$$

[3] Note that, given that we omit parentheses in conjunctions of multiple atoms, for some formulas ψ, $guard(\psi)$ may not be uniquely determined because *several* leading conjunctions of atoms may cover the free variables in the remaining conjuncts. In case of such an ambiguity we may always choose the smallest such sequence as the guard $guard(\varphi')$. Moreover, we extend in the obvious way the last item of Definition 4 to the trivial case where φ is empty and corresponds to the truth constant *true* and is actually not written, i.e., where φ' is simply $\exists\bar{y}(\alpha_1 \wedge \cdots \wedge \alpha_i)$.

The formula Φ is a 2-guarded formula, i.e., $\Phi \in \mathrm{GF}_2(\mathrm{L})$. The guard of formula Φ is $guard(\Phi) = \{a(S,X,T,R),\ b(S,Y,U,P)\}$. For the inner existential formulas $\psi_1 = \exists V f(R,P,V)$, $\psi_2 = \exists Z(g(X,Y) \wedge c(T,U,Z) \wedge (\exists W d(W,X,Z)) \wedge e(Y,Z))$ and $\psi_3 = \exists W d(W,X,Z)$, we have the following guards: $guard(\psi_1) = \{f(R,P,V)\}$, $guard(\psi_2) = \{g(X,Y), c(T,U,Z)\}$, and $guard(\psi_3) = \{d(W,X,Z)\}$.

Definition 5. A formula is *straight* if

1. each of its bound variables is quantified over only once, and
2. each quantified variable occurs in some atom within the scope of its quantifier.

Lemma 5. *Every formula in* $GF_k(\mathrm{L})$ *is equivalent to some straight formula of* $GF_k(\mathrm{L})$.

Proof (Sketch). To achieve condition 1, it is sufficient to suitably rename variables which are multiply quantified. To achieve condition 2, it is sufficient to remove vacuous quantifications. It is easy to see that these adjustments preserve k-guardedness. □

Definition 6. Let $\Phi \in GF_k(\mathrm{L})$ be a straight formula. The conjunctive query Q_Φ associated with Φ is the query "$ans \leftarrow body_\Phi$", where $body_\Phi$ consists of the conjunction of all atoms occurring in Φ.

Example 6. The query Q_Φ associated with the 2-guarded straight formula Φ of Example 5 is the query Q_1 of Example 3.

Lemma 6. *Let* $\Phi \in \mathrm{L}$ *be a straight sentence. Then* $Q_\Phi \equiv \Phi$, *i.e., for each database* **DB**, $\mathbf{DB} \models \Phi$ *if and only if* $\mathbf{DB} \models Q_\Phi$.

Proof (Sketch). It suffices to move all quantifiers of Φ in front. We get an equivalent formula in prenex form which is in turn equivalent to the conjunctive query Q_Φ. □

Notation. For straight sentences $\Phi \in \mathrm{L}$, $\mathscr{H}(\Phi)$ denotes the hypergraph $\mathscr{H}(Q_\Phi)$.

Definition 7. A straight sentence $\Phi \in \mathrm{L}$ is connected if $\mathscr{H}(\Phi)$ is connected.[4] Then, Φ is disconnected if $\mathscr{H}(\Phi)$ consists of more than one connected components.

Lemma 7. *Let* $\Phi \in GF_k(\mathrm{L})$ *be a disconnected straight sentence whose hypergraph* $\mathscr{H}(\Phi)$ *has* r *connected components* $C_1, \ldots, C_r$. *Let* $Q_1, \ldots, Q_r$ *be the Boolean conjunctive queries corresponding to these components. Then there exist connected sentences* $\Phi_1, \ldots, \Phi_r \in GF_k(\mathrm{L})$ *such that* $\Phi \equiv \Phi_1 \wedge \Phi_2 \wedge \cdots \wedge \Phi_r$ *and, for* $1 \leqslant i \leqslant r$, $Q_{\Phi_i} = Q_i$. *Moreover,* $hw(Q_\Phi) = \max\{hw(Q_i) \mid 1 \leqslant i \leqslant r\}$.

[4] Here, we refer to the standard notion of component in hypergraph theory, which corresponds to the notion of $[\emptyset]$-*component* defined in Section 2.

Proof. For $1 \leqslant i \leqslant r$, obtain Φ_i from Φ by simply eliminating all atoms not belonging to component C_i, and by eliminating all quantifiers over variables outside C_i. Each formula Φ_i is still k-guarded: all relevant guards for the remaining variables have their variables in C_i and were thus not removed. By definition of Φ_i it holds that $Q_{\Phi_i} = Q_i$. Moreover, by repeated application of the rule

$$\exists \mathbf{xy}(\Psi_1(\mathbf{x}) \wedge \Psi_2(\mathbf{y})) \equiv [(\exists \mathbf{x} \Psi_1(\mathbf{x})) \wedge (\exists \mathbf{y} \Psi_2(\mathbf{y}))]$$

which is valid for disjoint lists of variables $\mathbf{x}$ and $\mathbf{y}$, we get that $\Phi \equiv \Phi_1 \wedge \Phi_2 \wedge \cdots \wedge \Phi_r$. Finally, the hypertree width of any hypergraph consisting of the union of a number of connected components is (by the definition of hypertree width) equal to the maximum of the hypertree widths of the components. This applies in particular to $\mathscr{H}(\Phi) = \mathscr{H}(Q_\Phi) = \bigcup_{1 \leqslant i \leqslant r} \mathscr{H}(Q_i)$, and thus $hw(Q_\Phi) = \max\{hw(Q_i) \mid 1 \leqslant i \leqslant r\}$. □

Definition 8. Let Φ be a straight sentence in $GF_k(\mathrm{L})$ and let C be a set of variables occurring in Φ. Then we denote by $\Gamma_\Phi(C)$ the smallest guarded subformula of Φ containing all quantifiers that quantify over the variables in C.

The following lemma follows immediately from Definition 8.

Lemma 8. *Let Φ be a straight sentence in $GF_k(\mathrm{L})$, let M and M' be sets of edges from $\mathscr{H}(\Phi)$, and let C be a $[var(M)]$-component and C' a $[var(M')]$-component. If $C' \subseteq C$, then $\Gamma_\Phi(C')$ is a subformula of $\Gamma_\Phi(C)$.*

Lemma 9. *Let Φ be a straight sentence in $GF_k(\mathrm{L})$, where $\mathscr{H}(\Phi)$ is connected. Let C be a $[var(M)]$-component of $\mathscr{H}(\Phi)$, where $M \subseteq edges(\mathscr{H}(\Phi))$. Then $\Gamma_\Phi(C)$ is an existential subformula of Φ.*

Proof. First note that if an atom A contains some variable from C then A must occur within $\Gamma_\Phi(C)$ and cannot occur in Φ outside $\Gamma_\Phi(C)$ because Φ is a sentence and hence all its variables are quantified.

