Noname manuscript No.
(will be inserted by the editor)

A Method for Reduction of Examples in Relational Learning

Ondřej Kuželka · Andrea Szabóová · Filip Železný

Received: date / Accepted: date

Abstract Feature selection methods often improve the performance of attribute-value learning. We explore whether also in relational learning, examples in the form of clauses can be reduced in size to speed up learning *without affecting the learned hypothesis*. To this end, we introduce the notion of safe reduction: a safely reduced example cannot be distinguished from the original example *under the given hypothesis language bias*. Next, we consider the particular, rather permissive bias of bounded treewidth clauses. We show that under this hypothesis bias, examples of arbitrary treewidth can be reduced efficiently. We evaluate our approach on four data sets with the popular system Aleph and the state-of-the-art relational learner nFOIL. On all four data sets we make learning faster in the case of nFOIL, achieving an order-of-magnitude speed up on one of the data sets, and more accurate in the case of Aleph.

Keywords Relational learning · Feature selection · Bounded Treewidth

1 Introduction

Reducing the complexity of input data is often beneficial for learning. In attribute-value learning, a wide range of *feature selection* methods is available [23]. These methods try to select a strict subset of the original example features (attributes) while maintaining or even improving the performance of the model learned from it with respect to that learned from the original feature set. For binary classification tasks with Boolean features, the REDUCE [22] algorithm has been proposed that removes so called *irrelevant* features. For any model learned with the original feature set, a model with same or better fit on the learning examples may be expressed without the irrelevant

O. Kuželka · A. Szabóová · F. Železný
Czech Technical University in Prague
Faculty of Electrical Engineering
E-mail: {kuzelon2,szaboand,zelezny}@fel.cvut.cz

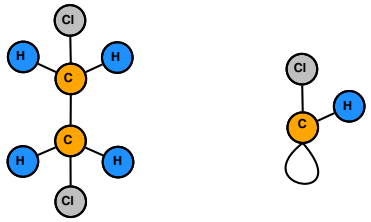


Fig. 1 A learning example and its reduction.

features. In the later work [1], the REFER algorithm extended REDUCE to the multiple-class learning setting.

In inductive logic programming—an important framework for relational learning [7]—examples are not expressed as tuples of feature values but rather take a structured form. For instance in the classical setting of *learning from entailment* [6], examples are represented as first-order clauses. Feature-selection methods are thus not applicable to simplify such learning examples. Here we are interested to see whether also first-order clausal examples can somehow be reduced while guaranteeing that the set of logical formulas which can be induced from such reductions would not be affected.

An obvious approach would be to look for θ-reductions [29] of the input clauses. A θ-reduction of a clause is a smaller, but subsumption-equivalent (and thus also logically equivalent) clause. We have explored an approach based on θ-subsumption before in [20], achieving learning speed-up factors up to 2.63. However, the main problem of θ-reduction is that finding it is an NP-hard problem, rendering the approach practically unfeasible in domains with large examples such as those describing protein structures [27].

Here we follow the key idea that the complexity curse can be avoided by sacrificing part of the generality of θ-reduction. In particular, we will look for reductions which may not be equivalent to the original example in the logical sense, but which are equivalent *given the language bias* of the learning algorithm. In other words, if the learning algorithm is not able to produce a hypothesis covering the original example but not covering its reduction (or vice versa), the latter two may be deemed equivalent. For instance, consider the clausal example $\leftarrow \mathsf{atom}(\mathsf{a1}) \wedge \mathsf{carbon}(\mathsf{a1}) \wedge \mathsf{bond}(\mathsf{a1}, \mathsf{a2}) \wedge \ldots$, whose entire structure is shown in the left of Figure 1. Assume that all terms in hypotheses are variables and hypotheses must correspond to trees when viewed graphically. Then the learning example is equivalent to the simpler one shown in the right of the figure.

Our first main contribution is a formal framework for example reduction based on the given language bias for hypotheses. Our second main contribution is the application of the above framework to the specific bias of *bounded treewidth* clauses. We show that in this case, interestingly, learning examples can be reduced in polynomial time, and moreover, that, in some cases, they can be reduced even more than they would be using the NP-hard θ-reduction.

Intuitively, a clause viewed as a graph (not necessarily a tree) has a small treewidth if it can be recursively decomposed into small subgraphs that have small overlap [9]. As the previous paragraph indicates, the benefits gained from the bounded treewidth assumption are significant. Notably though, the price we pay for them is not too high in that the bias would be over-restrictive. First remind that, as a hypothesis bias, it only constrains the learned clauses, and not the learning examples. Second, low treewidth is in fact characteristic of clauses induced in typical ILP experiments. In [18] we observed that all clauses learned by the ILP system Progol in all the conducted experiments had treewidth 1 although this had not been stipulated by the language bias. Similarly, in experiments in another study [19], all clauses learned by the systems nFOIL and kFOIL were of treewidth 1 after the removal of the variable formally identifying the learning example. These observations become plausible when viewing the exemplary chemical-domain clauses shown in the leftmost column of Table 1, which have the respective treewidths 1 and 2.

A salient feature of our approach is its general application scope. Indeed, example reduction can take place independently of the type of ILP learner employed subsequently. While we evaluate the approach in the standard ILP setting of learning from entailment, it is also relevant to propositionalization [15], which aims at the construction of feature-based descriptions of relational examples. Interestingly, propositionalization can simultaneously benefit from both the relational example reduction step employed before the construction of features, and any feature selection algorithm applied subsequently on the constructed feature set. In this sense, our approach can be combined with standard feature selection methods.

A shorter version of this paper already appeared in [17]. The work presented here significantly extends the scope of the already presented method. While in [17] only mutually non-resolving clausal theories composed of bounded-treewidth clauses were considered, here we extend applicability of the method also to the case of possibly mutually resolving Horn clausal theories composed of bounded-treewidth clauses. To establish this result we need to employ the subsumption theorem [28] and show that the class of clauses with treewidth bounded by some k is closed under formation of binary resolvents. Besides these new theoretical results, we also introduce *generalizing* and *specializing* safe reduction of learning examples, which can be applied on learning examples even if the hypothesis language is not fixed. However, in this case, the theoretical guarantees are weaker than in the case when a hypothesis language bias is fixed.

The paper is organized as follows. In the next section we review the preliminaries for the study, namely θ-subsumption and reduction, their correspondence to the constraint satisfaction problem model, and the concepts of tree decomposition and treewidth. Sections 3 and 4 present the framework of bounded subsumption and bounded reduction. Section 5 introduces the framework for safe reduction of relational examples under a given hypothesis language bias. Section 6 instantiates the framework to the langauge bias of bounded-treewidth clauses and shows that reduction under this bias can be

conducted effectively. We experimentally evaluate our method in Section 7 and conclude in Section 8.

2 Preliminaries: Logic, Constraint Satisfaction, Treewidth

In this section, we briefly describe basic concepts of logic-based relational machine learning. We start by defining the *learning from entailment* setting [6]. Then we describe θ-subsumption and the so-called *subsumption theorem*, which provides a link between θ-subsumption, resolution and entailment. After that we describe θ-subsumption as a *constraint satisfaction problem*. Finally, we briefly introduce tree decompositions and treewidth of clauses and their relationship to tractability of θ-subsumption.

First, we state some notational conventions used in this paper. A first-order-logic clause is a universally quantified disjunction of first-order-logic literals. For convenience, we do not write the universal quantifiers explicitly. We treat clauses as disjunctions of literals and as sets of literals interchangeably. We will sometimes use a slightly abused notation $a(x,y) \subseteq a(w,x) \vee a(x,y)$ to denote that a set of literals of one clause is a subset of literals of another clause. To denote the number of literals in a clause A, we use the set notation $|A|$. The set of variables in a clause A is written as $vars(A)$ and the set of all terms as $terms(A)$. Terms can be variables or constants. A set of clauses is said to be *standardized-apart* if no two clauses in the set share a variable.

A substitution θ is a mapping from variables to terms. Notation-wise, $\theta = \{V_1/t_1, V_2/t_2, V_k/t_k\}$ represents a substitution which maps the variable V_1 to the term t_1, the variable V_2 to the term t_2 etc. The variables which are not explicitly listed in a substitution are assumed to be mapped to themselves. Two substitutions are considered equivalent if and only if they map any variable to the same value. Thus, for example, $\theta_1 = \{V_1/t\}$ is considered equivalent to $\theta_2 = \{V_1/t, V_2/V_2\}$.

2.1 Learning from Entailment

One of the basic learning settings in logic-based relational learning is *learning from entailment*, where learning examples are first-order logic clauses and hypotheses are first-order logic clausal theories.

Definition 1 (Covering under Learning from Entailment) Let H be a clausal theory and e be a clause. Then we say that H covers e under entailment if and only if $H \models e$.

The basic learning task is to find a clausal theory H which covers all positive examples and no negative examples.

Example 1 (Learning from Entailment) In the learning from entailment setting, there is no background knowledge. Let us assume that the sets of positive

and negative examples, which are clauses, are

$$E^+ = \{bird(parrot) \leftarrow flies(parrot)\}$$

and

$$E^- = \{bird(hippo) \leftarrow \neg flies(hippo)\}.$$

The formula $H = flies(X) \rightarrow bird(X)$ is a valid solution since

$$(flies(X) \rightarrow bird(X)) \models (bird(parrot) \leftarrow flies(parrot))$$

and

$$(flies(X) \rightarrow bird(X)) \not\models (bird(hippo) \leftarrow \neg flies(hippo))$$

(there are models of H which are not models of $bird(hippo) \leftarrow \neg flies(hippo)$).

Of course, the ILP problem as described above is only an idealized problem which focuses only on the *search* aspect of learning. It does not take into account generalization performance of the learned theories. One approach to control generalization performance of learned theories which has been applied in practical implementations of logic-based relational machine learning [26] is to limit the set of allowed hypotheses either by imposing limits on maximum length of learned theories [26] or by considering only theories from a syntactically limited class - determinate theories, theories consisting of non-recursive definite clauses etc.

2.2 θ-subsumption and the Subsumption Theorem

In order to approximate the undecidable entailment relation Plotkin introduced the so-called θ-subsumption [29]. If A and B are clauses, then we say that the clause A θ-subsumes the clause B, if and only if there is a substitution θ such that $A\theta \subseteq B$. If $A \preceq_\theta B$ and $B \preceq_\theta A$, we call A and B θ-equivalent (written $A \approx_\theta B$). If $A \preceq_\theta B$ then $A \models B$ but the other direction of the implication does not hold in general. However, it does hold for non-self-resolving function-free clauses. Another way to look at $A \preceq_\theta B$ is to view it as homomorphism between relational structures. Informally, $A \preceq_\theta B$ holds if and only if there is a homomorphism $A \rightarrow B$ where A and B are treated as relational structures.

If A is a clause and there is another clause $\widehat{A}$ such that $A \approx_\theta \widehat{A}$ and $|\widehat{A}| < |A|$ then A is said to be θ-reducible. A minimal such $\widehat{A}$ is called θ-reduction of A. For example, the clause

$$A = east(T) \leftarrow hasCar(T, C) \wedge hasLoad(C, L1) \wedge hasLoad(C, L2) \wedge box(L2)$$

is θ-reducible because $A \approx_\theta \widehat{A}$ where

$$\widehat{A} = east(T) \leftarrow hasCar(T, C) \wedge hasLoad(C, L) \wedge box(L).$$

In this case, $\widehat{A}$ is also a θ-reduction of A. The concept of θ-reduction of clauses is equivalent to the concept of *cores* [2] of relational structures. *Core* of a relational structure A is a minimal relational structure homomorphically equivalent to A.

