Mach Learn (2011) 83: 163–192
DOI 10.1007/s10994-010-5208-5

Block-wise construction of tree-like relational features with monotone reducibility and redundancy

Ondřej Kuželka · Filip Železný

Received: 20 September 2009 / Revised: 14 February 2010 / Accepted: 27 May 2010 /
Published online: 19 August 2010

Abstract We describe an algorithm for constructing a set of tree-like conjunctive relational features by combining smaller conjunctive blocks. Unlike traditional level-wise approaches which preserve the monotonicity of frequency, our block-wise approach preserves monotonicity of feature reducibility and redundancy, which are important in propositionalization employed in the context of classification learning. With pruning based on these properties, our block-wise approach efficiently scales to features including tens of first-order atoms, far beyond the reach of state-of-the art propositionalization or inductive logic programming systems.

Keywords Inductive logic programming · Relational machine learning · Propositionalization

1 Introduction

Propositionalization aims at converting structured descriptions of examples into attribute-value descriptions which can be processed by most established machine learning algorithms. A major stream of propositionalization approaches (Lavrač and Flach 2001; Krogel et al. 2003; Železný and Lavrač 2006) proceeds by constructing a set of features (first-order formulas) which follow some prescribed syntactical constraints and play the role of Boolean attributes. Here we assume that examples are represented as first-order Herbrand interpretations and features are conjunctions of first-order function-free atoms. Feature F acquires value *true* for example I (*covers* the example) if $I \models F$, otherwise it has the *false* value.

Editors: Hendrik Blockeel, Karsten Borgwardt, Luc De Raedt, Pedro Domingos, Kristian Kersting, Xifeng Yan.

O. Kuželka (✉) · F. Železný
Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic
e-mail: kuzelon2@fel.cvut.cz

F. Železný
e-mail: zelezny@fel.cvut.cz

Features may be viewed as patterns taking the form of relational queries. Thus propositionalization is similar to the framework of *frequent query discovery* where one seeks queries that are frequent in that they hold true for at least a specified minimum number of examples. This framework is represented e.g. by system WARMR (Dehaspe and Toivonen 1999). Indeed, propositionalization systems such as RSD (Železný and Lavrač 2006) also adopt a minimum frequency condition that admissible features must comply with. As well known, the minimum frequency condition is very practical in the popular *level-wise* approach to construct conjunctions. In this approach, followed also by the mentioned systems, each conjunction in level n extends by one atom some conjunction in level $n-1$. Frequency is *monotone* in that if F is not frequent then $F \wedge a$ is not frequent for any atom a. Thus all descendants of an infrequent query may be safely pruned, substantially adding to the efficiency of the level-wise systems.

The problem is that frequency is arguably not the ideal criterion to assess the quality of features in the propositionalization framework when employed for classification learning. Here, rather than frequent features, one naturally prefers features that well discriminate examples in terms of pre-defined classes. For sakes of pruning, one may want to discard *redundant* features, i.e. those for which another feature providing better discrimination exists. Unfortunately, redundancy is not monotone in the level-wise approach and thus cannot be translated here into a pruning mechanism similar to the one based on frequency. Besides redundancy, another important property of features is *reducibility w.r.t. θ-subsumption*. Given the assumed relational representation, two queries may be semantically equivalent despite their difference in syntax. We have good reason—if only for sakes of interpretability—to discard all *reducible* features, i.e. those equivalent to an existing feature that is syntactically simpler. Reducibility is unfortunately neither monotone in the level-wise approach. For example $p(X)$ is irreducible, $p(X) \wedge p(Y)$ is reducible, and $p(X) \wedge p(Y) \wedge q(X,Y)$ is again irreducible.

The purpose of this work is to remove these deficiencies by elaborating a novel, *block-wise* approach to construct a feature set by identifying building blocks (smaller conjunctions) out of which all features can be composed. The main assumption on which our algorithm is built is that the features it constructs, when viewed graphically, correspond to hypertrees while blocks correspond to their subtrees. Our feature construction strategy is bottom-up so that the initial set of blocks corresponds to all leaves of possible features. Blocks are then progressively combined together with further connecting atoms into larger blocks and eventually into features.

The main advantage of this approach is that it facilitates monotonicity of the two properties of interest in that features containing a redundant (reducible) block are themselves redundant (reducible). Such blocks are thus immediately and safely discarded. As we also show, the assumption of tree-like features would not guarantee monotonicity of the two properties in the more traditional level-wise approach. Thanks to the mentioned assumption, we are able to decide whether a block is reducible in time only quadratic in the size of the block despite the NP-completeness of the reducibility problem for unconstrained conjunctions. As a last advantage, the block-wise approach also allows us to efficiently compute extensions of features in a way similar to the well known *query pack* technique (Blockeel et al. 2002). In experiments, we show our approach to result in both very fast feature construction but also in high classification accuracies.

This paper extends our initial study (Kuželka and Železný 2009) by the full elaboration of proofs, a significantly more extensive experimental evaluation, but also by new technical enhancements such as the quadratic-time algorithm for reducibility checking.

The rest of the paper is organized as follows. In the next section we define the concepts of templates and features. Templates are used to specify the syntactic bias for features. Section 3 shows that the basic conditions we have imposed on templates and features imply the hypertree structure of features. In Sect. 4 we introduce blocks and the *graft* operation through which blocks are combined. Section 5 shows how to detect reducible blocks and establishes that reducibility is block-wise monotone. An efficient algorithm for checking reducibility is then described in Sect. 6. In Sect. 7 we formalize the semantics of features through the concepts of domain and extension. Using these concepts, Sect. 8 defines redundancy of features and shows how to detect blocks whose inclusion in any feature will make that feature redundant. In Sect. 9 we synthesize the methodological ingredients into a propositionalization algorithm and show that it is complete in that it generates all useful features complying with the given template. Section 10 subjects our algorithm to comparative experimental evaluation in 9 relational classification benchmarks. In Sect. 11 we discuss related work and Sect. 12 concludes the study. All proofs of theorems as well as all lemmas are located in the Appendix.

2 Features

We will be working with the language of first-order predicate logic free of functions up to constants. Learning examples and features will correspond to interpretations and existentially quantified constant-free atom conjunctions, respectively. We will write first-order logic variables in upper-cases and constants in lower-cases. As the order of atoms in conjunctions will not be important, we will use set-notation even for conjunctions. So expressions such as $|C|$, $a \in C$, $C \subseteq C'$ will be interpreted as if C and C' were sets of atoms. A set of conjunctions is said to be *standardized-apart* if no two conjunctions in the set share a variable. By $vars(C)$ we denote the set of all variables found in conjunction C. Given a set A of atoms, we denote $args(A) = \{(a, n) | a \in A, 1 \leq n \leq \text{arity}(a)\}$, i.e. $args(A)$ is the set of all argument places in A. For an atom a, $arg_i(a)$ is its i-th argument. Atoms a_1 and a_2 in conjunction C are *connected in* C if a_1 shares a term with a_2 or with some atom a_3 of C that is connected to a_2. Conjunction C is said to be *connected* if every two atoms in C are connected in C. Otherwise C is *disconnected*.

A feature construction algorithm should provide the user with suitable means of constraining the feature syntax. In commonplace inductive logic programming systems such as Progol (Muggleton 1995) or Aleph, the syntax is specified through *mode declarations* expressed via dedicated meta-predicates. Mode declarations define the predicates which can be used to build a clause, and assign a *type* and *mode* to each argument of these predicates. Here we propose the notion of a feature *template*, which is equally expressive to mode declarations, yet formally simpler. Besides simplicity, another advantage of feature templates is that compliance therewith is verified through the standard subsumption check. A template will be defined as a set of ground atoms of which all arguments fall in exactly one of two categories ('input' and 'output'). Templates will be further subjected to two restrictions needed to guarantee acyclicity of features that are to be derived from templates.