Towards a contradiction assume that $\Gamma_\Phi(C)$ is a conjunctive subformula $\alpha \wedge \beta$ of Φ. Note that α and β partition the variables of C. Then, some variable of C, say x, must occur in α and not in β, and some variable y must occur in β but not in α. (This follows from both the fact that every variable is quantified only once in Φ and that $\Gamma_\Phi(C)$ is minimal.) However, x and y are both in C and are thus $[var(M)]$-*connected*, hence there is a chain of variables $x = x_1, x_2, \ldots, x_r = y$ in C and there are atoms $A_1[x_1, x_2], A_2[x_2, x_3], \ldots, A_{r-1}[x_{r-1}, x_r]$ of Φ, where $A[u, v]$ means that u and v both occur in A. But then one of these atoms $A_i[x_i, x_{i+1}]$ must be such that x_i occurs only in α and x_{i+1} occurs only in β. This, however, is impossible, because then the atom $A_i[x_i, x_{i+1}]$ could neither belong to α nor to β. Contradiction. The claim is proved. □

Definition 9. A strong hypertree decomposition of a hypergraph H is a hypertree decomposition $\langle T, \chi, \lambda \rangle$ of H such that, for each vertex v of T, $\chi(v) = var(\lambda(v))$. The strong hypertree width $strhw(H)$ of H is the smallest integer h such that there exists a strong hypertree decomposition of width h of H. Accordingly, if H is a hypergraph corresponding to a query Q, then we define

$strhw(Q) := strhw(H)$. The set of all conjunctive queries Q such that $strhw(Q) \leqslant k$ is denoted by $STRHW[k]$.

Lemma 10. *$STRHW[k] = HW[k]$, i.e., for each conjunctive query Q there exists an equivalent conjunctive query Q' such that $strhw(Q') = hw(Q)$, and vice-versa.*

Proof. The direction $STRHW[k] \subseteq HW[k]$ follows trivially from the definition of strong hypertree width. Let us show that $HW[k] \subseteq STRHW[k]$. Let Q be a query and let $D = \langle T = (V, E), \chi, \lambda \rangle$ be a hypertree decomposition of $\mathscr{H}(Q)$ of width $h \leqslant k$ which is not strong. By Proposition 1, we may assume, without loss of generality, that D is a complete hypertree decomposition.

Since D is not a strong hypertree decomposition, there are vertices $p \in T$ such that some variable $X \in \lambda(p)$ does not belong to $\chi(p)$. We next show that D can be transformed into a width h strong hypertree decomposition of a query Q' equivalent to Q. Intuitively, for each vertex containing such problematic variables, the new query Q' contains a new atom where these variables are suitably replaced by fresh ones.

Note that a hyperedge e may occur in the λ-labels of multiple nodes. A *hyperedge occurrence* can be described as a pair (e, v), where e is a hyperedge, v is a vertex of T, and $e \in \lambda(v)$. Let Occ denote the set of *all* hyperedge occurrences in D. We assume w.l.o.g. that there is a function $f : Occ \rightarrow atoms(Q)$ such that $\bigcup_{\alpha \in Occ} f(\alpha) = atoms(Q)$. In fact, if such a function does not exist, then we can transform as follows the complete decomposition D to a complete decomposition D^* of the same width, for which such a function exists. First associate with each occurrence (e, v) all query atoms A such that $var(A) = e$. Note that each occurrence has at least one associated atom and all of $atoms(Q)$ is covered this way; however, to some occurrences, more than one query atoms may be associated. For each occurrence (e, v) which has only one associated query atom A, let $f(e, v) := A$. For each occurrence (e, v) to which several atoms are associated, do the following. Choose one particular associated atom A among this set and let $f(e, v) := A$. For each atom $B \neq A$ associated with (e, v), create a new child v_B of v, let $\lambda(v_B) := \{var(B)\} = \{e\}$ and $\chi(v_B) = \chi(v) \cap var(B)$, and define $f(e, v_B) := B$. It is obvious that f is a function with the desired property, and that D^* is a hypertree decomposition of $\mathscr{H}$ having the same width as D (we just added leaves with singleton λ-labels to D).

We transform Q to Q' and D to $D' = \langle T, \chi', \lambda' \rangle$ such that $Q \equiv Q'$ and D' is a strong hypertree decomposition of $\mathscr{H}(Q')$.

For each vertex v of T,

- $\lambda'(v)$ is obtained from $\lambda(v)$ by replacing, in every edge $e \in \lambda(v)$, each variable of $var(\lambda(v)) - \chi(v)$ by a new variable not occurring anywhere else. (*Note*: This replacement is done in the label $\lambda(v)$ only, not in the query itself.) Denote the corresponding variable substitution by ϑ_v. We thus have $\lambda'(v) = \{e\vartheta_v \mid e \in \lambda(v)\}$.
- $\chi'(v) := var(\lambda'(v))$.

We furthermore define $head(Q') = head(Q)$ and

$$body(Q') := \bigcup_{v \in T} \{f(e, v)\vartheta_v \mid e \in \lambda(v)\}.$$

Note that $atoms(Q) \subseteq atoms(Q')$. In fact, by the completeness of decomposition D, each atom of Q is entirely covered by $\chi(v)$ for some vertex v of T and thus survives while $\lambda(v)$ is transformed into $\lambda'(v)$. It is not hard to see that the resulting Q' is equivalent to Q and that D' is a correct strong hypertree decomposition for Q' having the same width h as T. $\square$

Remark 2. At a first glance, the above lemma may suggest that one could give up on general hypertree decompositions and concentrate on strong hypertree decompositions instead, given that $STRHW[k] = HW[k]$. We have two good reasons for not doing so. First, the equality $STRHW[k] = HW[k]$ is valid at the semantical query level only, but not at the syntactic level. For many conjunctive queries Q, it holds that $strhw(Q) > hw(Q)$. Second, it can be seen that the problem of determining whether $strhw(Q) \leqslant k$ is NP-hard for fixed constants $k \geqslant 4$. (This is implicit in our proof for the NP-hardness of checking for bounded query width in [16].)