A precise relationship between logical entailment, θ-subsumption and first-order resolution is given by the so-called *subsumption theorem* [28]. Before stating the subsumption theorem for Horn clauses, we need to briefly describe SLD resolution. We start the description by introducing some auxiliary concepts which are used in SLD resolution.

Definition 2 (Most general unifier) Let $L = \{l_1, l_2, \ldots, l_k\}$ be a set of literals. A substitution ϑ is called a unifier of L if and only if $l_1\vartheta = l_2\vartheta = \cdots = l_k\vartheta$. If θ is a unifier of L and if for any unifier θ' of L there is a substitution θ'' such that $\theta' = \theta\theta''$ then θ is called a most general unifier of L.

Example 2 Let us have a set of literals $L = \{lit(A, B, c), lit(X, X, Y)\}$. A most general unifier of L is $\theta_1 = \{A/X, B/X, Y/c\}$. Another most general unifier of L is $\theta_2 = \{A/W, B/W, X/W, Y/c, W/X\}$. Clearly, θ_1 can be obtained from θ_2 as $\theta_1 = \theta_2\{X/W, W/X\}$ and, similarly, θ_2 can be obtained from θ_1 as $\theta_2 = \theta_1\{X/W, W/X\}$. Notice that θ_2 maps all variables occurring in L to 'fresh' variables which are not contained in L or to constants. In general, one can always find such most general unifiers.

The next definition introduces the notion of a *binary resolvent* of clauses.

Definition 3 (Binary resolvent) Let $A = l_1 \vee l_2 \vee \cdots \vee l_m$ and $B = m_1 \vee m_2 \vee \cdots \vee m_n$ be two standardized-apart clauses. Let θ be a most general unifier of the literals l_i, $\neg m_j$ not affecting any variable in the set $(vars(A) \cup vars(B)) \setminus (vars(l_i) \cup vars(m_j))$ such that $vars(l_i\theta) \cap vars(A) = vars(l_i\theta) \cap vars(B) = \emptyset$. Next, let

$$C = (l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_m \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta$$

Then C is called a binary resolvent of A and B.

Now, we can define what is meant by deriving a clause by SLD-resolution.

Definition 4 (Derivation by SLD-resolution) Let $\mathcal{S}$ be a set of Horn clauses and A be a Horn clause. A derivation of the clause A from $\mathcal{S}$ by SLD-resolution is a finite sequence of Horn clauses $R_1, R_2, \ldots, R_k$ where $R_k = A$ such that $R_0 \in \mathcal{S}$ and each R_i is a binary resolvent of R_{i-1} and a definite clause (i.e. a Horn clause having exactly one positive literal) $C_i \in \mathcal{S}$ where the literals resolved upon are the literal in the head of C_i and a selected negative literal from R_{i-1}.

Finally, we can state the subsumption theorem for SLD-resolution. Notice that this theorem 'works' only for Horn clauses.

Proposition 1 (Subsumption theorem for SLD-resolution [28]) *Let $\mathcal{S}$ be a set of Horn clauses and A be a Horn clause which is not a tautology. Then $\mathcal{S} \models A$ if and only if there is a clause B derivable by SLD resolution from $\mathcal{S}$ such that $B \preceq_\theta A$.*

The description of the subsumption theorem for SLD-resolution which was presented in this section follows the description from the work of Nienhuys-Cheng and de Wolf [28]. There is a small difference regarding the definition of *binary resolvent* which is that we require the unifier θ to never map any variable to a variable already contained in the resolved clauses. However, the definitions are equivalent for our purposes because for any derivation $R_1, R_2, \ldots, R_k$ by SLD-resolution according to the definition used here there is a derivation $R'_1, R'_2, \ldots, R'_k$ by SLD-resolution according to the definition used in their work such that all R_i and R'_i are logically equivalent (and vice versa).

2.3 θ-subsumption as a Constraint Satisfaction Problem

Both θ-subsumption and θ-reduction are *NP-hard problems*. *Constraint satisfaction* [8] with finite domains represents a class of problems closely related to the θ-subsumption problem. Constraint satisfaction algorithms can be used to compute θ-subsumption and θ-reduction relatively efficiently (though, still with *exponential runtime* in the worst case). In fact, as shown by Feder and Vardi [11], constraint satisfaction problems with finite domains and θ-subsumption are two almost identical problems although the terminology for stating them differs. This equivalence of CSP and θ-subsumption has been exploited by Maloberti and Sebag [25] who used off-the-shelf CSP algorithms to develop a fast θ-subsumption algorithm. In this paper, we will use a CSP local-consistency algorithm known as *k-consistency* for reduction of learning examples. For this we will need a CSP representation of θ-subsumption problems.

Definition 5 (Constraint Satisfaction Problem) A constraint satisfaction problem is a triple $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, where $\mathcal{V}$ is a set of variables, $\mathcal{D} = \{D_1, \ldots, D_{|\mathcal{V}|}\}$ is a set of domains of values (for each variable $v \in \mathcal{V}$), and $\mathcal{C} = \{C_1, \ldots, C_{|\mathcal{C}|}\}$ is a set of constraints. Every constraint is a pair (s, R), where s (*scope*) is an n-tuple of variables and R is an n-ary relation. An evaluation of variables θ satisfies a constraint $C_i = (s_i, R_i)$ if $s_i\theta \in R_i$. A solution is an evaluation that maps all variables to elements in their domains and satisfies all constraints.

Now, we explain how θ-subsumption problems can be represented as constraint satisfaction problems. The CSP representation of the problem of deciding $A \preceq_\theta B$ has the following form. There is one CSP variable X_v for every variable $v \in vars(A)$. The domain of each of these CSP variables contains all terms from $terms(B)$. The set of constraints contains one constraint $C_l = (s_l, R_l)$ for each literal $l = pred_l(t_1, \ldots, t_k) \in A$. We denote by $I_{var} = (i_1, \ldots, i_m) \subseteq (1, \ldots, k)$ the indexes of variables in arguments of l (the

other arguments might contain constants). The scope s_l of the constraint C_l is $(X_{t_{i_1}}, \ldots, X_{t_{i_m}})$ (i.e. the scope contains all CSP variables corresponding to variables in the arguments of literal l). The relation R_l of the constraint C_l is then constructed in three steps.

1. A set L_l is created which contains all literals $l' \in B$ such that $l \preceq_\theta l'$ (note that checking θ-subsumption of two literals is a trivial linear-time operation).
2. Then a relation R_l^* is constructed for every literal $l \in A$ from the arguments of literals in the respective set L_l. The relation R_l^* contains a tuple of terms $(t'_1, \ldots, t'_k)$ if and only if there is a literal $l' \in L_l$ with arguments $(t'_1, \ldots, t'_k)$.
3. Finally, the relation R_l of the constraint C_l is the projection of R_l^* on indexes I_{var} (only the elements of tuples of terms which correspond to variables in l are retained).

When we refer to *CSP encoding* of θ-subsumption problems in the remainder of this paper, we implicitly mean this encoding. Next, we exemplify the transformation process.

Example 3 (Converting θ-subsumption to CSP) Let us have clauses A and B as follows

$$A = hasCar(C) \vee hasLoad(C, L) \vee shape(L, box)$$

$$B = hasCar(c) \vee hasLoad(c, l_1) \vee hasLoad(c, l_2) \vee shape(l_2, box).$$

We now show how we can convert the problem of deciding $A \preceq_\theta B$ to a CSP problem. Let $\mathcal{V} = \{C, L\}$ be a set of CSP-variables and let $\mathcal{D} = \{D_C, D_L\}$ be a set of domains of variables from $\mathcal{V}$ such that $D_C = D_L = \{c, l_1, l_2\}$. Further, let $\mathcal{C} = \{C_{hasCar(C)}, C_{hasLoad(C,L)}, C_{shape(L,box)}\}$ be a set of constraints with scopes (C), (C, L) and (L) and with relations $\{(c)\}$, $\{(c, l_1), (c, l_2)\}$ and $\{(l_2)\}$, respectively. Then the constraint satisfaction problem given by $\mathcal{V}$, $\mathcal{D}$ and $\mathcal{C}$ represents the problem of deciding $A \preceq_\theta B$ as it admits a solution if and only if $A \preceq_\theta B$ holds.

2.4 Tree Decompositions and Treewidth

Tractability of constraint satisfaction problems can be 'measured' by the *treewidth of their Gaifman graphs*. The Gaifman (or primal) graph of a clause A is the graph with one vertex for each variable $v \in vars(A)$ and an edge for every pair of variables $u, v \in vars(A)$, $u \neq v$ such that u and v appear in a literal $l \in A$. Similarly, we define Gaifman graphs for CSPs. The Gaifman graph of a CSP problem $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ is the graph with one vertex for each variable $v \in \mathcal{V}$ and an edge for every pair of variables which appear in a scope of some constraint $c \in \mathcal{C}$. Gaifman graphs can be used to define treewidth of clauses or CSPs.

Definition 6 (Tree decomposition of a Graph, Treewidth) A tree decomposition of a graph $G = (V, E)$ is a tree T with nodes labelled by sets of vertices such that

- For every vertex $v \in V$, there is a node of T with a label containing v.
- For every edge $(v, w) \in E$, there is a node of T with a label containing v, w.
- For every $v \in V$, the subgraph of T obtained by removing all its nodes not containing v (along with the associated edges) remains connected.

The width of a tree decomposition T is the maximum cardinality of a label in T minus 1. The treewidth of a graph G is the smallest number k such that G has a tree decomposition of width k. The treewidth of a clause is equal to the treewidth of its Gaifman graph. Analogically, the treewidth of a CSP is equal to the treewidth of its Gaifman graph.

Note that tree decompositions are not unique in general. That is why treewidth is defined as the *smallest* number k such that G has a tree decomposition of width k. Finding a tree decomposition of minimum width is an NP-hard problem. For the methods presented in this paper, we will not need to construct tree decompositions explicitly.

Example 4 To provide intuition, an illustration of Gaifman graphs of two exemplar clauses

$$C_1 = atm(A, h) \vee bond(A, B, 1) \vee atm(B, c) \vee bond(B, C, 2) \vee atm(C, o)$$
$$C_2 = bond(A, B, 1) \vee bond(B, C, 1) \vee bond(C, D, 1) \vee \ldots$$
$$\ldots bond(D, E, 1) \vee bond(E, A, 1)$$

and their tree decompositions is shown in Table 1. The tree decomposition of the Gaifman graph of the clause C_1 of minimum width can be found easily because the Gaifman graph of C_1 is a tree. The tree decomposition of the Gaifman graph of the clause C_2 with minimum width 2, shown in Table 1, looks more complicated. For instance, notice that A, C and E are contained together in one label of a node of the tree decomposition despite the fact that neither A nor E are connected with C in the Gaifman graph. However, if C was not contained in this particular label, the subgraph of the tree decomposition obtained by removing all its nodes not containing C would be disconnected which would violate the requirements from Definition 6. In Table 1, we also show an alternative tree decomposition of the Gaifman graph of the clause C_2 with non-minimal width.

For instance, all trees have treewidth 1, cycles have treewidth 2, rectangular $n \times n$ grid-graphs have treewidth n. Treewidth is usually used to isolate tractable sub-classes of NP-hard problems. A general result about polynomial-time solubility of some problems on graphs of bounded treewidth is Courcelle's theorem [5] which essentially states that every problem definable in Monadic Second-Order logic can be solved in linear time on graphs of bounded treewidth (though, the run-time is exponential in the treewidth parameter of the graphs).