Definition 1 (Template) A *pre-template* is a pair $\tau = (\gamma, \mu)$ where γ is a finite set of ground atoms and $\mu \subseteq args(\gamma)$. Elements of μ are called input arguments and elements of $args(\gamma) \setminus \mu$ are called output arguments. We say that τ is a *template* if (i) every atom in γ has at most one input argument and (ii) there is a partial irreflexive order $\prec$ on constants in γ such that $c \prec c'$ if and only if there is an atom a with c in its input argument and c' in its output argument.

A template $\tau = (\gamma, \mu)$ is conveniently represented by writing γ with elements of μ ($args(\gamma) \setminus \mu$) marked with $+$ ($-$) signs, called *input* (*output*) *modes*. We will use the sign $\approx$ to associate τ with this kind of template representation as in the following example.

Example 1 Let $\tau = (\gamma, \mu)$ where $\gamma = \{hasCar(c), hasLoad(c, l), box(l), tri(l), circ(l)\}$ and $\mu = \{(hasLoad(c, l), 1), (box(l), 1), (tri(l), 1), (circ(l), 1)\}$. Here, τ is a template since there is an order $c \prec l$ complying with Definition 1 and no atom in γ has two input arguments. In the simplified notation we show τ as

$$\tau \approx hasCar(-c), hasLoad(+c, -l), box(+l), tri(+l), circ(+l)$$

On the other hand, $\tau_2 \approx a(+x, -y), b(+y, -x)$ is not a template because there is no order that would comply with Definition 1.

We will now define the type of connected conjunctions that comply with a given template τ. The compliance will be witnessed by a substitution θ. We shall distinguish certain roles of variables in such conjunctions; these will be determined by τ and θ.

Definition 2 (τ_θ-conjunction) Given a template $\tau = (\gamma, \mu)$ a *τ_θ-conjunction* C is a connected conjunction where all terms are variables and $C\theta \subseteq \gamma$. The occurrence of a variable in the i-th argument of atom a in C is an *τ_θ-input* occurrence in C if the i-th argument of $a\theta$ is in μ and it is an *τ_θ-output* otherwise. A variable is:

- *τ_θ-positive* in C if it has exactly one τ_θ-input occurrence and no τ_θ-output occurrence
- *τ_θ-negative* in C if it has exactly one τ_θ-output occurrence and no τ_θ-input occurrence
- *τ_θ-neutral* in C if it has at least one τ_θ-input and exactly one τ_θ-output occurrence

Example 2 Conjunction $hasLoad(C, L) \wedge box(L)$ is a τ_θ-conjunction for τ from Example 1 and $\theta = \{C/c, L/l\}$. Variable C is τ_θ-positive and variable L is τ_θ-neutral in the conjunction.

In what follows, we will abbreviate the phrases 'τ_θ-input occurrence' and 'τ_θ-output occurrence' by the respective words 'input' and 'output.'

Through the condition $C\theta \subseteq \gamma$, the definition effectively requires that variables in τ_θ-conjunction C comply with *types* specified by γ, specifically that each variable in C maps to exactly one type. Features, which we define in turn, are τ_θ-conjunctions which additionally comply to a condition pertaining to *modes*.

Definition 3 (Feature) A τ_θ-conjunction F is a *τ_θ-feature* if all its variables are τ_θ-neutral in F. We say that F is a τ-feature if there exists a substitution θ such that F is a τ_θ-feature.

For example, using τ from Example 1, the following conjunction is a τ-feature

$$hasCar(C) \wedge hasLoad(C, L1) \wedge hasLoad(C, L2) \wedge box(L1) \wedge tri(L2)$$

as it is a τ_θ-feature for $\theta = \{C/c, L1/l, L2/l\}$. In general, there can be more than one substitution θ such that $F\theta \subseteq \gamma$, and some of them can make some of the variables non-neutral. For example, if $\tau = atom(-a), bond(+a, -b), bond(-c, +b), atom(+c)$ and $F = atom(A), bond(A, B), bond(C, B), atom(C)$ then F is a τ_{θ_1}-feature for $\theta_1 = \{A/a, B/b, C/c\}$ but it is not τ_{θ_2}-feature for $\theta_2 = \{A/c, B/b, C/c\}$. In any case, F is a τ-feature because of the existence of θ_1. It would be easy to avoid the illustrated ambiguity of templates, e.g.

by requiring that no predicate occurs twice in a template. However, we do not need this restriction.[1]

Templates introduced in this section are similar to WARMR's warmode declarations but not the same. In a template, every atom can have at most one input argument, unlike warmodes which do no restrict the number of input or output arguments. We do need the restriction to guarantee tree-like structure of features. Furthermore, variables in our features must have both an input and an output. This requirement increases the expressiveness of our framework (the case without the requirement can clearly be emulated by adding dummy atoms). Further justification for the requirement is given in Lavrač and Flach (2001).

3 Tree-like structure of features

The two conditions we imposed on templates in Definition 1 and the neutrality condition we stipulated on variables in features in Definition 3 guarantee a constrained structure of features when viewed graphically. For precise characterization, we shall define tree-likeness of conjunctions which is a special case of acyclicity. We follow the established definition from Fagin (1983) that relates to hypergraphs. While we translate the notions into the context of conjunctions, the definition also shows the original hypergraph concepts in parentheses since hypergraphs offer easier intuitive understanding.

Definition 4 (Tree-like conjunction) A conjunction (hypergraph) C is *tree-like* if the iteration of the following rules on C produces the empty conjunction (hypergraph):

1. Remove an atom (hyperedge) which contains fewer than 2 terms (vertices).
2. Remove a term (vertex) which is contained in at most one atom (hyperedge).

The right-most panel in Fig. 1 illustrates the concept of tree-likeness. A few remarks are needed to fully clarify the correspondence between conjunctions and hypergraphs. Vertices correspond to terms and hyperedges correspond to atoms that contain the terms as arguments. In general, multiple atoms in a conjunction may contain the same set of terms. Therefore conjunctions correspond in fact to *multi-hypergraphs*. This fact is not important when applying the iterative reduction above and thus for brevity we maintain the shorter

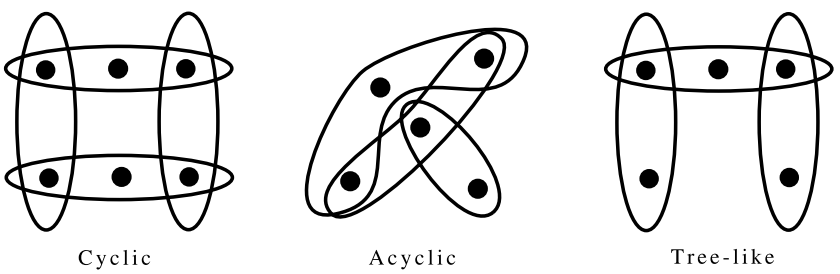

Fig. 1 Examples of cyclic, acyclic and tree-like hypergraphs that represent first-order conjunctions

[1] In fact, with certain abstraction, this ambiguity is analogical to that present in inductive logic programming systems where different sequences of refinements may result in the same first-order expression.

term *hypergraph*. In Definition 4, removing a term from C means removing all its occurrences from C. That is, the arity of each atom in C will be decreased by the number of occurrences of the term in the atom.