We next provide a very natural logical characterization of conjunctive queries having hypertree width at most k in terms of k-guarded formulas. For two query languages or logics L_1 and L_2, we write $L_1 = L_2$ in order to express that L_1 and L_2 allow one to formulate exactly the same class of queries, where a query is defined semantically as a mapping from finite structures to finite structures (or to $\{true, \ false\}$, if we restrict ourselves to Boolean queries). The equality $L_1 = L_2$ is thus a semantic equality and does *not* mean that L_1 and L_2 express the same syntactic queries, nor that the syntactic queries expressible in L_1 can be efficiently translated into equivalent syntactic queries of L_2 or vice versa.

Theorem 2. $HW[k] = GF_k(\mathrm{L})$.

Proof. We first show that $GF_k(\mathrm{L}) \subseteq HW[k]$. Let $\Phi \in GF_k(\mathrm{L})$ be a k-guarded sentence. By virtue of Lemma 5 we assume w.l.o.g. that Φ is straight. By Lemma 6, Φ is equivalent to the conjunctive query Q_Φ. It thus suffices to prove that $Q_\Phi \in HW[k]$. If $\mathcal{H}(\Phi)$ is not connected, then, by Lemma 7, proving that $Q_\Phi \in HW[k]$ amounts to proving that, for a number of formulas $\Phi_i \in GF_k(\mathrm{L})$ whose hypergraph $\mathcal{H}(\Phi_i)$ is connected, it holds that $Q_{\Phi_i} \in HW[k]$. Therefore, we may assume w.l.o.g. that Φ is connected.

To prove that $Q_\Phi \in HW[k]$, we play the robber and marshals game on the hypergraph $\mathcal{H}(\Phi) = \mathcal{H}(Q_\Phi)$.

In particular, we show that there is a winning strategy for k marshals on this hypergraph.

Note that, since $\mathcal{H}(\Phi)$ is connected, $var(\Phi)$ is a $[\emptyset]$-*component*. By Lemma 9, $\Phi = \Gamma_\Phi(var(\Phi))$ itself must be an existential formula.

Let us define a strategy σ as follows:

If M is a position for the marshals and C is an $[M]$-component, then $\sigma(M, C) = guard^*(\Gamma_\Phi(C))$, where, for each existentially quantified k-guarded subformula Ψ of Φ, $guard^*(\Psi)$ denotes the set of hyperedges of $\mathcal{H}(\Phi)$ that correspond to the atoms of $guard(\Psi)$.

Note that, as defined in Section 3, the game always begins with the k-configuration $(\emptyset, V)$, where $V = var(\Phi)$ is the set of variables of the hypergraph. Thus the starting move of σ is well determined by $\sigma(\emptyset, V) = guard^*(\Phi)$. Moreover, at any time of the game, for every node (M, C), by Lemma 9, the formula $\Gamma_\Phi(C)$ is an existential subformula of Φ.

In order to show that σ is a winning strategy for k marshals on $\mathscr{H}(\Phi)$, it suffices to show by induction that each nonleaf vertex (M, C) of the game tree T w.r.t. strategy σ fulfills the following property $\Pi(M, C)$:

$\Pi(M, C)$: For each child (M', C') of (M, C) in T, it holds that $C' \subset C$.

Indeed, this property entails that there are no escape steps and that, eventually, each branch ends up in a capture configuration. To prove Π, we use induction on the depth $\delta(M, C)$ of (M, C) in the tree T (where the root has depth 0).

Induction basis: $\delta(M, C) = 0$. In this case, we have $M = \emptyset$ and $C = V = \mathit{var}(\Phi)$. The formula Φ is of the form $\exists \mathbf{x}, \mathbf{y}(\mathit{guard}(\Phi) \wedge \Psi)$, where $\mathbf{x} = \mathit{var}(\mathit{guard}(\Phi))$ is a nonempty list of variables (for otherwise (M, C) would be a leaf of T), and $\mathbf{y}$ is a possibly empty list of superfluous variables not occurring anywhere else. Then, $M' = \sigma(\emptyset, V) = \mathit{guard}^*(\Phi)$. Thus, by the definition of game tree, the children of the root $(\emptyset, V)$ are all the vertices $(\mathit{guard}^*(\Phi), C')$ where C' is a $[\mathit{var}(\mathit{guard}^*(\Phi))]$-*component* of $\mathscr{H}(\Phi)$. All occurrences of the variables of C' are entirely inside the guarded formula Ψ, therefore $C' \subseteq C$. Moreover, the variables in $\mathbf{x}$ belongs to $\mathit{guard}^*(\Phi)$ and hence do not belong to C'. Therefore $C' \subset C$.

Induction step: Assume that property Π holds for all non-leaf vertices of T at depth $i - 1 \geqslant 0$. Let (M, C) be a nonleaf vertex at depth i and let (M_0, C_0) be its parent node in T. Let (M', C') be a child of (M, C) in T. We have to prove that $C' \subset C$. If C' is the empty escape space, then we are done. Otherwise, C' is a $[\mathit{var}(M')]$-*component*, where $M' = \sigma(M, C) = \mathit{guard}^*(\Gamma_\Phi(C))$.

Let us first show that $C' \subseteq C$. To show this, it suffices to prove that during the move $M \rightarrow M'$ of the marshals, the robber is unable to escape outside C. Towards a contradiction, assume the contrary. Then, there are a variable $t \in C' - C$ and a variable $s \in C$ such that there is a $[\mathit{var}(M) \cap \mathit{var}(M')]$-*path* from s to t. Note that $t \notin C$, hence s is not $[\mathit{var}(M)]$-*connected* to t. It follows that there exist a variable $z \in \mathit{var}(M) - \mathit{var}(M')$ and a variable $u \in C$ such that:

- t is $[\mathit{var}(M) \cap \mathit{var}(M')]$-*connected* to z, and
- u and z both occur in some atom $A[u, z]$ of $Q(\Phi)$ (possibly with other variables).

Intuitively, during the move $M \rightarrow M'$ of the marshals, the robber can leave C, going from s to t by crossing the "bridge" atom $A[u, z]$.

Consider the subformula $\Gamma_\Phi(C)$ of Φ. This subformula is of the form $(\exists \bar{y})(G \wedge \Psi)$ where G is the guard of $\Gamma_\Phi(C)$, i.e., G consists of the conjunction of all atoms of the set $\mathit{guard}(\Gamma_\Phi(C))$. Recall that the set $\mathit{var}(G) = \mathit{var}(M')$ contains all free variables of Ψ.

Assume $z \in \mathit{free}(\Psi)$. Then z must also belong to $\mathit{var}(G) = \mathit{var}(M')$ and thus $z \in \mathit{var}(M) \cap \mathit{var}(M')$ which is impossible, given that $z \in \mathit{var}(M) - \mathit{var}(M')$.