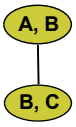
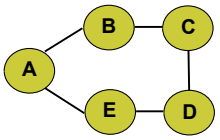
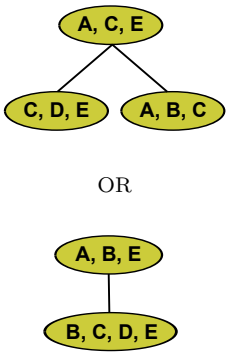
Clause	Gaifman graph	Tree decomposition
$\mathsf{atm}(A,\mathsf{h}) \vee \mathsf{bond}(A,B,1) \vee \mathsf{atm}(B,\mathsf{c}) \vee \mathsf{bond}(B,C,2) \vee \mathsf{atm}(C,\mathsf{o})$		
$\mathsf{bond}(A,B,1) \vee \mathsf{bond}(B,C,1) \vee \mathsf{bond}(C,D,1) \vee \mathsf{bond}(D,E,1) \vee \mathsf{bond}(E,A,1)$		

Table 1 An illustration of Gaifman graphs and tree decompositions of clauses.

Constraint satisfaction problems with treewidth bounded by k can be solved in polynomial time by the k-consistency algorithm[1].

Sometimes, it will be convenient to use the following definition of *tree decomposition of a clause* which does not directly refer to the concept of a Gaifman graph. The *treewidth* of a clause given by this definition and the *treewidth* given by Definition 6 are equivalent.

Definition 7 (Tree decomposition of a Clause) A tree decomposition of a clause A is a tree T with nodes labelled by sets of first-order-logic variables such that

- For every variable $V \in vars(A)$, there is a node of T with a label containing V.
- For every pair of variables $U, V \in vars(A)$ such that $U \neq V$ and U, V are both contained in a literal $l \in A$, there is a node of T with label containing U, V.
- For every $V \in vars(A)$, the subgraph of T obtained by removing all its nodes with labels not containing V (along with the associated edges) remains connected.

[1] In this paper we follow the conventions of [2]. In other works, what we call k-*consistency* is known as *strong* $k+1$-*consistency* [30].

The width of a tree decomposition T is the maximum cardinality of a label in T minus 1. The treewidth of a clause A is the smallest number k such that A has a tree decomposition of width k.

We have already noted that constraint satisfaction problems with treewidth bounded by k can be solved in polynomial time by the k-consistency algorithm. We now briefly describe the k-consistency algorithm and review some of its properties which are important for the presentation in this paper. The description is based on the presentation by Atserias et al. [2]. Let us have a CSP $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ where $\mathcal{V}$ is the set of variables, $\mathcal{D}$ is the set of domains of the variables and $\mathcal{C}$ is the set of constraints. A partial solution ϑ is an evaluation of variables from $\mathcal{V}' \subseteq \mathcal{V}$ which is a solution of the sub-problem $\mathcal{P}' = (\mathcal{V}', \mathcal{D}, \mathcal{C})$. If ϑ and φ are partial solutions, we say that φ extends ϑ (denoted by $\vartheta \subseteq \varphi$) if $\mathsf{Supp}(\vartheta) \subseteq \mathsf{Supp}(\varphi)$ and $\vartheta(V) = \varphi(V)$ for all $V \in \mathsf{Supp}(\vartheta)$, where $\mathsf{Supp}(\vartheta)$ and $\mathsf{Supp}(\varphi)$ denote the sets of variables which are affected by the respective evaluations ϑ and φ. The k-consistency algorithm then works as follows:

k-consistency algorithm:

1. Given a constraint satisfaction problem $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ and a positive integer k.
2. Let H be the collection of all partial solutions ϑ with $|\mathsf{Supp}(\vartheta)| \leq k + 1$.
3. For every $\vartheta \in H$ with $|\mathsf{Supp}(\vartheta)| \leq k$ and every $V \in \mathcal{V}$, if there is no $\varphi \in H$ such that $\vartheta \subseteq \varphi$ and $V \in \mathsf{Supp}(\varphi)$, remove ϑ and all its extensions from H.
4. Repeat step 3 until H is unchanged.
5. If H is empty return *false*, else return *true*.

If the k-consistency algorithm returns *true* and $\mathcal{P}$ has treewidth bounded by k then $\mathcal{P}$ is guaranteed to have a solution [12]. For constraint satisfaction problems with generally unbounded treewidth, k-consistency is only a necessary but not a sufficient condition to have a solution. If the k-consistency algorithm returns *false* for a CSP problem $\mathcal{P}$ then $\mathcal{P}$ is guaranteed to have no solutions. If it returns *true* then the problem may or may not have some solutions. So, equivalently, if the problem is soluble then k-consistency always returns *true*. This can be seen as follows. If the problem has a solution then there must be a collection of partial solutions which cannot be removed by the k-consistency algorithm. This is because, for every variable V and every partial solution which is a subset of a complete solution of the problem, we can find another partial solution which is a superset of the partial solution and contains the variable V - simply by selecting an appropriate subset of the complete solution. Therefore the set of partial solutions reduced by the k-consistency algorithm will not become empty and the k-consistency algorithm will return true.

A basic property of k-consistency that we will also need is the following. If the k-consistency algorithm returns *true* for a CSP problem then it will also return *true* for any problem created from the original problem by removing some variables and some constraints, i.e. with a *subproblem*. This can be seen

by noticing that if the k-consistency algorithm starts with a set H of all partial solutions and returns *true* then it must also return *true* if it starts with a set of all partial solutions of some problem which is a superset of this set. The set of partial solutions of the subproblem must necessarily be a superset of the set of partial solutions of the original problem projected on the variables of the subproblem (from monotonicity of constraints).

It is easy to check that if a clause A has treewidth bounded by k then also the CSP representation of the problem of deciding $A \preceq_\theta B$ has treewidth bounded by k for any clause B. It is known that due to this and due to the equivalence of CSPs and θ-subsumption, the problem of deciding θ-subsumption $A \preceq_\theta B$ can be solved in polynomial time when clause A has bounded treewidth (which has been known for long time by the above mentioned widely known correspondence between θ-subsumption and CSP problems).

Proposition 2 *We say that a clause A is k-consistent w.r.t. a clause B (denoted by $A \lhd_k B$) if and only if the k-consistency algorithm executed on the CSP representation of the problem of deciding $A \preceq_\theta B$ returns* true. *If A has treewidth at most k and $A \lhd_k B$ then $A \preceq_\theta B$.*

Proof Follows directly from the solubility of CSPs with bounded treewidth by the k-consistency algorithm [2] and from the equivalence of CSPs and θ-subsumption shown earlier in this section.

3 Bounded Subsumption

In this section, we describe x-subsumption and x-equivalence which were introduced in our work on learning using so-called *bounded least general generalization* [16]. The x-subsumption and x-equivalence are weaker versions of θ-subsumption and θ-equivalence.

Definition 8 (x-subsumption, x-equivalence) Let X be a possibly infinite set of clauses. Let A, B be clauses not necessarily from X. We say that A x-subsumes B w.r.t. X (denoted by $A \preceq_X B$) if and only if $(C \preceq_\theta A) \Rightarrow (C \preceq_\theta B)$ for every clause $C \in X$. If $A \preceq_X B$ and $B \preceq_X A$ then A and B are called x-equivalent w.r.t. X (denoted by $A \approx_X B$). For a given set X, the relation $\preceq_X$ is called x-subsumption w.r.t. X and $\approx_X$ is called x-equivalence w.r.t. X.

Conventional θ-subsumption is a special case of x-subsumption. It is an x-subsumption w.r.t. the set of all clauses. It is not hard to check that x-subsumption is a transitive and reflexive relation on clauses and that x-equivalence is an equivalence-relation on clauses. If two clauses are x-equivalent w.r.t. a set X, they are indistinguishable by single-clause non-self-resolving hypotheses from X under learning from entailment which follows from the fact that entailment coincides with θ-subsumption for single-clause non-self-resolving hypotheses. However, a problem is that we seldom search for single-clause hypotheses. More often, we search for hypotheses in the form of clausal theories

for which θ-subsumption is not strong enough to decide entailment (which is undecidable anyway). We return to this problem in Section 6 where we show that a similar 'indistinguishability' property also holds for hypotheses in the form of clausal theories consisting of bounded-treewidth Horn clauses. This will allow us to develop a method for reduction of learning examples for the cases when we search for Horn clausal theories.

Definition 8 also provides no efficient way to decide x-subsumption between two clauses as it demands θ-subsumption of an infinite number of clauses to be tested in some cases. However, for many practically relevant sets of clauses X, there is a relation called *x-presubsumption* which implies x-subsumption and has other useful properties as we shall see later (for example, it allows quick finding of reduced versions of clauses etc.).

Definition 9 (x-presubsumption) Let X be a set of clauses. If $\lhd_X$ is a relation such that: (i) if $A \lhd_X B$ and $C \subseteq A$ then $C \lhd_X B$, (ii) if $A \in X$, ϑ is a substitution and $A\vartheta \lhd_x B$ then $A \preceq_\theta B$, (iii) if $A \preceq_\theta B$ then $A \lhd_X B$. Then we say that $\lhd_X$ is an x-presubsumption w.r.t. the set X.

The next proposition shows that if X is a set of clauses and $\lhd_X$ is an x-presubsumption w.r.t. X then $\lhd_X$ provides a sufficient condition for x-subsumption w.r.t. X.

Proposition 3 *Let X be a set of clauses. If $\preceq_X$ is x-subsumption w.r.t. X and $\lhd_X$ is an x-presubsumption w.r.t. X then $(A \lhd_X B) \Rightarrow (A \preceq_X B)$ for any two clauses A, B (not necessarily from X).*

We will use this proposition in the next section where we deal with bounded reduction of clauses. We will use it for showing that certain procedures which transform clauses always produce clauses which are x-equivalent w.r.t. a given set X.

In this paper, we use x-presubsumption w.r.t. bounded-treewidth clauses based on the polynomial-time k-consistency algorithm.

Proposition 4 *Let $k \in N$ and let $\lhd_k$ be a relation on clauses defined as follows: $A \lhd_k B$ if and only if the k-consistency algorithm run on the CSP-encoding of the θ-subsumption problem $A \preceq_\theta B$ returns true. The relation $\lhd_k$ is an x-presubsumption w.r.t. the set X_k of all clauses with treewidth at most k.*

In practice, when we have two clauses A and B and want to check the x-presubsumption based on k-consistency for some fixed k, we first convert the θ-subsumption problem $A \preceq_\theta B$ to the CSP representation described in Section 2.3 but we do not attempt to solve the resulting CSP problem. We just run the k-consistency algorithm and if the k-consistency algorithm returns *true* then the x-presubsumption $A \lhd_X B$ holds and if not then the x-presubsumption does not hold.

The main reason why we focus on the x-presubsumption based on the k-consistency algorithm is that, as we show in Section 6, bounded-treewidth

clauses are closed under formation of binary resolvents. This will allow us to find efficient methods for reduction of learning examples and prove their correctness. There are x-presubsumptions for other interesting sets, such as the set of all *acyclic clauses*, however, some of these sets are unfortunately not closed under formation of binary resolvents. Therefore, unlike the x-presubsumption based on k-consistency, these x-presubsumptions cannot be used for reduction of learning examples.