Obviously, general τ_θ-conjunctions may be non-tree-like. For example, $C = a(A, B) \wedge a(C, B) \wedge a(C, D) \wedge a(A, D)$ is a τ_θ-conjunction for $\tau = (\gamma, \mu)$, $\gamma = \{a(a, b)\}$, $\theta = \{A/a, B/b, C/a, D/b\}$ and any $\mu \subseteq args(\gamma)$. C is not tree-like as may be verified by applying Definition 4 (clearly, none of the two reduction rules can be applied on C) or by projecting C onto the left-most example in Fig. 1. However, as the following theorem (proven in the Appendix) asserts, features are indeed tree-like.

Theorem 1 *Every τ-feature F is tree-like.*

A convenient consequence is that the decision problem $I \models F$ can be computed in time polynomial in $|F|$. In contrast, for a general conjunction C, $I \models C$ corresponds to the NP-complete θ-subsumption test. Polynomial-time solubility of tree-like queries is well known in database literature (Yannakakis 1981) and in the field of constraint satisfaction where this problem is efficiently tackled by the directional-arc-consistency algorithm (Dechter 2003).

4 Blocks

Here we first define *blocks* used to build a feature through the *graft* operation.[2] Before we do so formally, we motivate the concepts through an example. Given a template

$$\tau \approx hasCar(-c), hasLoad(+c, -l), has2Loads(+c, -l, -l), box(+l), tri(+l)$$

an exemplary τ_θ-feature for $\theta = \{C/c, L1/l, L2/l, L3/l\}$

$$hasCar(C) \wedge \tag{1}$$

$$hasLoad(C, L1) \wedge box(L1) \wedge \tag{2}$$

$$has2Loads(C, L2, L3) \wedge box(L2) \wedge box(L3) \tag{3}$$

is graphically shown left-most in Fig. 2 as a hypertree containing two subtrees corresponding to lines 2 and 3 above. The latter subtree is also shown separately in the middle of Fig. 2.

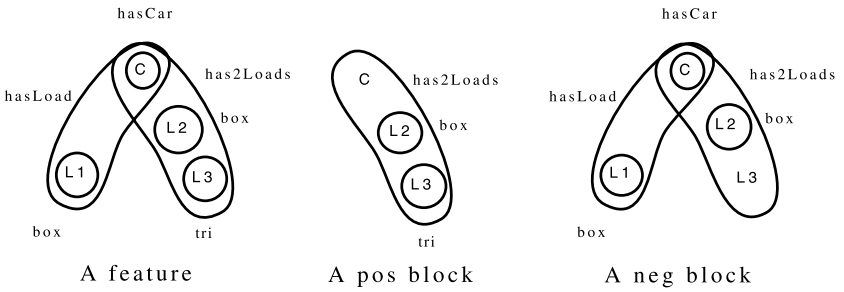

Fig. 2 A feature, a pos block and a neg block displayed as hypergraphs

[2]Not to be confused with grafting as introduced in Perkins and Theiler (2003).

Conjunctions corresponding to such subtrees will be called *pos τ_θ-blocks* to abbreviate the fact that they contain exactly one τ_θ-positive variable (here C). This is the only variable that the block shares with the rest of the feature. The share of only one variable has a convenient consequence towards monotonicity, as we will also formally show: if the block is redundant (reducible), the containing feature is also redundant (reducible).

The second kind of block we shall define corresponds to what is left of a τ_θ-feature when a pos τ_θ-block is removed. Such a block, called a neg τ_θ-block contains exactly one τ_θ-negative variable. An example of a neg τ_θ-block is shown right-most in Fig. 2.

Definition 5 (Blocks) Let B be a τ_θ-conjunction. If there is exactly one τ_θ-positive (τ_θ-negative) variable in B, it will be denoted $\mathrm{p}(B)$ ($\mathrm{n}(B)$). B is a *pos τ_θ-block* if it has $\mathrm{p}(B)$ ($\mathrm{n}(B)$) and all other variables in B are τ_θ-neutral. We say that B is a pos (neg) τ-block if there exists a substitution θ such that B is a pos (neg) τ_θ-block.

Of course, a pos τ_θ-block can itself in general contain smaller pos τ_θ-blocks. To exploit the above commented monotonicity observed on blocks, our strategy to assemble features will be *bottom-up*, starting with the smallest pos τ_θ-blocks (e.g. $tri(L3)$ in the current example) and safely discarding those deemed redundant or reducible. Larger pos τ_θ-blocks result from conjoining a number of smaller τ_θ-blocks (e.g. $box(L2)$ and $tri(L3)$) with further atoms needed to make the conjunction a pos τ_θ-block (in this case $has2Loads(C, L2, L3)$) or a τ_θ-feature. This operation, which we call a *graft*, is formalized below.

Definition 6 (Graft) Let C be a τ_θ-conjunction, v a variable in C, and $\phi^+ = \{B_i^+\}$ a standardized-apart set of pos τ-blocks. We define $C \oplus_v \phi^+ = C \wedge_i B_i\theta_i$ where $\theta_i = \{\mathrm{p}(B_i^+)/v\}$. In the special case when C is a neg τ-block, we denote $C \oplus \phi^+ = C \oplus_{\mathrm{n}(C)} \phi^+$.

In the mentioned special case when C is a neg τ-block, $C \oplus \phi^+$ is a τ_θ-feature. Otherwise, for generality, the definition does not require that the graft results in a pos τ-block. Speaking in terms conventional for inductive logic programming, graft is an operator yielding a common specialization of its operands. However, in comparison with well known operators such as Plotkin's least general generalization (Nienhuys-Cheng and de Wolf 1997), graft is defined in a highly constrained context of tree-like function-free conjunctions rather than general first-order clauses.

5 Reducibility

In this section, we elaborate properties of features and pos blocks regarding their reductions. Let C and D be conjunctions of atoms and let there be a substitution θ such that $C\theta \subseteq D$. Then we say that C *subsumes* D (written $C \preceq D$). If $C \preceq D$ and $D \preceq C$ then C and D are said to be *equivalent*, written $C \approx_\theta D$. We say that C is *reducible* if there exists a conjunction C' such that $C \approx_\theta C'$ and $|C| > |C'|$; C' is called a *reduction of* C. An example of a reducible conjunction of atoms is $hasCar(C) \wedge hasLoad(C, L1) \wedge hasLoad(C, L2) \wedge box(L1)$ equivalent to the shorter conjunction $hasCar(C) \wedge hasLoad(C, L1) \wedge box(L1)$.

In this work we cannot rely directly on the established notion of reducibility as defined above. This is because a reduction of a τ-feature may not be a τ-feature itself. For example, for $\tau \approx \{car(-c_1), hasLoad(+c_1, -l), hasLoad(-c_2, +l), hasCar(+c_2)\}$, the conjunction $car(C_1) \wedge hasLoad(C_1, L_1) \wedge hasLoad(C_2, L_1) \wedge car(C_2)$ is a correct τ-feature but its reduction $hasCar(C_1) \wedge hasLoad(C_1, L_1)$ is not. The fact that reduction does not preserve correctness of feature syntax may represent a problem because we would like to work only with

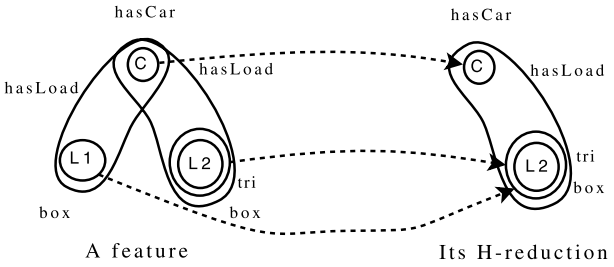

Fig. 3 The feature F from Example 3 (*left*) and its H-reduction (*right*); *arrows* indicate the substitution θ_1

reduced features, if only for sakes of improved interpretability. Relying on the acyclicity of features and the implied acyclicity of pos blocks, we are going to introduce H-subsumption and H-reduction such that H-reduction of a τ_θ-feature is always a τ_θ-feature. The two concepts differ from their classical counterparts by requiring that substitutions involved in the subsumption relations preserve variable *depths*. We define the notion of depth in turn.