Now assume that z belongs to the set $\mathit{bound}(\Psi)$ of bound variables of Ψ. By the induction hypothesis we have $C \subset C_0$ and therefore, by Lemma 8, $\Gamma_\Phi(C)$ is a subformula of $\Gamma_\Phi(C_0)$. Since z appears in $M = \mathit{guard}^*(\Gamma_\Phi(C_0))$ but does not appear in $M' = \mathit{guard}^*(\Gamma_\Phi(C))$, the formula $\Gamma_\Phi(C)$ must be a *proper* subformula of $\Gamma_\Phi(C_0)$. It follows that Ψ is a proper subformula of $\Gamma_\Phi(C_0)$. Our assumption that $z \in \mathit{bound}(\Psi)$ implies that z cannot occur outside Ψ; however, it does occur in $M = \mathit{guard}^*(\Gamma_\Phi(C_0))$, and $\mathit{guard}(\Gamma_\Phi(C_0))$ *is* outside Ψ. Therefore, z cannot belong to $\mathit{bound}(\Psi)$.

Given that z is neither in $\mathit{free}(\Psi)$ nor in $\mathit{bound}(\Psi)$ and does not even occur in the guard of $\Gamma_\Phi(C)$, the atom $A[u, z]$ must occur somewhere in Φ outside the subformula $\Gamma_\Phi(C)$. But this is impossible: The variable u belongs to C and is, by definition of $\Gamma_\Phi(C)$, existentially quantified within the subformula $\Gamma_\Phi(C)$. Given that Φ is a straight sentence, u cannot appear in Φ outside of the subformula $\Gamma_\Phi(C)$.

In summary, we have refuted the assumption that during the move $M \to M'$ the robber can escape outside C, and we have thus proven that $C' \subseteq C$.

To prove $\Pi(M, C)$, it remains to show that this inclusion is proper, i.e., that $C' \subset C$. By definition, $\Gamma_\Phi(C)$ is the *smallest* guarded subformula of Φ that quantifies over all variables from C. We have $\Gamma_\Phi(C) = (\exists \bar{y})(G \wedge \Psi)$ where G is the guard and Ψ is a guarded subformula. Assume that no variable from C appears in the quantifier prefix of $\Gamma_\Phi(C)$. Then Ψ is a guarded subformula containing all quantifiers for the variables of C. This contradicts the minimality of $\Gamma_\Phi(C)$. It follows that at least for one variable $x \in C$, a quantification '$\exists x$' appears in the quantifier prefix of $\Gamma_\Phi(C)$. Given that $x \in C$, some atom $A[x]$ in which x appears must occur in $\Gamma_\Phi(C)$. If $A[x]$ occurs in Ψ, then it occurs freely in Ψ and therefore x must also occur in the guard G. If $A[x]$ does not occur in Ψ, it must be one of the guard atoms of G. Thus, in any case, the variable x must occur in the guard G. But $var(M') = var(G)$, and therefore x occurs in M' and cannot occur in C'. Thus $C' \subset C$. This concludes the proof that $GF_k(\mathrm{L}) \subseteq HW[k]$.

We now show that $HW[k] \subseteq GF_k(\mathrm{L})$. Let Q be a conjunctive query and let $\langle T, \chi, \lambda \rangle$ be a width k hypertree decomposition of $\mathcal{H}(Q)$. By Proposition 1, we may assume, without loss of generality, that $\langle T, \chi, \lambda \rangle$ is a complete hypertree decomposition. Moreover, by Lemma 10 and its proof, we may assume, without loss of generality, that it is a *strong* hypertree decomposition, i.e., for each vertex $v \in vertices(T)$, it holds that $var(\lambda(v)) = \chi(v)$.

We construct from Q an equivalent formula $\Phi_Q \in GF_k(\mathrm{L})$ as follows. The root r_0 of T corresponds to the entire formula $\Phi_Q = form(r_0)$ and each vertex v of T corresponds to a subformula $form(v)$ of Φ_Q such that the composition of Φ_Q from these subformulas exactly corresponds to the structure of T.

In particular, for each vertex v of T, define $form(v) = (\exists \bar{y})(G \wedge \Psi)$, where

- $\exists \bar{y}$ is a quantifier prefix that existentially quantifies over all variables in $var(\lambda(v))$ that do not already appear in $var(\lambda(u))$ for some predecessor u of v in T;
- G consists of the conjunction $\bigwedge_{e \in \lambda(v)} first(e)$, where $first(e)$ is the lexicographically first atom A of Q such that $var(A) = e$; and
- Ψ is the conjunction of all other atoms A of Q such that $var(A) \subseteq var(\lambda(v))$ and of $\bigwedge_{w \in ch(v)} form(w)$, where $ch(v)$ denotes the set of the children of v in T.

It is very easy to see that $\Phi_Q = form(r_0)$ is in $GF_k(\mathrm{L})$ and is equivalent to Q. In particular, the guardedness of Φ_Q follows immediately from the fact that the decomposition $\langle T, \chi, \lambda \rangle$ satisfies the connectedness condition for variables. □

Note that, as recently observed by J. Flum (pers. comm.), and independently noticed by ourselves, a direct proof of the relationship $GF_k(\mathrm{L}) \subseteq HW[k]$ is also possible. This direct proof, rather than recurring to the robber and marshals game, directly constructs a strong hypertree decomposition from a straight $GF_k(\mathrm{L})$ sentence. In essence, the direct proof takes the k-guards of the formula as λ-labels of a strong hypertree decomposition, starting with the main (i.e., outer level) guard $guard(\Phi)$, and recursively adding a child labelled by $guard(\Psi)$ for each guarded subformulas Ψ of the next nesting level. It then remains to show that the constructed labelled tree is effectively a hypertree decomposition. Both proofs are very similar; in fact, the winning strategy σ defined in the proof of Theorem 2 and the hypertree decomposition obtained in the sketched direct proof correspond to each other. We used the game-theoretic proof in order to exhibit robbers and marshals "at work".

Not surprisingly, given the relationship between acyclic queries and bounded hypertree-width queries, the above result also gives a very simple yet elegant logical characterization of acyclic conjunctive queries.

Corollary 3. ACYCLIC $= GF(\mathrm{L})$.

This corollary is reminiscent of two recent investigations that connect guardedness with acyclicity in a more general context, dealing with full first-order logic FO rather than with the fragment L.

In [12], a translation from the alternation-free fragment of fixpoint logic into a version of Datalog called Datalog LITE is given. Restricting the domain of this translation to $GF(FO)$ yields a translation from $GF(FO)$ to a form of acyclic Datalog.