4 Reduction of Clauses

If there is an efficiently decidable x-presubsumption relation then it can be used to search for clauses which are smaller than the original clause but are still x-equivalent to it (due to Proposition 3). This process will be used for reduction of learning examples in Section 6. Ideally, we would like the process to result in the smallest clause x-equivalent to the original clause, which we term *x-equivalent reduction*. This turns out not to be possible in many cases in practice because we usually do not have a decision procedure for x-subsumption but only for some respective x-presubsumption[2]. The next definition introduces the notion of *x-equivalent reduction* formally.

Definition 10 (x-equivalent reduction) Let X be a set of clauses. We say that a clause $\widehat{A}$ is an x-equivalent reduction of a clause A if and only if $A \approx_X \widehat{A}$ (where $\approx_X$ denotes x-equivalence w.r.t. X) and $|\widehat{A}|$ is minimal.

Note that the notion of x-equivalent clause is different from the notion of *x-reduction* introduced in our work on learning using bounded least general generalization [16]. Whereas an *x-reduction* of a clause must always be more or equally general as the reduced clause, the same does not hold for the x-equivalent reduction, which can be equally or more general, more specific than the original clause or none of these (i.e. neither more general nor more specific).

Example 5 Let X_1 be the set of all clauses with treewidth at most 1 and let us have the following clause

$$A_1 = e(A, B) \vee e(B, C) \vee e(C, A).$$

Then $\widehat{A}_1 = e(A, B) \vee e(B, A)$ is the x-equivalent reduction of A_1. We can see that $A_1 \preceq_\theta \widehat{A}_1$ but $\widehat{A}_1 \not\preceq_\theta A_1$, i.e. the x-equivalent reduction of A_1 is more specific than A_1. Let us now consider another set X_2. It will consist of all clauses with treewidth at most 1 composed only of literals based on predicate $f/2$. Let us have the following two clauses A_2 and A_3 on which we will demonstrate the remaining two possibilities (i.e. an x-equivalent reduction

[2] This is not always the case. For example, we have a decision procedure for x-subsumption w.r.t. the set of all clauses – the ordinary θ-subsumption which is decidable but NP-hard.

is more general or neither more general nor more specific than the original clause).

$$A_2 = e(A, A) \vee f(A, B) \vee f(B, A)$$
$$A_3 = e(A, B) \vee f(A, B) \vee f(B, C) \vee f(C, A)$$

Now, the x-equivalent reductions of A_2 and A_3 w.r.t. the set X_2 are the same $\widehat{A}_2 = \widehat{A}_3 = f(A, B) \vee f(B, A)$. In the case of A_2, its x-equivalent reduction is more general than it and in the case of A_3 it is neither more general nor more specific than the original clause.

In order to be able to compute x-equivalent reductions, we need to be able to decide x-subsumption. However, as already stated, we very often have efficient decision procedures for x-presubsumption, but not for x-subsumption. Importantly, if there is an x-presubsumption $\lhd_X$ w.r.t. a set X decidable in polynomial time then there are polynomial-time algorithms for computing good approximations of x-equivalent reductions. We present here two such algorithms. The first one is called *literal elimination algorithm* and the second one *literal substitution algorithm.*

Literal elimination algorithm (LEA):

1. Given a clause A which should be reduced.
2. Set $A' := A$.
3. Select a literal L from A' such that $A' \lhd_X A' \setminus \{L\}$. If there is no such literal, return A' and finish.
4. Set $A' := A' \setminus \{L\}$
5. Go to step 3.

The next proposition states formally the properties of the literal-elimination algorithm. It also gives a bound on the size of the reduced clause which is output of the literal elimination algorithm.

Proposition 5 *Let us have a set X and a polynomial-time decision procedure for checking $\lhd_X$ which is an x-presubsumption w.r.t. the set X. Then, given a clause A on input, the literal-elimination algorithm finishes in polynomial time and outputs a clause $\widehat{A}$ satisfying the following conditions:*

1. *$\widehat{A} \preceq_\theta A$ and $A \preceq_X \widehat{A}$ where $\preceq_X$ is the x-subsumption w.r.t. the set X.*
2. *$|\widehat{A}| \leq |\widehat{A}_\theta|$ where $\widehat{A}_\theta$ is a θ-reduction of a subset of A's literals with maximum length.*

So, the output $\widehat{A}$ of the literal elimination algorithm has the same properties as x-equivalent reduction ($\widehat{A} \preceq_\theta A$ and $A \preceq_X \widehat{A}$ implies $\widehat{A} \approx_X A$) with one difference and that is that it may not be minimal in some cases, i.e. that there may be some other smaller clause having these properties. The output of the literal elimination algorithm is also never more specific than the original clause.

The second reduction algorithm, the *literal substitution algorithm* differs from the literal elimination algorithm mainly in that it never returns a clause which would be more general than the original clause.

Literal substitution algorithm (LSA):

1. Given a clause A which should be reduced.
2. Set $A' := A$
3. Select a substitution $\vartheta = \{V/t\}$ where $V \in vars(A')$, $t \in terms(A')$ and $V \neq t$ such that $A'\vartheta \lhd_X A'$. If there is no such substitution, return A' and finish.
4. $A' := A'\vartheta$
5. Go to step 3.

The properties of the literal substitution algorithm are stated formally in the next proposition.

Proposition 6 *Let us have a set X and a polynomial-time decision procedure for checking $\lhd_X$ which is an x-presubsumption w.r.t. the set X. Then, given a clause A on input, the literal substitution algorithm finishes in polynomial time and outputs a clause $\widehat{A}$ satisfying the following conditions:*

1. *$\widehat{A} \preceq_X A$ and $A \preceq_\theta \widehat{A}$ where $\preceq_X$ is the x-subsumption w.r.t. the set X.*
2. *$|terms(\widehat{A})| \leq |terms(\widehat{A}_\theta)|$ where $\widehat{A}_\theta$ is a θ-reduction of a clause $A\vartheta$ with maximum cardinality of the set $terms(\widehat{A}_\theta)$ where ϑ is some substitution mapping variables to elements of the set $terms(A)$.*

5 Safe Reduction of Learning Examples

The learning task that we consider in this paper is fairly standard. We are given labelled learning examples encoded as first-order-logic clauses and we would like to find a classifier predicting the class labels of examples as precisely as possible. We work within the *learning from entailment setting* [6] which was discussed in Section 2.1. We aim at finding a reduction procedure that would allow us to reduce the number of literals in the examples while guaranteeing that the coverage of any hypothesis from a pre-fixed hypothesis language $\mathcal{L}$ would not be changed. Let us present the necessary definitions.

Definition 11 (Safe Equivalence and Safe Reduction under Entailment) Let e and $\widehat{e}$ be two clauses and let $\mathcal{L}$ be a language specifying all possible hypotheses. Then $\widehat{e}$ is said to be safely equivalent to e if and only if $\forall \mathcal{H} \in \mathcal{L} : (\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \widehat{e})$. If e and $\widehat{e}$ are safely equivalent and $|\widehat{e}| < |e|$ then $\widehat{e}$ is called safe reduction of e.

Clearly, if we have a hypothesis $\mathcal{H} \in \mathcal{L}$ which splits the examples to two sets X and Y then this hypothesis $\mathcal{H}$ will also split the respective set of safely reduced examples to the sets $\widehat{X}$, $\widehat{Y}$ containing the safely reduced examples from the sets X and Y, respectively. Also, when predicting classes of test-set examples, any deterministic classifier that bases its decisions on the queries using the covering relation $\models$ will return the same classification even if we replace some of the examples by their safe reductions. The same is also true for

propositionalization approaches that use the $\models$ relation to construct boolean vectors which are then processed by attribute-value-learners.

When we search for classifiers in the form of clausal theories, it is possible to reduce the examples without limiting the hypothesis language. Nevertheless, the price paid for this is that some good clausal theories may be missed. However, we can guarantee that the learnt theories will not be worse than any clausal theory complying with the given hypothesis language. In fact, they can be often much better. What we need for this, are two new notions: *generalizing* and *specializing safe reduction*.

Definition 12 (Generalizing and Specializing Safe Reduction under Entailment) Let e and $\widehat{e}$ be two clauses and let $\mathcal{L}$ be a language specifying all possible hypotheses. If e and $\widehat{e}$ are safely equivalent, $|\widehat{e}| < |e|$ and $\widehat{e} \preceq_\theta e$ ($e \preceq_\theta \widehat{e}$, respectively) then $\widehat{e}$ is called generalizing (specializing, respectively) safe reduction of e.

If we have a set of positive examples E^+ and a set of negative examples E^-, then we can reduce the positive examples using generalizing safe reduction and the negative examples using specializing safe reduction, resulting in a new set of reduced positive examples $\widehat{E}^+$ and a new set of reduced negative examples $\widehat{E}^-$. Now, any hypothesis $\mathcal{H}$ which covers n^+ examples from the set $\widehat{E}^+$ must cover $m^+ \geq n^+$ positive examples from E^+. Similarly, any hypothesis $\mathcal{H}$ which covers n^- examples from the set $\widehat{E}^-$ must cover $m^- \leq n^-$ negative examples from E^-. Therefore, most of the scores used typically in logic-based relational learning (such as *coverage* and *compression*) computed on the sets of reduced examples must be lower-bounds of the respective scores computed on the original sets of non-reduced examples – and it does not matter if $\mathcal{H} \notin \mathcal{L}$. If $\mathcal{H} \in \mathcal{L}$, then the scores computed on the sets of reduced and non-reduced examples are equal. As a consequence, any relational learning algorithm which guarantees to find a clause maximizing a given scoring function must return a theory, which is at least as good as any theory from $\mathcal{L}$, but often better. The reasoning is as follows. Let $\mathcal{H}^*$ be the clausal theory with the maximum score among the clausal theories from $\mathcal{L}$. This maximum score is the same when computed from the sets of non-reduced examples and from the sets of reduced examples. Now, let $\mathcal{H}^{**}$ be the clausal theory with the maximum score among all clauses (not just those from $\mathcal{L}$). The score of $\mathcal{H}^{**}$ on the sets of reduced examples must be greater or equal than the score of $\mathcal{H}^*$. Since the scores computed on the sets of reduced examples are lower bounds of the scores computed on the sets of non-reduced examples, the score of $\mathcal{H}^{**}$ on the sets of non-reduced examples must be also greater or equal than the score of $\mathcal{H}^*$.

6 Safe Reduction by Literal Elimination and Substitution Algorithms

In this section, we describe how learning examples can be reduced using the literal elimination and substitution algorithms. We do this in two steps. First,

we show that if we have a hypothesis language determining a set of allowed constants, then we can variabilize all constants which are not allowed by the hypothesis language and the resulting examples will remain safely equivalent to the original learning examples. Second, we show that if the hypothesis language is known to permit only hypotheses consisting of bounded-treewidth Horn clauses, then it is possible to use the literal elimination and literal substitution algorithms using k-consistency as x-presubsumption for safe reduction of learning examples.

We start by showing formally that by replacing constants which do not appear in the given hypothesis language $\mathcal{L}$ and by θ-reducing the variabilized examples, we always obtain examples safely equivalent to the original examples w.r.t. $\mathcal{L}$.

Proposition 7 *Let $\mathcal{L}$ be a hypothesis language and let e be a clause. Let $\tilde{e}$ be a clause obtained from e by variabilizing the constants which are not contained in the hypothesis language. Then $(\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \tilde{e})$ for any $\mathcal{H} \in \mathcal{L}$ (i.e. e and $\tilde{e}$ are safely equivalent). If $\widehat{e}$ is a θ-reduction of $\tilde{e}$ and $|\widehat{e}| < |e|$ then $\widehat{e}$ is also a safe reduction of e.*

Note that the above proposition holds for conventional entailment relation $\models$. For instance, it would not hold if we used *negation as failure* (because then we could create a $nonequal(X, Y)$ predicate without using any constant).