Definition 7 (Depth) Let C be a τ_θ-feature or a pos τ-block and a its atom. If a contains no inputs or if it contains the variable $\mathrm{p}(C)$ then its *depth in* C, denoted $d_C(a)$ is 1. Otherwise $d_C(a) = n$ such that $a_1, a_2, \ldots, a_n$ is the shortest sequence of atoms such that $d_C(a_1) = 1$, $a_n = a$ and for each $1 \leq i < n$, there is a variable having an output in a_i and an input in a_{i+1}. The *depth of variable* v *in* C is $d_C(v) = \min\{d_C(a) | a \text{ contains } v\}$.

Definition 8 (H-subsumption) We say that τ_{θ_1}-feature (pos τ-block, respectively) F *H-subsumes* τ_{θ_2}-feature (pos τ-block, respectively) G (written $F \preceq_H G$) if and only if there is a substitution ϑ such that $F\vartheta \subseteq G$ and for every atom $a \in F$ it holds $d_F(a) = d_G(a\vartheta)$; such a ϑ is called a *H-substitution*.

Definition 9 (H-reduction) If $F \preceq_H G$ and $G \preceq_H F$, we call F and G *H-equivalent* (written $F \approx_H G$). We say that τ_{θ_1}-feature (pos τ-block, respectively) F is *H-reducible* if there is a τ_{θ_2}-feature (pos τ-block, respectively) F' such that $F \approx_H F'$ and $|F| > |F'|$. A τ_θ-feature (pos τ-block) F' is said to be an *H-reduction* of F if $F \approx_H F'$ and F' is not H-reducible. A τ-feature F is said to be *H-reducible* if there is a substitution θ such that F is an H-reducible τ_θ-feature.

Example 3 Continuing with τ from Example 1, the τ-feature

$$F_1 = \mathit{hasCar}(C) \wedge \mathit{hasLoad}(C, L1) \wedge \mathit{box}(L1) \wedge \mathit{hasLoad}(C, L2) \wedge \mathit{box}(L2) \wedge \mathit{tri}(L2)$$

is H-reducible because there is feature $F_2 = \mathit{hasCar}(C) \wedge \mathit{hasLoad}(C, L) \wedge \mathit{box}(L) \wedge \mathit{tri}(L)$ for which it holds $F_1 \approx_H F_2$, as $F_1\theta_1 \subseteq F_2$ and $F_2\theta_2 \subseteq F_1$ for substitutions $\theta_1 = \{C/C, L1/L, L2/L\}$, $\theta_2 = \{C/C, L/L2\}$, which map variables in depth d in one feature to variables in the same depth in the other feature (cf. Fig. 3), and it holds $|F_2| < |F_1|$.

Since H-reducibility is the only concept of reducibility we will work with, we will occasionally afford to omit the H-prefix. The next theorem provides a way to detect reducibility of pos blocks and also asserts that reducibility is monotone.

Theorem 2 *Let B be a pos τ-block and let B^- be a neg τ-block. Then the following holds*: (i) *B is H-reducible if and only if B contains pos τ-blocks B_1, B_2 such that $B_1 \neq B_2$,*

$\mathrm{p}(B_1) = \mathrm{p}(B_2)$ *and* $B_1 \preceq_H B_2$. (ii) *If* B *is* H*-reducible, then* $B^- \oplus (\{B\} \cup \mathcal{S})$ *is also* H*-reducible for any set of pos* τ*-blocks* $\mathcal{S}$.

Monotonicity of reducibility is an important property enabling pruning during feature construction. It is essential to realize that this property does not simply follow from the tree-like structure of features. Indeed, if we constructed tree-like features in the usual level-wise manner as e.g. the systems WARMR or RSD would proceed, it would not be possible to prune on reducibility. For example, extending $hasCar(C) \wedge hasLoad(C, L) \wedge box(L)$ by atom $hasLoad(C, L2)$ yields a reducible conjunction. If we however pruned it along with all of its descendants, we would also prune the irreducible feature $hasCar(C) \wedge hasLoad(C, L) \wedge box(L) \wedge hasLoad(C, L2) \wedge tri(L2)$.

Lastly, thanks to the reducibility concept, we can state the following theorem which is important since we aim at constructing *complete* feature sets.

Theorem 3 *Let* τ *be a template. There is only a finite number of* τ*-features, which are not* H*-reducible.*

This holds despite that there is an infinite number of τ-features for a sufficiently rich template τ.

Example 4 Let us work again with τ from Example 1. There is an infinite number of τ-features but there are only 19 irreducible τ-features. An example of a reducible τ-feature is

$$F_r = hasCar(C) \wedge hasLoad(C, L) \wedge box(L) \wedge hasLoad(C, L2) \wedge box(L2) \wedge circ(L2)$$

This τ-feature is indeed reducible, which according to Theorem 2 follows from the presence of the two pos blocks below in it.

$$hasLoad(C, L) \wedge box(L) \preceq_H hasLoad(C, L2) \wedge box(L2) \wedge circ(L2)$$

As may be routinely checked, there are exactly eighteen further irreducible τ-features.

6 A fast algorithm for H-reduction

We now describe an algorithm for checking H-reducibility of features (Algorithm 1) that runs in time quadratic in the number of atoms in the tested feature. We need to introduce some additional notation which will be constrained to this section. Let F be a τ_θ-feature or a pos τ_θ-block and P_F, I_F, J_F, functions defined as follows. If two atoms $a, b \in F$ share a variable, which has an output in the i-th argument of a and an input in the j-th argument of b we define $P_F(b) = a$, $I_F(b) = i$, $J_F(b) = j$. Lastly, for an atom b in F we define $C_F(b) = \{a \in F \mid P_F(a) = b\}$. That is, $P_F(a)$ denotes the 'parent' atom of a, which, due to the tree-like structure of features and pos blocks, is unique if it exists. Inversely, $C_F(b)$ is the set of all 'children' atoms of b. Functions I_F and J_F index the arguments in the respective literals where the 'connecting variable' occurs.