A stronger, more explicit, and bidirectional connection between guardedness and acyclicity at the full first-order level was given in a recent paper by Flum et al. [10]. Roughly speaking, they proved that the guarded fragment $GF(FO)$ of first-order logic is equivalent to the class of nonrecursive stratified Datalog programs having acyclic rules. Note that Corollary 3 is not directly covered as a special case of their result, because a guarded sentence of L with nested quantified subformulas is translated in their approach (similarly as in [12]) into a Datalog program with *several* acyclic rules. However, by suitable modifications of the translation, specifically, by converting the resulting Datalog program into a single acyclic query, one obtains an alternative way of proving Corollary 3.

5. On K-guarded first-order queries and datalog

Inspired by the interesting work of Flum et al. [10], we show in this section that the correspondence between k-guarded formulas and queries of k-bounded hypertree width can be lifted to the more expressive contexts of full first-order logic and stratified Datalog, respectively.

We assume the reader to be familiar with the basics of Datalog. A literal is an atom (positive literal) or a negated atom (negative literal). As usual in deductive databases, we consider safe Datalog programs, i.e., programs where, for each rule $H \leftarrow B$, every variable occurring either in the head H, or in some negated literal in the body B, must occur also in some positive literal in B. We denote the set of literals of the body B by $lit(B)$. Predicates defined by the rules of a Datalog program are called *intensional database* (*IDB*) predicates, while predicates corresponding to input database relations are called *extensional database* (*EDB*) predicates. Accordingly, atoms with IDB (resp., EDB) predicate symbols are called IDB (resp., EDB) atoms.

For brevity, we will restrict our attention to closed FO sentences and Boolean Datalog queries, but the following results can be generalized to queries with output in a similar way as done in [10]. A Boolean Datalog program P is a Datalog program jointly specified with some Boolean "output" atom p which occurs in one or more of its rule heads. We say that the program P evaluates to *true* over a database **DB** if p can be derived from **DB** via P.

Given a set S of literals, $\mathcal{H}(S)$ is the hypergraph whose set of edges is $\{var(L) \mid L \in S\}$. We denote the class of all nonrecursive stratified (safe) Boolean Datalog programs by *NRS-DATALOG*.

Definition 10. Let $r : head \leftarrow body$, be a Datalog rule, possibly containing negative literals, and $HD = \langle T, \chi, \lambda \rangle$ a hypertree decomposition of hypergraph $\mathcal{H}(lit(body))$. We say that a vertex v of T is an EDB vertex if for every hyperedge $e \in \lambda(v)$ there is a (positive) extensional database atom $A \in lit(body)$ such that $var(A) = e$. The hypertree HD is a rule decomposition of r if, for each non-EDB vertex v in T, there exists an EDB vertex w such that $\chi(v) \subseteq \chi(w)$. The hypertree HD is a strict rule decomposition if there exists a vertex v_0 of T such that $var(head) \subseteq \chi(v_0)$. The (strict) hypertree width of r is the smallest width over its (strict) rule decompositions.

NRS-DATALOG$[k]$ denotes the class of all NRS-DATALOG programs whose rules have strict hypertree-width bounded by k.

Example 7. Consider the Boolean Datalog program P_1 consisting of the two following rules:

$$\begin{aligned} r_1: \quad & ans \leftarrow a(S, X, X', C, F) \wedge b(S, Y, Y', C', F') \wedge c(C, C', Z) \\ & \wedge \neg d(X, Z) \wedge \neg e(Y, Z) \wedge f(F, F', Z') \wedge \neg g(X', Z') \\ & \wedge \neg h(Y', Z') \wedge j(J, X, Y, X', Y') \\ r_2: \quad & e(Y, Z) \leftarrow a'(S, X, T, R) \wedge b'(S, Y, U, P) \wedge f'(R, P, V) \\ & \wedge g'(X, Y) \wedge c'(T, U, Z) \wedge d'(W, X, Z) \wedge \neg e'(Y, Z) \end{aligned}$$

All the predicates are EDB predicates but *ans* and *e*. This program belongs to *NRS-DATALOG* [2], because both r_1 and r_2 have strict hypertree width 2. Indeed, the hypertree shown in Fig. 3 is a width 2 rule decomposition for r_1, because it is a width 2 hypertree decomposition of the hypergraph associated with the body of r_1 and, for each non-EDB vertex v (any leaf of the tree in the figure), its parent w is an EDB vertex and $\chi(v) \subseteq \chi(w)$. For instance, consider the leaf v_e with $\chi(v_e) = \{Y, Z\}$, covering the hyperedge corresponding to the literal $\neg e(Y, Z)$. The χ labelling of its parent w_e is $\chi(w_e) = \{X, Y, C, C', Z\}$, and clearly $\chi(v_e) \subseteq \chi(w_e)$. Moreover, no variable occurs in the head of r_1, and thus this rule decomposition is trivially strict.

For rule r_2, note that the hypergraph associated with its body is the same as the hypergraph shown in Fig. 1a, and it is easy to verify that the width 2 hypertree decomposition shown in Fig. 1b for this hypergraph is, in fact, a rule decomposition for r_2 (apart from the name of the literals of r_2, that are primed). Finally, note that this rule decomposition is strict, because the set $\{Y, Z\}$ of the variables occurring in the head of r_2 is included in the label $\chi(v_h) = \{X, Y, T, Z, U\}$ of the rightmost child v_h of the root of the decomposition.

Definition 11. k-guarded NRS-DATALOG is the subclass of NRS-DATALOG where every rule body has at most k EDB atoms that act as guards, i.e., all variables occurring in the rule occur in these atoms.

Example 8. The program P_2 consisting of the following rules is in 2-guarded NRS-DATALOG:

$$\begin{array}{lrl}
r_1: & ans & \leftarrow j(J,X,Y,X',Y') \wedge hd_1(X,Y,X',Y') \\
r_2: & hd_1(X,Y,X',Y') & \leftarrow a(S,X,X',C,F) \wedge b(S,Y,Y',C',F') \\
 & & \quad \wedge hd_2(X,Y,C,C') \wedge hd_3(X'Y',F,F') \\
r_3: & hd_2(X,Y,C,C') & \leftarrow c(C,C',Z) \wedge j(J,X,Y,X',Y') \wedge \neg d(X,Z) \wedge \neg e(Y,Z) \\
r_4: & hd_3(X',Y',F,F') & \leftarrow f(F,F',Z') \wedge j(J,X,Y,X',Y') \wedge \neg g(X',Z') \wedge \neg h(Y',Z')
\end{array}$$

In this program, the IDB predicates are *ans*, hd_1, hd_2, and hd_3. All the others are EDB predicates. Note that in the first rule the guard is the atom $j(J,X,Y,X',Y')$, while in each of r_2, r_3, and r_4, the first two atoms act as guards.