Now, we turn our attention to showing how the literal elimination and the literal substitution algorithm can be used for safe reduction. The literal elimination and substitution algorithms are able to produce reduced clauses which are indistinguishable from the original non-reduced clauses by clauses from a given set X. This is very close to what we are looking for in a safe reduction procedure. The difference is that we would like the reduced clauses to be indistinguishable from the original non-reduced clauses not just by clauses from a given set X, but also by clausal theories $\mathcal{H} \in 2^X$. The next example shows that there are sets X for which the literal elimination and substitution algorithms can produce clauses which are distinguishable from the original non-reduced clauses by clausal theories $\mathcal{H} \in 2^X$.

Example 6 Let X_A be the set of all acyclic clauses [10]. Let us have an example

$$e = has4cycle \leftarrow e(a, b) \wedge e(b, c) \wedge e(c, d) \wedge e(d, a).$$

Then its x-equivalent reduction w.r.t. the set X_A, which can be computed by the literal elimination and substitution algorithms is

$$\widehat{e} = has4cycle \leftarrow e(A, B) \wedge e(B, C) \wedge e(C, A).$$

Now, let us consider a clausal theory $\mathcal{H}$

$$\begin{aligned}\mathcal{H} = \{\ & has4cycle \leftarrow cycle4(W, X, Y, Z),\\ & cycle4(W, X, Y, Z) \leftarrow e(W, X) \wedge e(X, Y) \wedge e(Y, Z) \wedge e(Z, W)\}.\end{aligned}$$

Despite the fact that $\mathcal{H}$ is composed only of acyclic clauses and that $e \approx \widehat{e}$ w.r.t. the set X_A, it holds $\mathcal{H} \not\models e$ and $\mathcal{H} \models \widehat{e}$.

So, there are sets of clauses X, such that if two examples are x-equivalent w.r.t. the set X, then they may still be distinguished by hypotheses composed of clauses from the set X, as the example above illustrates. The question is: are there any sets X w.r.t. which x-equivalence would imply safe equivalence w.r.t. 2^X? The next proposition gives a concrete answer to this question. It shows that if X is the set of bounded-treewidth function-free Horn clauses then x-equivalence w.r.t. the set X implies safe equivalence w.r.t. the set 2^X (which is the set of theories consisting of bounded-treewidth function-free Horn clauses).

Proposition 8 *Let X_k be the set of all function-free Horn clauses with treewidth at most k and let $\mathcal{L} = 2^{X_k}$ be the set of theories consisting of function-free Horn clauses with treewidth at most k. Then any two clauses which are x-equivalent w.r.t. X_k are also safely equivalent w.r.t. $\mathcal{L}$.*

Thus, the main result presented in this section can be phrased as follows: The safe reduction method based on the literal elimination and substitution algorithms can be used when considering theories composed of possibly mutually resolving bounded-treewidth Horn clauses. When learning clausal theories consisting of possibly mutually resolving bounded-treewidth Horn clauses, it is possible to start by reducing learning examples using the literal elimination and substitution algorithms with the polynomial-time x-presubsumption based on the k-consistency algorithm.

The literal elimination algorithm and the literal substitution algorithm can be used also for computing generalizing and specializing safe reductions of learning examples. Generalizing safe reductions can be computed by variabilization of constants not contained in the hypothesis language followed by application of the literal elimination algorithm. Similarly, specializing safe reductions can be computed using the literal substitution algorithms.

7 Experimental Evaluation of Safe Reduction

We experimentally evaluate usefulness of the *safe reduction of learning examples* with real-world datasets and two relational learning systems – the popular system Aleph and the state-of-the-art system nFOIL [21]. We implemented *literal-elimination* and *literal-substitution* algorithms for treewidth 1, i.e. for treelike clausal theories. We used the efficient algorithm AC-3 [24] for checking 1-consistency[3]. We forced nFOIL and Aleph to construct only clauses with treewidth 1 using their *mode declaration* mechanisms. We used three datasets in the experiments: *predictive toxicology challenge* [14], *CAD* [31] and *hexose-binding proteins* [27]. The PTC dataset contains descriptions of 344 molecules classified according to their toxicity for male rats. The molecules are described using only *atom* and *bond* information. The CAD dataset contains

[3] Note again the terminology used in this paper following [2]. In CSP-literature, it is often common to call 2-consistency what we call 1-consistency.

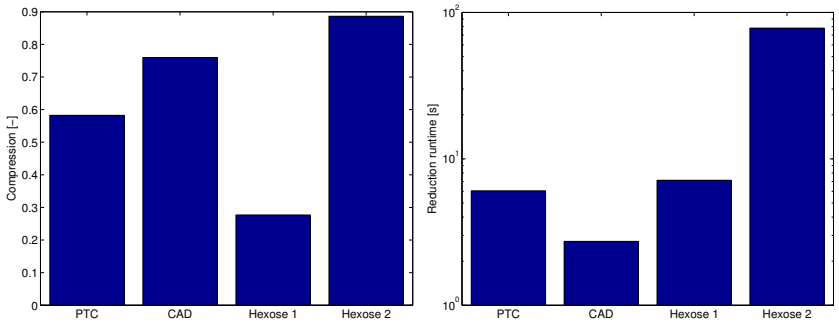


Fig. 2 Left: Compression rates achieved by *literal-substitution algorithm* on four datasets (for treewidth 1). **Right:** Time for computing reductions of learning examples on four datasets (for treewidth 1).

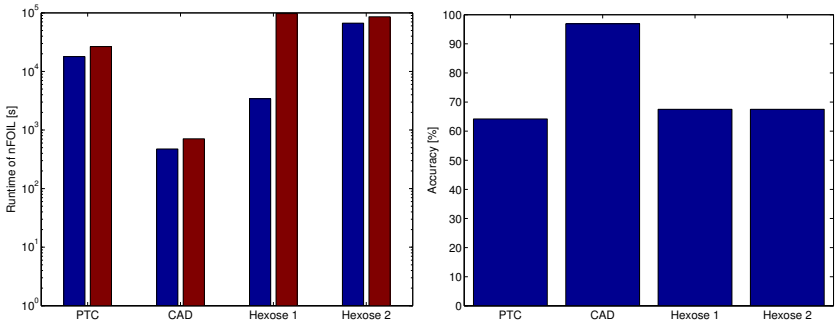


Fig. 3 Left: Runtime of nFOIL on reduced (blue) and non-reduced (red) datasets. **Right:** Predictive accuracies of nFOIL on four datasets estimated by 10-fold cross-validation.

descriptions of 96 class-labelled product-structure designs. Finally, the hexose-binding dataset contains 80 hexose-binding and 80 non-hexose-binding protein domains. Following [27] we represent the protein domains by atom-types and atom-names (each atom in an amino acid has a unique name) and pair-wise distances between the atoms which are closer to each other than some threshold value. We performed two experiments with the last mentioned dataset for cut-off set to 1 Angstrom (Hexose 1) and 2 Angstroms (Hexose 2).

We applied the *literal-elimination* algorithm followed by the *literal-substitution* algorithm on the three datasets. The compression rates (i.e. ratios of number of literals in the reduced learning examples divided by the number of literals in the original non-reduced examples) are shown in the left panel of Figure 2. The right panel of Figure 2 then shows the time needed to run the reduction algorithms on the respective datasets. We note that these times are generally negligible compared to runtimes of nFOIL and with the exception of Hexose 2 also to runtimes of Aleph.

7.1 Experiments with nFOIL

We used nFOIL to learn predictive models and evaluated them using 10-fold cross-validation. We used beam-size 100 for all experiments with the exception of the hexose-binding dataset with cut-off value 2 Angstroms, where we used beam-size 50. From one point of view, this is much higher than the beam-sizes used by [21], but on the other hand, we have the experience that this allows nFOIL to find theories which involve longer clauses and at the same time have higher predictive accuracies. The runtimes of nFOIL operating on reduced and non-reduced data are shown in the left panel of Figure 3. It can be seen that the reduction was beneficial in all cases but that the most significant speed-up of more than an order of magnitude was achieved on Hexose data. This could be attributed to the fact that nFOIL constructed long clauses on this dataset and the covering test used by it had not probably been optimized. So, in principle, nFOIL could be made faster by optimizing the efficiency of its covering test. The main point, however, is that we can speed-up the learning process for almost any relational learning algorithm merely by preprocessing its input. The right panel of Figure 3 shows nFOIL's predictive accuracies (estimated by 10-fold cross-validation). The accuracies were not affected by the reductions. The reason is that (unlike Aleph) nFOIL exploits learning examples only through the entailment queries.

7.2 Experiments with Aleph

We performed another set of experiments using the relational learning system Aleph. Aleph restricts its search space by bottom-clauses. After constructing a bottom-clause it searches for hypotheses by enumerating subsets of literals of the bottom-clause. When we reduce learning examples, which also means reduction of bottom-clauses, we are effectively reducing the size of Aleph's search space. This means that Aleph can construct longer clauses earlier than if it used non-reduced examples. On the other hand, this also implies that, with the same settings, Aleph may run longer on reduced data than on non-reduced data. That is because computing coverage of longer hypotheses is more time-consuming. Theories involving longer clauses may often lead to more accurate predictions than theories involving clauses which are too short to capture a desired concept. On the other hand, theories involving longer clauses may be prone to overfitting, which corresponds to the problem of so-called *bias-variance trade-off* [13]. For these reasons, we measured not only runtime and accuracy, but also the average number of learnt rules and the average number of literals in these rules on reduced and non-reduced data.

We ran Aleph on reduced and non-reduced versions of the datasets and evaluated it using 10-fold cross-validation. We used the literal elimination algorithm for reducing examples. In this case, we did not use the literal substitution algorithm because it could result in making Aleph's bottom clauses overly specific (note that, unlike nFOIL, Aleph does not access the learning examples

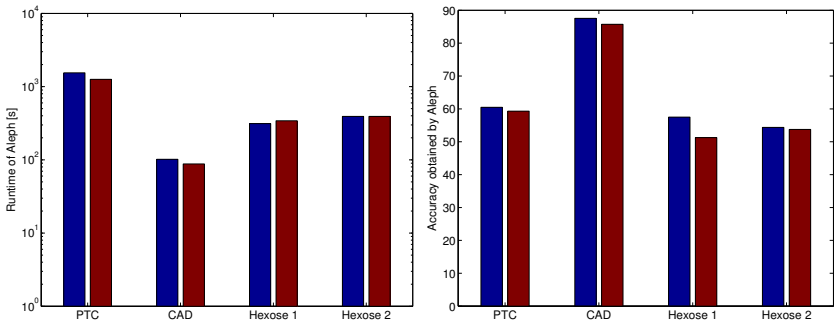


Fig. 4 Left: Runtime of Aleph on reduced (blue) and non-reduced (red) datasets. **Right:** Predictive accuracies of Aleph on reduced (blue) and non-reduced (red) datasets estimated by 10-fold cross-validation.