The algorithm is based on Theorem 2, namely on the fact that a τ_θ-feature F is H-reducible if and only if it contains two pos τ-blocks B_1 and B_2 such that $\mathrm{p}(B_1) = \mathrm{p}(B_2)$ and $B_1 \preceq_H B_2$. The main idea exploited by the algorithm is to maintain an array called *Subsumed*

Algorithm 1 H-Reducibility(F)

Input: τ_θ-feature (pos τ-block) F;
Subsumed ← a two-dimensional array indexed by atoms. All elements are initialized to 0. /* *Subsumed*(A, B) is the number of child-blocks of A, which have been found to H-subsume some child-blocks of B */
Open ← {} /* A stack data-structure */
for $d = 1 \ldots d_F$ **do**
  Open ← *Open* $\cup \{(a_i, a_j)\}$ where a_i, a_j $(a_i \neq a_j)$ are literals in depth d with no outputs and such that $predicate(a_i) = predicate(a_j)$ and $I_F(a_i) = I_F(a_j)$
end for
while *Open* $\neq \emptyset$ **do**
  $(a_i, a_j) \leftarrow Pop(Open)$
  if $P_F(a_i) = P_F(a_j)$ **then**
    return H-Reducible
  else
    $Subsumed(P_F(a_i), P_F(a_j)) = Subsumed(P_F(a_i), P_F(a_j)) + 1$
    if $Subsumed(P_F(a_i), P_F(a_j)) = |C_F(P_F(a_i))|$ **and** $predicate(P_F(a_i)) = predicate(P_F(a_j))$ **and** $I_F(P_F(a_i)) = I_F(P_F(a_j))$ **and** $J_F(P_F(a_i)) = J_F(P_F(a_j))$ **then**
      Open ← *Open* $\cup \{(P_F(a_i), P_F(a_j))\}$
    end if
  end if
end while
return Not H-Reducible

indexed by pairs (a_i, a_j) in which we store the number of child-blocks (i.e. pos τ-blocks) of the atom a_i, which H-subsume some child-blocks of the atom a_j such that $I_F(a_i) = I_F(a_j)$. If this number equals the number of children of a_i and if a_i and a_j share a common parent, then the tested τ-feature F is H-reducible. It is easiest to see how this algorithm works through an example.

Example 5 Consider template $\tau \approx a(-x), b(+x, -y), c(+y), d(+y)$ and the following feature

$$F = a(A) \wedge b(A, B) \wedge c(B) \wedge d(B) \wedge b(A, C), c(C)$$

Let us go through the steps Algorithm 1 performs to detect that F is H-reducible. It starts by filling the *Open* list: *Open* ← $\{(c(B), c(C)), (c(C), c(B))\}$. In the next step, the first pair $(c(B), c(C))$ is extracted from the list and we check whether $P_F(c(B)) = P_F(c(C))$. As the answer is negative, the algorithm continues by incrementing $Subsumed(b(A, B), b(A, C))$, i.e. we have $Subsumed(b(A, B), b(A, C)) = 1$. However, $Subsumed(b(A, B), b(A, C)) \neq |C_F(b(A, B))| = 2$, so we continue with the next iteration. We extract $(c(C), c(B))$ from the list *Open*. Again $P_F(c(B)) \neq P_F(c(C))$, therefore we increment $Subsumed(b(A, C), b(A, B))$. We have $Subsumed(b(A, C), b(A, B)) = |C_F(b(A, C))| = 1$, therefore we add $(b(A, C), b(A, B))$ to the *Open* list. In the next iteration, we extract $(b(A, C), b(A, B))$ from the list *Open* and check that $P_F(b(A, C)) = P_F(b(A, B))$. From this, we may conclude that F is H-reducible.

Theorem 4 *Algorithm 1 correctly decides whether a τ_θ-feature (pos τ-block) F is H-reducible in time $O(|F|^2)$.*

Algorithm 1, which only decides whether a τ-feature F is H-reducible, can be easily converted to an algorithm that also computes the H-reduction[3] of F. It suffices to replace line 10 in Algorithm 1 by '*Remove pos τ-block with root a_i*'.

7 Extensions and domains

So far we have been only concerned with the syntax of τ-features. Now we will also establish their semantics. We do this through the concepts of *domain* and *extension*. Extension of a τ-feature is simply the set of examples covered by the τ-feature. Domain is defined in a finer manner for both τ-features and pos τ-blocks, with respect to a single example. The method for computing domains and extensions will be based on a combination of a well-known algorithm for conjunctive acyclic query answering (Yannakakis 1981) and a variation of the query-pack method (Blockeel et al. 2002), integrated into our block-wise feature construction procedure.

Definition 10 (Domain, Extension) Let I be an interpretation, i.e. a set of ground atoms, and τ a template. For a pos τ-block B, *domain* $\mathrm{dom}_I(B)$ contains all terms t such that $I \models B\theta$, where $\theta = \{\mathrm{p}(B)/t\}$. For a τ-feature F, $\mathrm{dom}_I(F) = \{yes\}$ if $I \models F$ and $\mathrm{dom}_I(F) = \emptyset$ otherwise. For a set E of interpretations, the *extension* of a τ-feature F on E is defined as $\mathrm{ext}_E(F) = \{I \in E | I \models F\}$.

Domain, as defined in Definition 10, assigns to each pos τ-block B a set of terms $\{t_i\}$, for which there is a substitution θ of B such that $\mathrm{p}(B\theta) = t_i$ and $B\theta$ is true in I. Note that template τ is not indicated in the subscript of either dom or ext as both of them are independent of τ. This is instantly clear for features. For a pos τ-block B, we must recall from Sect. 2 that $\mathrm{p}(B)$ is identified uniquely as it is the only variable with a single occurrence in B. The domain of B may be computed by Algorithm 2 which runs in time polynomial in the size of B. This algorithm is not novel, it corresponds to the algorithm known as conjunctive acyclic query answering algorithm in database theory (Yannakakis 1981) and also to directed-arc-consistency algorithm in the field of CSP (Dechter 2003).

Algorithm 2 $\mathrm{dom}_I(B)$

1: **Input:** Pos τ-block B, Interpretation I;
2: $atomsDom \leftarrow \{a \in I | pred(a) = pred(root(B)) \wedge (I \models a)\}$
3: **for** $\forall$ output variables out_i of B's root **do**
4: $Children \leftarrow$ all pos blocks B such that $out_i = \mathrm{p}(B)$ (i.e. children of B's root hanging on its i-th output)
5: $argDom_i \leftarrow \bigcap_{c \in Children} \mathrm{dom}_I(c)$
6: $atomsDom \leftarrow atomsDom \cap \{a \in I | arg_i(a) \in argDom_i\}$
7: **end for**
8: **return** $\{t | t$ is term at input of an atom a and $a \in atomsDom\}$

[3]The basic principles of the algorithm can be also used to easily obtain an $O(|F| \cdot |G|)$ algorithm for deciding $F \preceq_H G$.

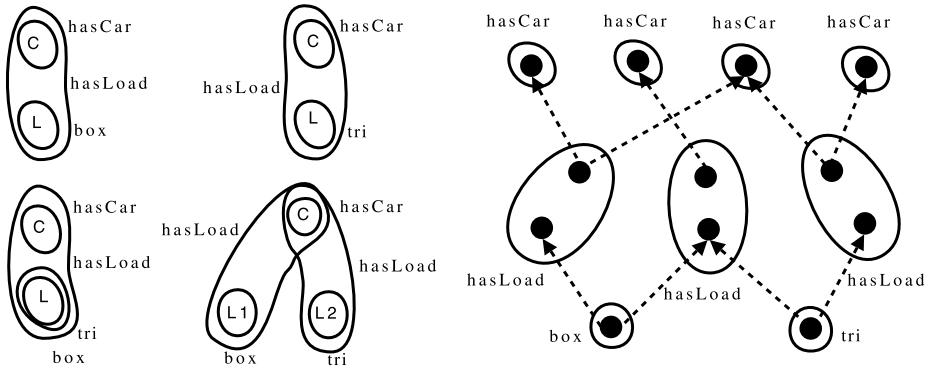

Fig. 4 *Left*: The set of all four non-H-reducible τ-features for $\tau \approx hasCar(-c), hasLoad(+c,-l), box(+l), tri(+l)$. *Right*: the structure of domain reusing when computing domains of these features. The *arrows* indicate the flow of domain information. If domains are reused, only 9 domains need to be computed, whereas, if features were treated in isolation from each other, 15 domains would need to be computed