Theorem 3. $GF_k(FO) = k$-guarded $NRS\text{-}DATALOG = NRS\text{-}DATALOG[k]$.

Proof. In [12] (proof of Theorem 8.5) a translation from $GF(FO)$ to 1-guarded NRS-DATALOG is presented. By taking k-guards instead of single-atom guards, this translation is easily generalized to a translation from $GF_k(FO)$ to k-guarded NRS-DATALOG. Vice versa, the well-known standard translation from NRS-DATALOG to FO preserves k-guardedness. Thus, $GF_k(FO) = k$-guarded NRS-DATALOG. It remains to show that k-guarded NRS-DATALOG is equal to $NRS\text{-}DATALOG[k]$.

To see that k-guarded $NRS\text{-}DATALOG \subseteq NRS\text{-}DATALOG[k]$, let R be a rule of such a program and consider the following rule decomposition $\langle T,\chi,\lambda\rangle$ of R. Let the root v_0 be such that $\lambda(v_0)$ is precisely the set of edges corresponding to the guard atoms of R, and let $\chi(v_0) = var(\lambda(v_0))$. Each other atom A of the rule gives rise to a child v_A of v_0 where $\lambda(v_A) = \{var(A)\}$ and $\chi(v_A) = var(A)$. This decomposition clearly fulfill all requirements of Definition 10, and is a strict rule decomposition. Thus, R has strict hypertree-width bounded by k. Hence the program is in $NRS\text{-}DATALOG[k]$.

For the other direction, proceed as follows. Let P be a $NRS\text{-}DATALOG[k]$ program and $R: H \leftarrow B$ a rule in P. Then, according to Definition 10, R has strict hypertree width at most k. Let $D = \langle T,\chi,\lambda\rangle$ be a strict width k rule decomposition of R.

We assume, w.l.o.g., that D is a complete hypertree decomposition of the hypergraph $\mathscr{H}(B)$ and, by a similar argument as in the proof of Lemma 10, that D is a strong hypertree decomposition. We furthermore assume, w.l.o.g., that the variables of the head of R are all contained in $\chi(v_0) = var(\lambda(v_0))$, where v_0 is the root of T (otherwise we may re-root T in the appropriate vertex). Finally, we can assume that non-EDB vertices occur in leaves of the decomposition only. In fact, if some non-EDB vertex v is a non-leaf vertex, then, by Definition 10, there must be an EDB vertex w such that $\chi(v) = var(\lambda(v)) \subseteq \chi(w)$, and hence we can detach all children of v and attach them as children of w. By repeating this operation as long as possible, we end up with an equivalent tree decomposition $\langle T',\chi,\lambda\rangle$ of width k, where all non-EDB vertices, i.e., negative and IDB literals, appear in leaves only. Denote the literals corresponding to $\lambda(u)$ for all leaf-children u of v by $leaflits(v)$, and the nonleaf children of v by $nl(v)$. For each vertex v of T, denote by $atoms(\lambda(v))$ the set of all atoms A such that $var(A) \in \lambda(v)$.

We now transform each non-leaf vertex v of T into a k-guarded rule $rule(v)$ of the form

$$rule(v)\colon\ H_v \leftarrow \bigwedge_{A \in atoms(\lambda(v))} A \wedge \bigwedge_{B \in leaflits(v)} B \wedge \bigwedge_{u \in nl(v)} H_u,$$

where for each node v except the root v_0, H_v denotes an atom $h_v(interface(v))$ with a new predicate symbol h_v, having as arguments the variables $interface(v) = \chi(v) \cap \chi(\uparrow v)$, where $\uparrow v$ denotes the father of v in T. For the root v_0, H_{v_0} simply denotes the head atom of R.

For instance, rules r_1, r_2, r_3, and r_4 of program P_2 in Example 8 can be obtained applying the above transformation to rule r_1 of program P_1 in Example 7 with the rule decomposition shown in Fig. 3.

After having transformed each rule this way into a set of equivalent rules, the resulting Datalog program is equivalent to the original one and is a k-guarded NRS-DATALOG program. □

We conclude this section with a complexity result for $GF_k(FO)$. Under the proviso that one uses the RAM model and a data representation such that a join between k relations can be done in time n^k (cf. the appendix of [10]), the following theorem holds.

Theorem 4. *The combined complexity of* $GF_k(FO)$ (*on a RAM*) *is* $O(d^k \times f^2)$, *where d is the size of the largest relation in the database and f is the formula size.*

Proof. We evaluate a $GF(FO)$ formula by first transforming it into an equivalent k-guarded NRS-DATALOG program and then evaluating that program.

As explained in [18] (see also the related discussion in [10]), the translation from $GF(FO)$ to Datalog LITE is quadratic. In the proof of Theorem 3, essentially the same method for translating $GF_k(FO)$ to k-guarded NRS-DATALOG is used. Therefore, the translation from GF_k to k-guarded NRS-DATALOG is quadratic, too. In turn, evaluating a k-guarded NRS-DATALOG program is similar to evaluating a Datalog LITE program of size ℓ, which is feasible in time $O(d \times \ell)$ by translating it in time $O(d \times \ell)$ into a propositional logic program of size $O(d \times \ell)$ and evaluating the latter in linear time [12]. The only difference is that now each k-guarded rule gives rise to $O(d^k)$ propositional rules. The resulting propositional program has thus size $O(d^k \times \ell) = O(d^k \times f^2)$, whence our upper bound. □

6. Conclusion and further research

In this paper, we established two main results that make a strong case for the naturalness of the concept of bounded hypertree width. The first main result is a characterization of bounded hypertree width in terms of a combinatorial game. Since bounded treewidth can be characterized in terms of a similar, but simpler, game, this result makes it possible to compare the two notions in a game-theoretic framework. The second main result characterizes bounded hypertree width in terms of definability in the loosely guarded fragment of first-order logic with a bounded number of guards. This contribution reveals a connection between bounded hypertree width and definability in a fragment of first-order logic that recently has received considerable attention because of its applications to modal logics.

We believe that our results answer important questions on the nature and the expressive power of bounded hypertree-width queries. They show that the concept of hypertree decomposition, while more general, is similarly natural as the one of tree decomposition. This concept was introduced in the context of conjunctive query processing. It was so far used in this area and in the area of constraint satisfaction, only. Given that the method of hypertree decompositions is a general method of problem simplification, we express the hope that it may be fruitfully used in other areas of computer science, too.