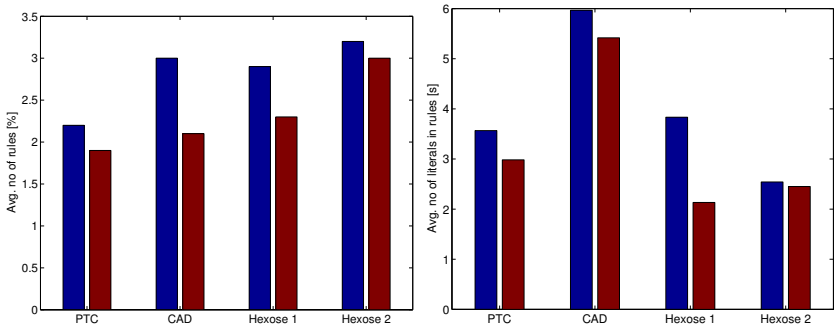


Fig. 5 Left: Average number of rules generated by Aleph on reduced (blue) and non-reduced (red) datasets. **Right:** Average number of literals in rules generated by Aleph on reduced (blue) and non-reduced (red) datasets.

only through entailment queries). We set the maximum number of explored *nodes* to 50000, the *noise* parameter to 1% of the number of examples in the respective datasets. The runtime in the performed experiments was higher for reduced versions of datasets PTC and CAD, the same for Hexose 2 and lower for Hexose 1 than for their non-reduced counterparts (see left panel of Figure 4). The accuracies were higher for reduced versions of all four datasets (see right panel of Figure 4). Similarly, the average number of rules, as well as the average number of literals in the rules, was higher for the reduced versions of all four datasets (see Figure 5). These results confirm the expectation that Aleph should be able to construct longer hypotheses on reduced datasets which, in turn, should result in higher predictive accuracies.

8 Conclusions

We have introduced a novel concept called *safe reduction*. We have shown how it can be used to safely reduce learning examples (without affecting learnability) which makes it possible to speed-up many relational learning systems

by merely preprocessing their input. In some cases, e.g. in the case of Aleph, where safe reduction of learning examples affects also size of the hypothesis search space, safe reduction may also lead to improvement of accuracy of learned models. The methods that we have introduced run in polynomial time for hypothesis languages composed of Horn clausal theories with treewidth bounded by a fixed constant (the time dependence on the fixed constant is exponential).

Acknowledgements This work was supported by the Czech Grant Agency through project 103/11/2170 *Transferring ILP techniques to SRL*. The authors would like to thank the anonymous reviewers of NFMCP'12 and of the JIIS special issue for helpful remarks.

A Propositions and Proofs

Proposition 3. *Let X be a set of clauses. If $\preceq_X$ is x-subsumption w.r.t. X and $\lhd_X$ is an x-presubsumption w.r.t. X then $(A \lhd_X B) \Rightarrow (A \preceq_X B)$ for any two clauses A, B (not necessarily from X).*

Proof We need to show that if $A \lhd_x B$ then $(C \preceq_\theta A) \Rightarrow (C \preceq_\theta B)$ for all clauses $C \in X$. First, if $A \lhd_x B$ and $C \not\preceq_\theta A$ then the proposition holds trivially. Second, $C \preceq_\theta A$ means that there is a substitution ϑ such that $C\vartheta \subseteq A$. This implies $C\vartheta \lhd_X B$ using the condition 1 from definition of x-presubsumption. Now, we can use the second condition which gives us $C \preceq_\theta B$ (note that $C \in X$ and $C\vartheta \lhd_X B$).

Proposition 4. *Let $k \in N$ and let $\lhd_k$ be a relation on clauses defined as follows: $A \lhd_k B$ if and only if the k-consistency algorithm run on the CSP-encoding of the θ-subsumption problem $A \preceq_\theta B$ returns true. The relation $\lhd_k$ is an x-presubsumption w.r.t. the set X_k of all clauses with treewidth at most k.*

Proof We need to verify that $\lhd_k$ satisfies the conditions stated in Definition 9.

1. *If $A \lhd_k B$ and $C \subseteq A$ then $C \lhd_k B$.* This holds because if the k-consistency algorithm returns *true* for a problem then it must also return *true* for any of its subproblems (recall the discussion in Section 2.4). It is easy to check that if $C \subseteq A$ are clauses then the CSP problem encoding the θ-subsumption problem $C \preceq_\theta B$ is a subproblem of the CSP encoding of the θ-subsumption problem $A \preceq_\theta B$. Therefore this condition holds.
2. *If $A \in X$, ϑ is a substitution and $A\vartheta \lhd_x B$ then $A \preceq_\theta B$.* If $A\vartheta \lhd_k B$ then the k-consistency algorithm applied on the CSP encoding $\mathcal{P}_\vartheta = (\mathcal{V}_\vartheta, \mathcal{D}_\vartheta, \mathcal{C}_\vartheta)$ of the problem $A\vartheta \preceq_\theta B$ finishes with a non-empty set of partial solutions H_ϑ. When applied on the CSP encoding $P = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ of the problem $A \preceq_\theta B$, the k-consistency algorithm finishes with a set of partial solutions H. What we need to show is that this set H is non-empty. We will do this by showing that H has a non-empty subset H^* which can be constructed as follows. For each partial solution $\varphi \in H_\vartheta$, we take all sets of variables $V \subseteq \mathcal{V}$ such that $|V| \leq k+1$ and $V\vartheta \subseteq \mathsf{Supp}(\varphi)$ and, for each such set V, we add to H^* a new partial solution φ^* such that $\mathsf{Supp}(\varphi^*) = V$ and $\varphi^*(v) = \varphi(v\vartheta)$ for all $v \in \mathsf{Supp}(\varphi^*)$. Clearly, any such φ^* is a valid partial solution of $\mathcal{P}$ because otherwise φ could not have been a valid partial solution of $\mathcal{P}_\vartheta$. Moreover, for every $\varphi^* \in H^*$ with $|\mathsf{Supp}(\varphi^*)| \leq k$ and every variable $v \in \mathcal{V}$, there exists a partial solution $\psi^* \in H^*$ such that $\varphi^* \subseteq \psi^*$ and $v \in \mathsf{Supp}(\psi^*)$. This can be shown by contradiction as follows. Let us suppose that there is a partial solution $\varphi^* \in H^*$ such that $|\mathsf{Supp}(\varphi^*)| \leq k$ and a variable $v \in \mathcal{V}$ such that there is no $\psi^* \in H^*$ which would satisfy $\varphi^* \subseteq \psi^*$ and $v \in \mathsf{Supp}(\psi^*)$. But then the respective solution $\varphi \in H_\vartheta$ from which φ^* was constructed should have been removed by the k-consistency algorithm because there could not have been any partial solution $\psi \in H_\vartheta$ such that $\varphi \subseteq \psi$ and $v\vartheta \in \mathsf{Supp}(\psi)$ which is a contradiction. For every partial solution $\varphi^* \in H^*$, the set H^* must also contain all partial 'sub-solutions' (i.e. solutions

φ'^* such that $\varphi'^* \subseteq \varphi^*$). As a consequence of the above, the k-consistency algorithm running on the CSP encoding of the problem $A \preceq_\theta B$ cannot remove from H any partial solution contained in the set H^* and, thus, $H^* \subseteq H$ and the k-consistency algorithm must return the value *true*. Since $A \in X_k$, it must also hold $A \preceq_\theta B$.
3. *If* $A \preceq_\theta B$ *then* $A \lhd_k B$. This is a property of k-consistency.

Proposition 5. *Let us have a set* X *and a polynomial-time decision procedure for checking* $\lhd_X$ *which is an* x*-presubsumption w.r.t. the set* X*. Then, given a clause* A *on input, the literal-elimination algorithm finishes in polynomial time and outputs a clause* $\widehat{A}$ *satisfying the following conditions:*

1. $\widehat{A} \preceq_\theta A$ *and* $A \preceq_X \widehat{A}$ *where* $\preceq_X$ *is an* x*-subsumption w.r.t. the set* X*.*
2. $|\widehat{A}| \leq |\widehat{A}_\theta|$ *where* $\widehat{A}_\theta$ *is a* θ*-reduction of a subset of* A*'s literals with maximum length.*

Proof We start by proving $\widehat{A} \preceq_\theta A$ and $A \preceq_X \widehat{A}$. This can be shown as follows. First, $A \preceq_X A'$ holds in any step of the algorithm which follows from $(A' \lhd_X A' \setminus \{L\}) \Rightarrow (A' \preceq_X A' \setminus \{L\})$ – recall that A' is replaced by $A' \setminus \{L\}$ in the literal elimination algorithm if and only if $A' \lhd_X A' \setminus \{L\}$ – and from transitivity of x-subsumption. Consequently we also have $A \preceq_X \widehat{A}$ because $\widehat{A} = A'$ in the last step of the algorithm. Second, $\widehat{A} \preceq_\theta A$ because $\widehat{A} \subseteq A$. Now, we prove the second part of the proposition. What remains to be shown is that the resulting clause $\widehat{A}$ will not be bigger than $\widehat{A}_\theta$. Since $\widehat{A} \subseteq A$, it suffices to show that $\widehat{A}$ cannot be θ-reducible. Let us assume, for contradiction, that it is θ-reducible. If $\widehat{A}$ was θ-reducible, there would have to be a literal $L \in \widehat{A}$ such that $\widehat{A} \preceq_\theta \widehat{A} \setminus \{L\}$. The relation $\lhd_X$ satisfies $(A \preceq_\theta B) \Rightarrow (A \lhd_X B)$ therefore it would also have to hold $A' \lhd_X A' \setminus \{L\}$. However, then L should have been removed by the literal-elimination algorithm which is a contradiction with $\widehat{A}$ being output of it. The fact that the literal-elimination algorithm finishes in polynomial time follows from the fact that, for a given clause A, it calls the polynomial-time procedure for checking the relation $\lhd_X$ at most $|A|^2$ times (the other operations of the literal-elimination algorithm can be performed in polynomial time as well).

We will need the following simple lemma in the proof of Proposition 6.

Lemma 1 *Let* A *be a clause. If* A *is* θ*-reducible then* $A\vartheta \approx_\theta A$ *where* $\vartheta = \{V/t\}$ *for some* $V \in vars(A)$ *and* $t \in terms(A)$ *,* $V \neq t$ *and* $|vars(A\vartheta)| < |vars(A)|$.