Example 6 Consider feature F and interpretation I

$$F = hasCar(C) \wedge hasLoad(C, L) \wedge tri(L) \wedge box(L),$$
$$I = \{hasCar(c), hasLoad(c, l1), hasLoad(c, l2), tri(l1),$$
$$circ(l1), tri(l2), box(l2), hasLoad(c, l3), box(l4)\}.$$

We may proceed as follows: (i) We compute domains of pos τ-block $box(L)$ and $tri(L)$, i.e. $\mathrm{dom}_I(box(L)) = \{l2, l4\}$, $\mathrm{dom}_I(tri(L)) = \{l1, l2\}$. (ii) Then we compute domain of pos τ-block with root $hasLoad(C, L)$, i.e. $\mathrm{dom}_I(hasLoad(C, L) \wedge tri(L) \wedge box(L)) = \{l1, l2, l3\} \cap \{l2, l4\} \cap \{l1, l2\} = \{l2\}$. The first set in the intersection comes from the fact that the predicate symbol is $hasLoad/2$. The second set is due to pos τ-block $box(L)$ and the third set is due to pos τ-block $tri(L)$. (iii) Since no domain is empty so far, we proceed further and compute domain of the feature with root $hasCar(C)$, which becomes $\{yes\}$. So, we see that $I \models F$.

The example makes it clear that computation of domains can be combined with the earlier described block grafting in a way resembling query packs (Blockeel et al. 2002). When a pos block is constructed grafting an already constructed block c, the domain previously computed for c is reused as $\mathrm{dom}_I(c)$ in line 5 of Algorithm 2. This principle is further illustrated in Fig. 4. To prevent excessive memory demands possibly resulting from the storing of domains, our implementation uses a Java mechanism enabling it to automatically discard computed domains when a memory limit is reached.

8 Redundancy

We shall now define when a feature is redundant. We do so in the spirit of Lavrač et al. (1999) where redundancy was termed *irrelevancy*. Redundancy of a feature will be defined with respect to a feature set $\mathcal{F}$ and two sets of examples (interpretations) E^+ and E^- although, for brevity, we will not explicitly say 'with respect to $\mathcal{F}$, E^+ and E^-' when speaking of redundancy concepts. Adopting these concepts, our approach becomes limited to binary

Table 1 An example set with five features used to illustrate the concept of redundancy in Example 7

	F_1	F_2	F_3	F_4	F_5	Class
Example 1	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
Example 2	$\oplus$	$\oplus$	$\ominus$	$\ominus$	$\oplus$	$\oplus$
Example 3	$\ominus$	$\oplus$	$\oplus$	$\ominus$	$\ominus$	$\ominus$
Example 4	$\oplus$	$\ominus$	$\oplus$	$\oplus$	$\oplus$	$\ominus$

classification problems, including of course the case where a single target class is to be discriminated from a number of other classes.

Definition 11 (Redundancy) Let $\mathcal{F}$ be a set of τ-features for some template τ. Let E^+ and E^- be two sets of interpretations (the positive and negative examples, respectively). F is *E^+-redundant* (*E^--redundant*) if there is $F' \in \mathcal{F}$, $F \neq F'$ such that $\text{ext}_{E^+}(F) \subseteq \text{ext}_{E^+}(F')$ and $\text{ext}_{E^-}(F') \subseteq \text{ext}_{E^-}(F)$ ($\text{ext}_{E^-}(F) \subseteq \text{ext}_{E^-}(F')$ and $\text{ext}_{E^+}(F') \subseteq \text{ext}_{E^+}(F)$. If one of the inclusions, for at least one example, is strict, F is said to be *strictly E^+-redundant* (*E^--redundant*). F is called *redundant* if it is both E^+-redundant and E^--redundant. It is called *strictly redundant* if it is (i) redundant, and (ii) strictly E^+-redundant or strictly E^--redundant.

Example 7 Consider the set of examples and features as displayed in Table 1. F_4 is strictly E^+-redundant because it covers the same negative examples as F_1 but the set of positive examples covered by it is a strict subset of the set of positive examples covered by F_1. Feature F_4 is also E^--redundant because it covers the same set of positive examples as F_3 but the set of negative examples covered by F_4 is a strict subset of those covered by F_3. Therefore F_4 is also strictly redundant. F_1 and F_5 are non-strictly redundant because they cover the same set of examples.

In Lavrač et al. (1999), redundancy was introduced for attribute-valued learning and for learning of constrained Horn clauses, which are equivalent in their expressive power to attribute-valued representations. Applied to our setting, it is obvious that when features are constructed, they can be filtered in a post-processing step, i.e. strictly redundant features can be discarded and from each non-strictly redundant set of features with equal extensions, one feature can be preserved discarding the rest. What is however less obvious is that a form of redundancy can be detected already for pos blocks and that it implies redundancy of features that contain them. This is formalized in turn.

Theorem 5 *Let E^+ (E^-) be a set of positive (negative) examples. Let $\mathcal{F}$ be a set of pos τ-blocks with equal types of input arguments and let $\mathcal{B} = \{B_i\}_{i=1}^n \subseteq \mathcal{F}$. Let $\text{dom}_I(B_1) \subseteq \bigcap_{i=2}^n \text{dom}_I(B_i)$ for all $I \in E^+$ ($I \in E^-$) and let $\bigcap_{i=2}^n \text{dom}_I(B_i) \subseteq \text{dom}_I(B_1)$ be true for all $I \in E^-$ ($I \in E^+$). Then for any neg τ-block B^-: $B^- \oplus (\{B_1\} \cup \mathcal{S})$ is E^+-redundant (E^--redundant), where $\mathcal{S} \subseteq \mathcal{F}$.*

During the generation of features, we will discard pos τ-blocks such as B_1 characterized by the above theorem, i.e. those pos τ-blocks giving rise to redundant features. Let us use the adjective redundant also for such pos τ-blocks. We use the term *monotonicity of redundancy* to denote that once a pos block B is redundant, any block or feature constructed using B is redundant as well. Again, this kind of pruning is enabled by our block-wise strategy and

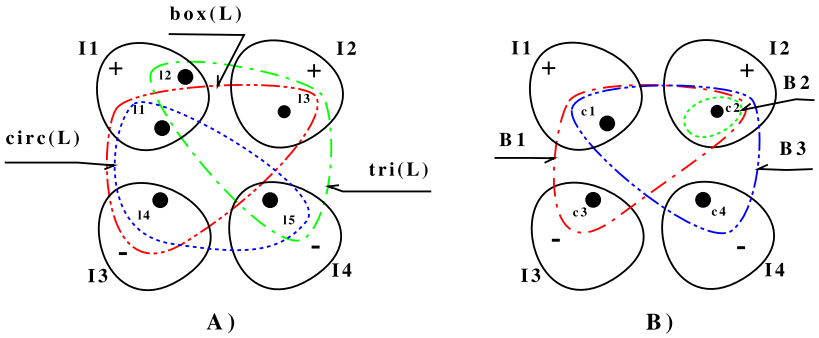

Fig. 5 Domains of pos blocks from Example 8. *Ovals* marked by I_1, I_2 correspond to the positive examples (but not to their hypergraph representation) and *ovals* marked by I_3, I_4 correspond to the negative examples. The *dashed/dotted/dash-dotted ovals* correspond to domains of the blocks. The *left panel* displays domains of blocks $box(L)$, $tri(L)$ and $circ(L)$ and the *right panel* displays domains of blocks $B_1 = hasLoad(C, L), box(L), B_2 = hasLoad(C, L), box(L), tri(L)$ and $B_3 = hasLoad(C, L), tri(L)$

there is no direct analogue to it in the traditional level-wise approach, as was illustrated in Sect. 5 in the context of reducibility.