Several interesting questions are left for future research. Let us conclude this paper by stating three mutually related problems that we think are of particular relevance.

Problem 1. The method of hypertree decompositions can be further generalized: Let us define the concept of *generalized hypertree decomposition* by just dropping condition 4 from the definition of hypertree decomposition (Definition 1). Correspondingly, we can introduce the concept of *generalized hypertree width* $ghw(\mathscr{H})$ of a hypergraph $\mathscr{H}$. This notion is *strictly* more general than the notion of hypertree width (this was independently observed by Adler [3]). We can prove that all classes of Boolean queries having bounded *ghw* can be answered in polynomial time. But we currently do not know whether these classes of queries are polynomially recognizable. This recognition problem is related to the mysterious *hypergraph sandwich problem* [24], which has remained unsolved for a long time. If the latter is polynomially solvable, then also queries of bounded *ghw* are polynomially recognizable.

Problem 2. The characterization of treewidth by Seymour and Thomas [28] by the robber and cops game does not assume the monotonicity of the game. Their deep result is actually that k cops can capture a robber if and only if they can do so monotonically. As recently shown by Adler [3], this is not the case for robbers and marshals. In fact, Adler constructed a hypergraph on which 3 marshals have a nonmonotonic winning strategy but at least 4 marshals are needed for capturing the robber monotonically. She also provided more complex examples that prove that for each constant k, a gap of k is achievable between the numbers of marshals needed in the two versions of the robber and marshals game. A tight bound of this gap in terms of the size of the input hypergraph is currently missing.

Acknowledgments

Research supported by *FWF* (*Austrian Science Funds*) under the project Z29-INF (Wittgenstein Award), and by *MURST* under Project COFIN-2000 "From Data to Information (D2I)" Part of the work of Francesco Scarcello and Nicola Leone has been carried out while they were visiting the Technische Universität Wien. The authors thank the anonymous referees and Joerg Flum for their useful comments.

Appendix. Compact vs general strategies

In this appendix, we formally prove that general strategies, where marshals can choose different moves for different positions of the robber staying in a given escape space, are in fact equivalent to the strategies, here called compact strategies, defined in Section 3. Thus, this additional degree of freedom gives in fact no additional power to marshals.

Note that general strategies requires a different formal representation. We define a *general strategy* for k marshals as a function σ that, given a k-configuration (M, r), returns a k-position M' for the marshals. Thus, the function σ depends now on the variable r the robber stands on, rather than just on her escape space.

Given a general strategy σ for k marshals, the *general game tree* for σ is a rooted tree T whose vertices are k-configurations of the game, defined as follows. The root of T is the configuration $(\emptyset, r_0)$, where r_0 is the variable representing the first position of the robber. Without loss of generality, we may assume r_0 is the lexicographically first variable of $\mathscr{H}$. In fact, this first choice is inessential, since there are no marshals on the hypergraph and the robber can freely run everywhere. The leaves of T are either capture configurations or configurations resulting after escape steps. A nonleaf vertex $v = (M, r)$ of T, has c children labelled by $(M', r_1), \ldots, (M', r_c)$, where $M' = \sigma(v)$ is the new position of the marshals provided by the σ strategy, and $r_1, \ldots, r_c$ are all possible choices for the robber given her previous standing variable r, and the past and the current locations M and M' of the marshals (i.e., $r_1, \ldots, r_c$ are the variables $[var(M) \cap var(M')]$-*connected to* r). A *winning general strategy* for the marshals is a strategy σ such that all the leaves of the game tree for σ are capture configurations.

Recall from Section 3 that we considered only the family of strategies where the marshals choose their next move just on the basis of the escape space of the robber, i.e., those strategies where marshals make a unique move for all the locations of the robber belonging to the same escape space. We next call the strategies with this restriction compact strategies. Formally, we say that a strategy σ is *compact* if, for every pair of configurations $c_1, c_2, Escape(c_1) = Escape(c_2)$ entails $\sigma(c_1) = \sigma(c_2)$.[5]

Theorem A.1. *There is a winning general strategy for k marshals if and only if there is a winning compact strategy for k marshals.*

Proof. The *if* part trivially holds, because any winning compact strategy is a winning general strategy, as well.

To prove the *only if* part, consider a winning general strategy σ for k marshals, and let T the general game tree for σ. We transform T into the (general) game tree for a winning compact strategy. The root remains unchanged. We visit T breadth-first and proceed as follows. Let (M, r_i) be a vertex of T. Each sibling (M, r_j) of (M, r_i) such that $Escape(M, r_i) = Escape(M, r_j)$ is modified in the following way. We first build a copy S of the subtree rooted at (M, r_i) and we change its root (M, r_i) to (M, r_j). We then replace the subtree rooted at (M, r_j) by S. Note that, at each step of the transformation, we get a game tree. Indeed, consider the configuration (M, r_j), and observe that the possible moves of the robber depend solely on her escape space and on the current and next positions of the marshals. Thus, by construction, the children of (M, r_j) represent all and only the legal steps from this configuration, i.e., if the marshals move from M to M', for each variable r'_j, (M, r_j) has a child (M', r'_j) if and only if r'_j is $[var(M) \cap var(M')]$-*connected* to

[5] With a small abuse of notation, for any configuration $c = (M, r)$, we will write also $\sigma(M, r)$ instead of $\sigma(c)$, as well as $Escape(c)$ instead of $Escape(M, r)$.

r_j. Moreover, observe that all the leaves of this tree are capture configurations. Indeed, since σ is a winning strategy, all the leaves of the subtree rooted at (M, r_i), which replaced the subtree rooted at (M, r_j), are capture configurations.

Let T' be the tree obtained at the end of this transformation. From T', we define a strategy σ' as follows. If (M, r) is any vertex of T' different from the root, and v is its parent, then $\sigma'(v) = M$. By construction, the strategy σ' is compact. Furthermore, it is a winning strategy. Indeed, as observed above, all the leaves of T' are capture configurations. □

References

[1] S. Abiteboul, O.M. Duschka, Complexity of answering queries using materialized views, in: Proceedings of the 17th ACM Symposium on Principles of Database Systems (PODS'98), Seattle, Washington, 1998, pp. 254–263.

[2] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley Publishing Company, Reading, MA, 1995.