Proof Let A be θ-reducible and let $\widehat{A}_\theta \subseteq A$ be θ-reduction of A. There must be a substitution θ^* such that $A\theta^* = \widehat{A}_\theta$ and $|A\theta^*| < |A|$ and therefore also $|vars(A\theta^*)| < |vars(A)|$ (because if $|vars(A\theta^*)| = |vars(A)|$ then A and $A\theta^*$ would be isomorphic and they would have to contain the same number of literals). Let V^* be a variable contained in $vars(A)$ but not contained in $vars(A\theta^*)$ (there must be some such variable because $|vars(A\theta^*)| < |vars(A)|$) and let $\vartheta = \{V^*/t\} \subseteq \theta^*$. There are two cases depending on whether t is a variable or a constant. Let us start with the case when t is a variable which we denote as $W = t$ for clarity. Now, there must be a variable $S \neq V^*$ such that $\{S/W\} \subseteq \theta$ which can be shown as follows. If there is no such variable (i.e. if the only variable which is mapped on W is V^*) then we can construct a new substitution $\theta^{**} := \theta^*\theta^*$ which has the following two 'interesting' properties ('interesting' because they will allow us to arrive at a contradiction): (i) $A\theta^{**} \approx_\theta A$ and (ii) $|vars(A\theta^{**})| < |vars(A\theta^*)|$. The first property holds because $A \preceq_\theta A\theta^* \subseteq \widehat{A}_\theta \subseteq A$ implies $A \preceq_\theta A\theta^*\theta^* \subseteq \widehat{A}_\theta$. The second property can be shown as follows. Clearly, none of the variables not in $vars(A\theta^*)$ can reappear in $vars(A\theta^{**})$. Moreover, the variable W cannot be contained in $vars(A\theta^{**})$ because $V^*\theta^{**} \neq W$ and there was no other variable mapped to V^* or W by θ^*. Therefore $|vars(A\theta^{**})| < |vars(A\theta^*)|$, i.e. the second property must be true as well. However, then we have a contradiction because we assumed that $\widehat{A}_\theta$ was θ-reduction but $\widehat{A}_\theta$ cannot be a θ-reduction because $A \approx_\theta A\theta^{**}$ and $|A\theta^{**}| < |A\theta^*|$ (which follows from $|vars(A\theta^{**})| < |vars(A\theta^*)| = |vars(\widehat{A}_\theta)|$ and $A\theta^{**} \subseteq \widehat{A}_\theta$). Thus, there must be a variable $S \neq V^*$ such that $\{S/W\} \subseteq \theta$. It follows that if we set $\vartheta = \{V^*/S\}$ it must hold $A\theta^* = A\vartheta\theta^*$. Therefore $A\vartheta \preceq_\theta A\theta^* \approx_\theta A$. Now, if t is not a variable but a constant, we can simply set $\vartheta = \{V^*/t\}$ and it must hold $A\theta^* = A\vartheta\theta^*$ and therefore also $A\vartheta \preceq_\theta A\theta^* \approx_\theta A$. Since also trivially $A \preceq_\theta A\vartheta$, we have $A\vartheta \approx_\theta A$ and $|vars(A\vartheta)| < |vars(A)|$ which finishes the proof of this lemma.

Proposition 6. *Let us have a set X and a polynomial-time decision procedure for checking $\lhd_X$ which is an x-presubsumption w.r.t. the set X. Then, given a clause A on input, the literal substitution algorithm finishes in polynomial time and outputs a clause $\widehat{A}$ satisfying the following conditions:*

1. *$\widehat{A} \preceq_X A$ and $A \preceq_\theta \widehat{A}$ where $\preceq_X$ is the x-subsumption w.r.t. the set X.*
2. *$|terms(\widehat{A})| \leq |terms(\widehat{A}_\theta)|$ where $\widehat{A}_\theta$ is a θ-reduction of a clause $A\vartheta$ with maximum cardinality of the set $terms(\widehat{A}_\theta)$ where ϑ is some substitution mapping variables to elements of the set $terms(A)$.*

Proof We start by proving $\widehat{A} \preceq_X A$ and $A \preceq_\theta \widehat{A}$. This can be shown as follows. First, $A' \preceq_X A$ holds in any step of the algorithm which follows from $(A'\vartheta \lhd_X A') \Rightarrow (A'\vartheta \preceq_X A')$ – recall that A' is replaced by $A'\vartheta$ in the literal substitution algorithm if and only if $A'\vartheta \lhd_X A'$ – and from transitivity of x-subsumption. Consequently we also have $\widehat{A} \preceq_X A$ because $\widehat{A} = A'$ in the last step of the algorithm. Second, $A \preceq_\theta \widehat{A}$ because $\widehat{A} = A\theta'$ for some substitution θ' (θ' is the substitution composed of the substitutions ϑ applied on A' in the literal substitution algorithm). Now, we prove the second part of the proposition. What remains to be shown is that the resulting clause $\widehat{A}$ will not have more terms than $\widehat{A}_\theta$. Since $\widehat{A} = A\theta'$, it suffices to show that $\widehat{A}$ cannot be θ-reducible. Let us assume, for contradiction, that it is θ-reducible. If $\widehat{A}$ was θ-reducible, then there would have to be a substitution $\vartheta = \{V/t\}$ such that $A\vartheta \approx_\theta A$ where $\vartheta = \{V/t\}$ for some $V \in vars(A)$, $t \in terms(A)$, $V \neq t$ (this follows from Lemma 1). The relation $\lhd_X$ satisfies $(A \preceq_\theta B) \Rightarrow (A \lhd_X B)$ therefore it would also have to hold $A'\vartheta \lhd_X A'$. However, then ϑ should have been applied on A' by the literal substitution algorithm which is a contradiction with $\widehat{A}$ being output of it. The fact that the literal-substitution algorithm finishes in polynomial time follows from the fact that, for a given clause A, it calls the polynomial-time procedure for checking the relation $\lhd_X$ at most $|A|^3$ times (the other operations of the literal-substitution algorithm can be performed in polynomial time as well).

Lemma 2 (Plotkin [29]) *Let A and B be clauses. If $A \preceq_\theta B$ then $A \models B$.*

Proposition 7. *Let $\mathcal{L}$ be a hypothesis language and let e be a clause. Let $\tilde{e}$ be a clause obtained from e by variabilizing the constants which are not contained in the hypothesis language. Then $(\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \tilde{e})$ for any $\mathcal{H} \in \mathcal{L}$. If $\widehat{e}$ is a θ-reduction of $\tilde{e}$ and $|\widehat{e}| < |e|$ then $\widehat{e}$ is also a safe reduction of e.*

Proof We will start by showing validity of the implication $(H \models e) \Rightarrow (H \models \tilde{e})$. For contradiction, let us assume that $H \models e$ and that there is a model M of the clausal theory H such that $M \models e$ and $M \not\models \tilde{e}$. Then there must be a substitution θ grounding all variables in $\tilde{e}$ such that[4] $M \models e\theta$ and $M \not\models \tilde{e}\theta$. Now, we will construct another model M' of H in which e will not be satisfied. We take each constant c in e that has been replaced by a variable V in $\tilde{e}$ and update the assignment ϕ of the constants c to objects from the domain of discourse in the model[5] so that $\phi(c) = \phi(V\theta)$. Clearly, we can do this for every constant c since every constant in e has been replaced by exactly one variable. Now, we see that $M' \not\models e$. However, we are not done yet as it might happen that the new model with the modified ϕ would no longer be a model of H. However, this is clearly not the case since none of the constants c appears in H and therefore the change of ϕ has no effect whatsoever on whether or not H is true in M'. So, we have arrived at a contradiction. We have a model M' such that $M' \models H$ and $M' \not\models e$ which contradicts the assumption $H \models e$. The implication $(H \models e) \Leftarrow (H \models \tilde{e})$ follows directly from Lemma 2. We have $\tilde{e} \preceq_\theta e$ therefore also $\tilde{e} \models e$ and finally $H \models \tilde{e} \models e$. (ii) In order to show $(H \models e) \Rightarrow (H \models \hat{e})$, it suffices to notice that $H \models e$ and $e \preceq_\theta \hat{e}$ imply $H \models \hat{e}$. The implication $(H \models e) \Rightarrow (H \models \hat{e})$ may be shown similarly as follows: $H \models \hat{e}$ and $\hat{e} \preceq_\theta e$ imply $H \models e$.

[4] We are applying θ also to e because e need not be ground.

[5] A first-order interpretation consists of several components, one them is the *domain of discourse*, and another is a function ϕ which maps constants to elements of the domain of discourse. We assume w.l.o.g that there is a constant c_d for every element d of the domain of discourse.

The next lemmas are used to prove Proposition 8. In Lemmas 3 and 4, we formulate rather general conditions under which x-equivalence w.r.t. a set X implies safe equivalence w.r.t. to the set 2^X. Then, in the subsequent lemmas, we show that these conditions are satisfied by the set of bounded-treewidth Horn clauses.

Lemma 3 *Let X be a set of clauses and let $\mathcal{L} \subseteq 2^X$ be a hypothesis language. Let A and B be clauses. Let $A \approx_X B$ w.r.t. the set X and let the following be true for any $\mathcal{H} \in \mathcal{L}$ and any clause C: if $\mathcal{H} \models C$ and C is not a tautology then there is a clause $D \in X$ such that $\mathcal{H} \models D$ and $D \preceq_\theta C$. Then for any $\mathcal{H} \in \mathcal{L}$, it holds $(\mathcal{H} \models A) \Leftrightarrow (\mathcal{H} \models B)$.*

Proof First, we need to consider the case when A and B are both tautologies. If both A and B are tautologies then $(\mathcal{H} \models A) \Leftrightarrow (\mathcal{H} \models B)$ naturally holds for any $\mathcal{H}$. Now, we can consider the case when at most one of the clauses A and B is a tautology. Let us assume w.l.o.g. that if one of the clauses is a tautology then it is the clause B. If $\mathcal{H} \models A$ then there is a clause $D \in X$ such that $\mathcal{H} \models D$ and $D \preceq_\theta A$ (by the assumptions of the proposition). Since $D \in X$, $D \preceq_\theta A$ and $A \preceq_X B$, we have $D \preceq_\theta B$ (from the definition of x-subsumption) and finally also $\mathcal{H} \models D \models B$ and so $\mathcal{H} \models B$. The other implication can be shown in a completely similar fashion if A is not a tautology.

Lemma 4 *Let X be a set of Horn clauses such that any clause which can be derived from a clausal theory $\mathcal{H} \in 2^X$ using SLD resolution is contained in X. If e and $\widehat{e}$ are two Horn clauses such that $e \approx_X \widehat{e}$ then for any $\mathcal{H} \in 2^X$: $(\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \widehat{e})$.*

Proof We will use the subsumption theorem for Horn clauses and Lemma 3. We will show that the conditions of this lemma imply conditions of Lemma 3. If A is a non-tautological clause and $\mathcal{H} \models A$ then by the subsumption theorem there must be a clause C derivable from $\mathcal{H}$ using resolution (SLD resolution, respectively) such that $C \preceq_\theta A$. Therefore for any non-tautological clause A, if $\mathcal{H} \models A$ where $\mathcal{H} \in 2^X$ then there must be a clause $C \in X$ such that $\mathcal{H} \models C$ (because resolution is sound) and $C \preceq_\theta A$. Now, since $e \approx_X \widehat{e}$, we may finish the proof using Lemma 3 which gives us $(\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \widehat{e})$ for any $\mathcal{H} \in 2^X$.

Lemma 5 (Clique containment lemma [4]) *Let A be a clause and T_A be its tree decomposition. For any $l \in A$, there is a vertex in T_A labelled by a set of variables V such that $vars(l) \subseteq V$.*

Proof The proof can be found in [4]. More precisely, the paper [4] contains a lemma which states that if C is a clique in a graph $G = (V, E)$ then any tree decomposition of G contains a vertex labelled by a set of vertices L such that $C \subseteq L$. Our statement of this lemma in terms of clauses and tree decompositions of clauses then follows immediately from this result which can be shown by noticing that the decomposition of the clause A can be easily converted to a tree decomposition of A's Gaifman graph G_A where, for any $l \in A$, $var(l)$ corresponds to a clique in G_A.

Lemma 6 *Let A be a function-free clause and T_A be its tree decomposition. Let $l^* \in A$ be a literal and let θ be a substitution not affecting any variable in the set $vars(A) \setminus vars(l^*)$, mapping variables to variables or terms (i.e. not to function symbols) and never mapping any variables to elements of the set $vars(A)$. Then a tree decomposition of $A\theta$ can be obtained by applying the substitution θ on the variables contained in the labels of the tree decomposition T_A and removing constants from these sets if necessary – we denote the new labelled tree by $T_A\theta$. As a consequence, the treewidth of $A\theta$ is never greater than the treewidth of A.*

Proof If we apply the substitution θ on the labels of the tree decomposition T_A then none of the label-sets associated to the vertices of T_A increases in size (this is in part due to the fact that we do not consider function symbols). Therefore if we are able to show that $T_A\theta$ is a tree decomposition of $A\theta$ then we will automatically get also the result that the treewidth of $A\theta$ is not greater than the treewidth of A. So, let us show that $T_A\theta$ is indeed a tree decomposition of $A\theta$.