Example 8 Let us continue with Example 3, considering a set of two positive examples E^+ and a set of two negative examples E^-.

$$E^+ = \{\{hasCar(c1), hasLoad(c1, l1), circ(l1), box(l1), hasLoad(c1, l2), tri(l2)\},$$
$$\{hasCar(c2), hasLoad(c2, l3), box(l3), tri(l3)\}\}$$
$$E^- = \{\{hasCar(c3), hasLoad(c3, l4), box(l4), circ(l4)\},$$
$$\{hasCar(c4), hasLoad(c4, l5), tri(l5), circ(l5)\}\}$$

We will construct the E^+-non-redundant τ-features by block grafting. First, we generate and filter the following three pos τ-blocks: $box(L)$, $tri(L)$, $circ(L)$. We may check that $circ(L)$ is E^+-redundant (due to $box(L)$), so we may safely discard it. The next pos τ-blocks created by composing $box(L)$ and $tri(L)$ with $hasLoad(C', L')$ are pos τ-blocks $B_1 = hasLoad(C, L) \wedge box(L)$, $B_2 = hasLoad(C, L) \wedge box(L) \wedge tri(L)$ and $B_3 = hasLoad(C, L) \wedge tri(L)$. Notice that if we had not removed $circ(L)$, there would have been seven such pos τ-blocks.

We can now filter also B_1, B_2, B_3 in exactly the same manner as we filtered $box(L)$, $tri(L)$, $circ(L)$. In this case, B_2 is E^+-redundant because

$$\mathrm{dom}_{I_1}(B_2) = \emptyset \subseteq \mathrm{dom}_{I_1}(B_1) \cap \mathrm{dom}_{I_1}(B_3) = \{c1\},$$
$$\mathrm{dom}_{I_2}(B_2) = \{c2\} \subseteq \mathrm{dom}_{I_2}(B_1) \cap \mathrm{dom}_{I_2}(B_3) = \{c2\},$$
$$\mathrm{dom}_{I_3}(B_1) \cap \mathrm{dom}_{I_3}(B_3) = \emptyset \subseteq \mathrm{dom}_{I_3}(B_2) = \emptyset,$$
$$\mathrm{dom}_{I_4}(B_1) \cap \mathrm{dom}_{I_4}(B_3) = \emptyset \subseteq \mathrm{dom}_{I_4}(B_2) = \emptyset.$$

Finally, we may compose these pos τ-blocks with $car(C')$ to obtain the resulting set of neutral features.

In principle, it could happen that by removing some redundant pos τ-blocks from $\mathcal{F}$, another redundant pos τ-block could become irredundant. While it indeed happens for non-strictly redundant blocks (if we have only two blocks with equal domains and we remove one of them, the second one will become irredundant), it does not happen if we filter the non-strictly redundant blocks before we start the process of discarding the strictly redundant ones. This is expressed more formally by the next theorem.

Theorem 6 *Let $\mathcal{F} = \{B_i\}_{i=1}^n$ be a set of pos τ-blocks with equal types of input arguments such that no two pos τ-blocks in the set $\mathcal{F}$ have equal domains for all examples. Let $\mathcal{F}'$ be a set of pos τ-blocks obtained from $\mathcal{F}$ by repeatedly removing redundant pos τ-blocks. Set $\mathcal{F}'$ is unique.*

9 Feature construction algorithm RELF

In this section, we combine the ideas from the previous sections and present a description of the algorithm RELF (Algorithm 3). The described algorithm constructs features using the graft operator and it filters reducible and redundant features using the rules presented in previous sections. An implementation of this algorithm is publicly available at http://ida.felk.cvut.cz/RELF. The next theorem asserts the completeness of the algorithm. In the subsequent comments, we address some details of the algorithm and also provide the intuition as to why the completeness property holds.

Theorem 7 *Let $\tau = (\gamma, \mu)$ be a template and let $E = E^+ \cup E^-$ be the set of examples. Further, let $\mathcal{F}_\tau$ be the set of all non-H-reducible τ-features and let $\mathcal{F}_{RelF}$ be the set of conjunction of atoms constructed by* RELF. *Then $\mathcal{F}_{RelF} \subseteq \mathcal{F}_\tau$ and for every τ-feature $F_\tau \in \mathcal{F}_\tau$, which is not strictly redundant, there is a feature F_{RelF} such that $\mathrm{ext}_E(F_{RelF}) = \mathrm{ext}_E(F_\tau)$.*

Let $\tau = (\gamma, \mu)$ be a template. Let us tackle a simplified task first and let us construct all non-H-reducible τ-features.The algorithm takes pos blocks and grafts them with atoms and other pos blocks in order to produce bigger blocks and continues with this process

Algorithm 3 RELF: Given a template and a set of examples, RELF computes the propositionalized table

1: **Input:** template τ, examples E;
2: *PosBlocks* $\leftarrow$ {}
3: *OrderedAtoms* $\leftarrow$ topologically ordered (w.r.t. $\prec$ from Def. 1) predicate definitions computed from τ
4: **for** $\forall$*Atom* $\in$ *OrderedAtoms* **do**
5: *NewPosBlocks* $\leftarrow$ Combine(*Atom*, *PosBlocks*, E) // See procedure Combine in Algorithm 4
6: Filter pos τ-blocks with equal domains for all examples from (keep the short ones and if they have equal sizes, select them arbitrarily) *NewPosBlocks*
7: Filter redundant pos τ-blocks from *NewPosBlocks*
8: Add *NewPosBlocks* to *PosBlocks*
9: **end for**
10: Save all correct features from *PosBlocks*

until it produces the whole set of features. We have not explained yet how the order of atoms from τ should be determined. It is obvious that we could always start with the atoms that are declared in τ without output arguments (e.g. $box(+l)$ or $tri(+l)$). Next, we would like to graft these one-atom blocks with other atoms from τ. At any point of the feature construction process, only the atoms with *types* of output arguments equal to the types of positive variables of the already generated blocks are usable. The question is whether we can expect that such atoms always exist. The answer is that if the set of features is non-empty and if it has not been constructed yet then such atoms must always exist, which is a consequence of the partial order $\prec$ on constants (i.e. *types*) given in Definition 1.

Example 9 Consider the template

$$\tau \approx a(-a), a(+b), a(+c), bond(+a,-b,-t), bond(+b,-c,-t), single(+t), double(+t).$$

RELF starts by building blocks corresponding to $a(+b)$, $a(+c)$, $single(+t)$ or $double(+t)$. Assume it has already built blocks corresponding to all these four atoms. Next, the only possibility is to take the atom $bond(+b,-c,-t)$ because if $bond(+a,-b,-t)$ were taken before $bond(+b,-c,-t)$, it would not be possible to construct all features. For example

$$a(A), bond(A,B,T1), bond(B,C,T2), a(C), single(T1), double(T2)$$

could not be constructed as can be easily seen. So it must take $bond(+b,-c,-t)$ and only after that it can take $bond(+a,-b,-t)$. In the end it can build features by grafting the already generated blocks with $a(-a)$.

When we have a procedure capable of constructing the whole set of non-H-reducible features, we will be able to enrich it by detection of redundant blocks. Due to monotonicity of redundancy we will still be able to obtain one feature for each set of irredundant features with equal extensions.