[3] I. Adler, Spiele als Hilfsmittel zu Strukturuntersuchungen bei Graphen und Hypergraphen (German), Diploma Thesis, Institut für mathematische Logik und Grundlagen der Informatik, Mathemartischa Fakultät, University of Freiburg, Freiburg im Breisgau, Germany, 2002. Available at http://www.math.uni-freiburg.de/archiv/diplom/isolde_adler.html

[4] J. Van Benthem, Dynamic bits and pieces, ILLC Research Report, University of Amsterdam, 1997.

[5] A.K. Chandra, P.M. Merlin, Optimal implementation of conjunctive queries in relational databases, in: Proceedings of the ACM Symposium on Theory of Computing (STOC'77), 1977, pp. 77–90.

[6] Ch. Chekuri, A. Rajaraman, Conjunctive query containment revisited, Theoret. Comput. Sci. 239(2) (2000) 211–229. A preliminary version appeared in: Proceedings of ICDT'97, Lecture Notes in Computer Science, Vol. 1186, Springer, Berlin, 1997, pp. 56–70.

[7] B. Courcelle, J. Engelfried, G. Rozenberg, Handle-rewriting hypergraph grammars, J. Comput. System Sci. 46 (1993) 218–270.

[8] R. Dechter, Constraint networks, in: Encyclopedia of Artificial Intelligence, 2nd Edition, Wiley, New York, 1992, pp. 276–285.

[9] R. Dechter, J. Pearl, Tree clustering for constraint networks, Artif. Intell. 38 (1989) 353–366.

[10] J. Flum, M. Frick, M. Grohe, Query evaluation via tree-decomposition, in: Proceedings of ICDT'01, Lecture Notes in Computer Science, Vol. 1973, Springer, Berlin, 2001, pp. 22–38.

[11] E.C. Freuder, A sufficient condition for backtrack-bounded search, J. ACM 32 (4) (1985) 755–761.

[12] G. Gottlob, E. Grädel, H. Veith, Datalog LITE: a deductive query language with linear time model checking, ACM Trans. Comput. Logic 3 (1) (2002) 42–79.

[13] G. Gottlob, N. Leone, F. Scarcello, On tractable queries and constraints, in: Proceedings of Database and Expert Systems Applications (DEXA'99), Lecture Notes in Computer Science, Vol. 1677, Springer, Florence, August 1999, pp. 1–15.

[14] G. Gottlob, N. Leone, F. Scarcello, A comparison of structural CSP decomposition methods, Artif. Intell. 124(2) (2000) 243–282. A preliminary version appeared, in: Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99), Vol. 1, 1999, pp. 394–399.

[15] G. Gottlob, N. Leone, F. Scarcello, The complexity of acyclic conjunctive queries, J. ACM 48(3) (2000) 431–498. An extended abstract concerning part of this work has been published, in: Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS'98), Palo Alto, CA, 1998, pp. 706–715.

[16] G. Gottlob, N. Leone, F. Scarcello, Hypertree decompositions and tractable queries, J. Comput. System Sci. 64(3) (2002) 579–627. An extended abstract concerning part of this work has been published, in: Proceedings of the Eighteenth ACM Symposium on Principles of Database Systems (PODS'99), Philadelphia, Pennsylvania, May 1999, pp. 21–32.

[17] G. Gottlob, N. Leone, F. Scarcello, Computing LOGCFL certificates, Theoret. Comput. Sci. 270(1–2) (2002) 761–777. A preliminary version appeared, in: Proceedings of the 26th International Colloquium on Automata,

Languages and Programming (ICALP'99), Lecture Notes in Computer Science, Vol. 1644, Springer, Prague, July, 1999, pp. 361–371.
[18] G. Gottlob, R. Pichler, Hypergraphs in model checking: acyclicity hypertree-width versus clique-width, in: Proceedings of ICALP 2001, 28th International Colloquium on Automata, Languages and Programming, Crete, Greece, July 8–12, 2001, pp. 708–719.
[19] E. Grädel, On the restraining power of guards, J. Symbolic Logic 64 (1999) 1719–1742.
[20] M. Grohe, T. Schwentick, L. Segoufin, When is the evaluation of conjunctive queries tractable? in: Proceedings of the ACM Symposium on Theory of Computing (STOC'01), 2001.
[21] M. Gyssens, P.G. Jeavons, D.A. Cohen, Decomposing constraint satisfaction problems using database techniques, Artif. Intell. 66 (1994) 57–89.
[22] M. Gyssens, J. Paredaens, A Decomposition Methodology for Cyclic Databases, in: Advances in Database Theory, Vol. 2, Plenum Press, New York, NY, 1984, pp. 85–122.
[23] Ph.G. Kolaitis, M.Y. Vardi, Conjunctive-query containment and constraint satisfaction, J. Comput. System Sci. 61 (2000) 302–332.
[24] A. Lustig, O. Shmueli, Acyclic hypergraph projections, J. Algorithms 30 (1999) 400–422.
[25] N. Robertson, P.D. Seymour, Graph minors II, Algorithmic aspects of tree-width, J. Algorithms 7 (1986) 309–322.
[26] F. Rossi, C. Petrie, V. Dhar, On the equivalence of constraint satisfaction problems, in: Proceedings of the 9th European Conference on Artificial Intelligence (ECAI'90), Stockholm, Sweden, 1990, pp. 550–556.
[27] F. Scarcello, A. Mazzitelli, The hypertree decomposition homepage, http://wwwinfo.deis.unical.it/~frank/Hypertrees/
[28] P.D. Seymour, R. Thomas, Graph searching and a min–max theorem for tree-width, J. Combin. Theory Ser. B 58 (1993) 22–33.
[29] J.D. Ullman, Principles of Database and Knowledge Base Systems, Vol. II, Computer Science Press, Rockville, MD, 1989.
[30] J.D. Ullman, Information integration using logical views, Theoret. Comput. Sci. 239 (2) (2000) 189–210.
[31] M. Vardi, Complexity of relational query languages, in: Proceedings of the 14th ACM Symposium on Theory of Computing (STOC'82), 1982, pp. 137–146.
[32] M. Vardi, Constraint satisfaction and database theory, Tutorial at the 19th ACM Symposium on Principles of Database Systems (PODS'00). Currently available at: http://www.cs.rice.edu/~vardi/papers/pods00t.ps.gz.
[33] M. Yannakakis, Algorithms for acyclic database schemes, in: C. Zaniolo, C. Delobel (Eds.), Proceedings of the International Conference on Very Large Data Bases (VLDB'81), Cannes, France, 1981, pp. 82–94.

Further reading

B. Courcelle, Monadic second-order logic of graphs VII: graphs as relational structures, Theoret. Comput. Sci. 101 (1992) 3–33.
E. Wanke, Bounded tree-width and LOGCFL, J. Algorithms 16 (1994) 470–491.