(i) Claim: *For every variable* $V \in vars(A\theta)$ *there is a vertex of* $T_A\theta$ *labelled by a set containing* V. This is obvious.

(ii) Claim: *For every pair of variables* U, V *which both appear in a literal* $l \in A\theta$, *there is a vertex of* $T_A\theta$ *labelled by a set containing both* U *and* V. There must be a literal $l' \in A$ containing two variables U' and V' such that $U'\theta = U$ and $V'\theta = V$ (because U and V are both contained in a literal). Therefore there must be a vertex t of T_A labelled by a set S_t containing both U' and V'. After applying the substitution θ on the set S_t, we get a set by which some vertex contained in $T_A\theta$ is labelled and it contains both U and V.

(iii) Claim: *For every* $V \in vars(A\theta)$, *the set of vertices of* $T_A\theta$ *labelled by sets containing* V *forms a connected subgraph of* $T_A\theta$. Let us assume (for contradiction) that there is a variable $V \in vars(A\theta)$ such that the set of vertices of $T_A\theta$ labelled by sets containing V forms a disconnected graph. It follows that there must be two variables U', V' ($U' \neq V'$) such that $U'\theta = V'\theta = V$ and the sets of vertices $S_{U'}$ and $S_{V'}$ of the tree decomposition T_A corresponding to the variables U' and V', respectively, must be disjoint (because the set of vertices with labels containing a given variable must form a connected subgraph in any tree decomposition). However, the variables U' and V' must appear in the literal l^* because the substitution θ affects only variables contained in l^* and maps no variables of A to elements of the set $vars(A)$ (since at least one of the variables must have been affected by the substitution and since it is equal to the other variable, the other variable must have been affected by the substitution as well). Thus, since both U' and V' must be contained in l^* there must be a vertex in the tree decomposition T_A labelled by a set which contains both U' and V'. The sets of vertices of T_A labelled by sets containing the variables U' and V', respectively, therefore cannot be disjoint which is a contradiction.

We have thus shown that $T_A\theta$ is a tree decomposition of $A\theta$.

Lemma 7 *Let* $A = l_1 \vee l_2 \vee \cdots \vee l_m$ *and* $B = m_1 \vee m_2 \vee \cdots \vee m_n$ *be two standardized-apart function-free clauses. Let* θ *be a most general unifier of the literals* l_i, $\neg m_j$ *not affecting any variable in the set* $(vars(A) \cup vars(B)) \setminus (vars(l_i) \cup vars(m_j))$ *such that* $vars(l_i\theta) \cap vars(A) = vars(l_i\theta) \cap vars(B) = \emptyset$. *Next, let*

$$C = (l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_m \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta$$

be a binary resolvent of A *and* B. *Then for the treewidth* k_C *of* C, *it holds* $k_C \leq \max\{k_A, k_B\}$ *where* k_A *is the treewidth of* A *and* k_B *is the treewidth of* B.

Proof Using Lemma 6, we get that $A\theta$ has a tree decomposition $T_{A\theta}$ of width at most k_A and $B\theta$ has a tree decomposition $T_{B\theta}$ of width at most k_B. We will now show how to construct a tree decomposition of width at most $\max\{k_A, k_B\}$ for the clause C. Let V_A (V_B, respectively) be a vertex from $T_{A\theta}$ ($T_{B\theta}$, respectively) which is labelled by a set of variables $\mathcal{V}_A$ ($\mathcal{V}_B$, respectively) such that $vars(l_i\theta) \subseteq \mathcal{V}_A$ ($vars(l_i\theta) \subseteq \mathcal{V}_B$) – such vertices must exist by Lemma 5. We construct the new tree decomposition T_C by connecting $T_{A\theta}$ and $T_{B\theta}$ by a new edge between the vertices V_A and V_B (we may remove the variables not contained in the clause C from the labels of the vertices of T_C). Clearly, T_C has width at most $\max\{k_A, k_B\}$. We need to show that it is indeed a tree decomposition of C. The first two conditions from Definition 7 are trivially satisfied which follows from the fact that $T_{A\theta}$ and $T_{B\theta}$ are tree decompositions of the two clauses $A\theta$ and $B\theta$ and from $C \subseteq A\theta \cup B\theta$. It remains to show validity of the third condition (connectedness).

Let us assume (for contradiction) that there is a variable $V \in vars(C)$ such that the vertices labelled by sets containing the variable V form a disconnected subgraph of T_C. The variable V cannot be contained in $vars(A)$ or $vars(B)$. If $V \in vars(A)$ then $V \notin vars(B)$ (because A and B were standardized apart) and also $V \notin vars(l_i\theta)$ (because we selected the unifier θ to satisfy $vars(l_i\theta) \cap vars(A) = \emptyset$) but then the set of vertices labelled by sets containing the variable V could not be disconnected because it is actually connected in $T_{A\theta}$ and none of the labels in $T_{B\theta}$ contains V. The same argument can be used to show that $V \notin vars(B)$. So the only remaining possibility is that $V \in vars(l_i\theta)$. However, this is not possible either. Since both $T_{A\theta}$ and $T_{B\theta}$ are tree decompositions, the set of vertices labelled by the sets containing the variable V forms a connected subgraph in both $T_{A\theta}$ and

$T_{B\theta}$. Moreover, a vertex from $T_{A\theta}$ and a vertex from $T_{B\theta}$ which are both labelled by sets containing all variables from $vars(l\theta)$ are connected by an edge in T_C therefore the set of vertices labelled by the sets containing the variable V must form a connected subgraph of T_C. Thus, we have arrived at a contradiction because there cannot be any variable with a disconnected subgraph of T_C associated to it.

We have verified that T_C is a tree decomposition of C with width at most $\max\{k_A, k_B\}$.

Lemma 8 *Let X_k be the set of all function-free Horn clauses with treewidth at most k. Then for any clause C derivable by SLD resolution from a clausal theory $\mathcal{H} \in 2^{X_k}$ it holds $C \in X_k$.*

Proof Since this lemma considers only SLD resolution, we can consider just the case of binary resolvents (we do not need to take factors into account). The proposition then follows immediately from Lemma 7 because any clause derived by applying the binary resolution rule on two clauses must always have treewidth bounded by the treewidth of the clauses from which it was derived.

Proposition 8. *Let X_k be the set of all function-free Horn clauses with treewidth at most k and let $\mathcal{L} = 2^{X_k}$ be the set of theories consisting of function-free Horn clauses with treewidth at most k. Then any two clauses which are x-equivalent w.r.t. X_k are also safely equivalent w.r.t. $\mathcal{L}$.*

Proof Follows directly from Lemmas 4 and 8.

References

1. Appice, A., Ceci, M., Rawles, S., Flach, P.A.: Redundant feature elimination for multi-class problems. In: ICML, vol. 69 (2004)
2. Atserias, A., Bulatov, A., Dalmau, V.: On the power of k-consistency. In: Proceedings of ICALP-2007, pp. 266–271 (2007)
3. Beeri, C., Fagin, R., Maier, D., Yannakakis, M.: On the desirability of acyclic database schemes. Journal of ACM **30**(3), 479–513 (1983)
4. Bodlaender, H.L., Mohring, R.H.: The pathwidth and treewidth of cographs. SIAM Journal of Discrete Methematics pp. 238 – 255 (1993)
5. Courcelle, B.: The monadic second-order logic of graphs. i. recognizable sets of finite graphs. Information and Computation **85**(1), 12–75 (1990)
6. De Raedt, L.: Logical settings for concept-learning. Artif. Intell. **95**(1), 187–201 (1997)
7. De Raedt, L.: Logical and relational learning. Springer (2008)
8. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers (2003)
9. Erickson, J.: CS 598: Computational Topology, course notes, University of Illinois at Urbana-Champaign (2009). URL `http://compgeom.cs.uiuc.edu/~jeffe/teaching/comptop/`
10. Fagin, R.: Degrees of acyclicity for hypergraphs and relational database schemes. J. ACM **30**(3), 514–550 (1983)
11. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. SIAM J. Comput. **28**(1), 57–104 (1998)
12. Freuder, E.C.: Complexity of k-tree structured constraint satisfaction problems. In: Proceedings of the eighth National conference on Artificial intelligence - Volume 1, AAAI'90, pp. 4–9. AAAI Press (1990)
13. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer (2001)
14. Helma, C., King, R.D., Kramer, S., Srinivasan, A.: The predictive toxicology challenge 2000-2001. Bioinformatics **17**(1), 107–108 (2001)
15. Krogel, M.A., Rawles, S., Železný, F., Flach, P., Lavrac, N., Wrobel, S.: Comparative evaluation of approaches to propositionalization. In: ILP. Springer (2003)

16. Kuželka, O., Szabóová, A., Železný, F.: Bounded least general generalization. In: ILP'12: Inductive Logic Programming (2013)
17. Kuželka, O., Szabóová, A., Železný, F.: Reducing examples in relational learning with bounded-treewidth hypotheses. In: New Frontiers in Mining Complex Patterns, pp. 17–32 (2013)
18. Kuželka, O., Železný, F.: Block-wise construction of acyclic relational features with monotone irreducibility and relevancy properties. In: ICML 2009: the 26th Int. Conf. on Machine Learning
19. Kuželka, O., Železný, F.: Block-wise construction of tree-like relational features with monotone reducibility and redundancy. Machine Learning **83**, 163–192 (2011)
20. Kuželka, O., Železný, F.: Seeing the world through homomorphism: An experimental study on reducibility of examples. In: ILP'10: Inductive Logic Programming, pp. 138–145 (2011)
21. Landwehr, N., Kersting, K., Raedt, L.D.: Integrating naïve bayes and FOIL. Journal of Machine Learning Research **8**, 481–507 (2007)
22. Lavrač, N., Gamberger, D., Jovanoski, V.: A study of relevance for learning in deductive databases. Journal of Logic Programming **40**(2/3), 215–249 (1999)
23. Liu, H., Motoda, H., Setiono, R., Zhao, Z.: Feature selection: An ever evolving frontier in data mining. Journal of Machine Learning Research - Proceedings Track **10**, 4–13 (2010)
24. Mackworth, A.: Consistency in networks of relations. Artificial Intelligence **8**(1), 99–118 (1977)
25. Maloberti, J., Sebag, M.: Fast theta-subsumption with constraint satisfaction algorithms. Machine Learning **55**(2), 137–174 (2004)
26. Muggleton, S.: Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming **13**(3-4), 245–286 (1995)
27. Nassif, H., Al-Ali, H., Khuri, S., Keirouz, W., Page, D.: An Inductive Logic Programming approach to validate hexose biochemical knowledge. In: Proceedings of the 19th International Conference on ILP, pp. 149–165. Leuven, Belgium (2009)
28. Nienhuys-Cheng, S.H., de Wolf, R. (eds.): Foundations of Inductive Logic Programming, *Lecture Notes in Computer Science*, vol. 1228. Springer (1997)
29. Plotkin, G.: A note on inductive generalization. Edinburgh University Press (1970)
30. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier (2006)
31. Žáková, M., Železný, F., Garcia-Sedano, J., Tissot, C.M., Lavrač, N., Křemen, P., Molina, J.: Relational data mining applied to virtual engineering of product designs. In: ILP06, *LNAI*, vol. 4455, pp. 439–453. Springer (2007)