Consider now the step in which blocks are composed to yield bigger blocks. It is quite straightforward to construct the set of all pos blocks with an atom a as root when we already have the set of all *legal* combinations of the pos τ-blocks that were already generated (cf. Algorithm 4). Legal combinations of pos τ-blocks are those combinations where all pos blocks B_i have equal *type* of the variable $p(B_i)$ (if τ is ambiguous and if we denote by $\hat{p}(B_i)$ the set of all possible *types* of the positive variable of B_i then we require the intersection $\bigcap_i \hat{p}(B_i)$ to be non-empty) and for $i \neq j$, we require $B_i \not\preceq_H B_j$ (i.e. we require that no combination gives rise to H-reducible blocks). Finding the legal combinations is a rather straightforward task, which can be also combined with redundancy filtering.

10 Experiments

In this section we evaluate the speed of RELF and the accuracy of classifiers constructed with RELF's features.[4] Our first intention is to compare the runtimes of feature construction conducted by RELF to those achieved by the existing propositionalization system RSD. In Krogel et al. (2003) RSD has been shown to be competitive w.r.t. predictive accuracy with

[4]The version of RELF evaluated in this paper differs from the one evaluated in Kuželka and Železný (2009) in that it puts more effort into constructing short features.

Algorithm 4 Combine: Given a set of already generated pos τ-blocks, an atom $A \in \gamma$ and a set of examples, this procedure generates τ-features or pos τ-blocks with their roots built upon the predicate symbol of A

Input: Atom $A \in \gamma$, Set of pos τ-blocks *PosBlocks*, Set of examples $Examples = E^+ \cup E^-$;
$Generated_0 \leftarrow \{a\}$, where a is an atom built upon the predicate symbol of A
for $i = 1, \ldots,$ arity(A) **do**
  if i-th argument of A is an output **then**
    $Generated_i \leftarrow \{\}$
    /* Procedure *BuildCombinations*(*PosBlocks*, *Type*, *Examples*) constructs the set of all *legal* combinations of pos blocks with the type of positive variables equal to *Type* */
    $Combinations \leftarrow BuildCombinations(PosBlocks, Type, Examples)$, where *Type* is the term appearing as i-th argument of A
    for $\forall B^- \in Generated_{i-1}$ **do**
      Let V be the i-th argument of a
      for $\forall \mathcal{S} \in Combinations$ such that $\exists I \in E^+ : \mathrm{dom}_I(B^- \oplus_V \mathcal{S}) \neq \emptyset$ **do**
        /* $\mathcal{S}$ is a set of pos blocks */
        $F \leftarrow B^- \oplus_V \mathcal{S}$
        $Generated_i \leftarrow Generated_i \cup \{F\}$
      end for
    end for
    Filter E^+-redundant pos τ-blocks from $Generated_i$
  else
    $Generated_i \leftarrow Generated_{i-1}$
  end if
end for
return $Generated_i$

propositionalization system SINUS (Krogel et al. 2003) and RELAGGS (Krogel and Wrobel 2001) in typical ILP tasks such as predicting mutagenicity or learning legal positions of chess-end-games, therefore we chose RSD for comparison with RELF. Our second goal is to assess the classification accuracies obtained with RELF's features in nine relational classification benchmarks. Nine different methods for classification combined from propositionalization and learning algorithms (RELF, RSD, Support Vector Machines, Logistic Regression, One Rule, kFoil, nFoil and a RELF's variation entitled NRELF) are tested in this study. Lastly, we aim at tracing the effects of reducibility and redundancy based pruning on RELF's performance. The experimental material is available at http://ida.felk.cvut.cz/RELF.

In the experiments described in this section we use l_2-regularized logistic regression from LIBLINEAR package (Fan et al. 2008) and support vector machines (Vapnik 1995) with RBF kernel from WEKA (Witten and Frank 2005) for all datasets except for the NCI dataset where we use linear SVM from LIBLINEAR package which is a faster alternative to the RBF kernel SVM. We follow suggestions given in Scheffer and Herbrich (1997) to obtain an unbiased estimate of quality of learned classifiers. All accuracies presented in this section are estimates obtained by 10-fold cross-validation except the accuracies for the NCI 786 dataset for which a train-test split was used. For each fold, the parameters (maximum depth of features for RELF, maximum feature length for RSD and the cost parameter for

Table 2 Properties of Mutagenesis dataset, Predictive Toxicology Challenge dataset (P-FM, P-MM, P-FR, P-MR), CAD dataset (CAD), Mesh dataset (Mesh), Carcinogenesis + PTC + Mutagenesis dataset (CPM) and NCI 786 dataset (NCI 786)

Dataset	Classes	Examples	Facts	Acc. of majority class
Mutagenesis	2	188	21024	66.5%
CAD	2	96	16674	57.2%
Mesh	13	278	2387	26.4%
CPM	2	830	67309	52.9%
P-FM	2	349	27762	59.0%
P-MM	2	336	25482	61.6%
P-FR	2	351	27762	65.5%
P-MR	2	344	26986	55.8%
NCI 786	2	3506	254380	52.3%

SVM and logistic regression) are optimized by 3-fold cross-validation on the remaining nine training folds. We perform experiments both with RSD having the same feature declaration bias as RELF and with RSD allowing cyclic features.

10.1 Datasets

We performed experiments with four molecular datasets and two datasets related to engineering problems listed in Table 2. The first experiment was done with the regression-friendly part of the well-known Mutagenesis dataset (Lodhi and Muggleton 2005), which consists of 188 organic molecules marked according to their mutagenicity. The next set of experiments was done with data from the Predictive Toxicology Challenge (Helma et al. 2001) which consists of more than three hundreds of organic molecules marked according to their carcinogenicity on male and female mice and rats. We also performed experiments with a dataset that we created by merging the Carcinogenesis dataset (Srinivasan et al. 1997), the PTC dataset and the Mutagenesis dataset. The task in this slightly artificial learning problem was to distinguish the generally *toxic* compounds from the remaining *control* compounds. The largest of the four molecular datasets that we use in this paper is the NCI 786 (Swamidass et al. 2005) dataset which contains 3506 molecules labeled according to their ability to inhibit growth of renal tumors. In all of these four experiments, we used only atom-bond descriptions and only the crude resolution with atom types like *carbon*, *hydrogen* and not the finer resolution with atom types like *aliphatic hydrogen* or *aromatic hydrogen*.

Next, we performed experiments in a domain describing CAD documents (product structures) (Žáková et al. 2007). The CAD dataset is interesting in that relatively long features are needed to obtain reasonable classification accuracy. An example of a feature from this dataset is

$$F = cadFile(A) \wedge hasCADEntity(A, B) \wedge hasBody(B, C) \wedge hasFeature(C, D)$$
$$\wedge\, hasSecondLimit(D, E) \wedge limitType(E, offset) \wedge next(D, F) \wedge \cdots 18 \text{ more atoms.}$$

Finally, we performed experiments with the well-known Mesh dataset (Dolsak and Muggleton 1992). The task in this problem is to predict, for each edge of a geometrical model, the best number of finite elements that should be placed on the edge in order to introduce small approximation errors to finite element modeling while keeping the complexity of the model as low as possible. The examples in the Mesh dataset are not in the form of isolated interpretations, but they form several interconnected *networks* corresponding to particular finite

element models. The current implementation of RELF copes with this problem by reading the *networks* and converting them to single interpretations according to a given template τ. The conversion process accepts a *network* $\mathcal{N}$ together with classification of the nodes and a template $\tau = (\gamma, \mu)$ where the declared predicates with no input arguments are supposed to refer to nodes in the network. An interpretation corresponding to a node $n \in \mathcal{N}$ is then $I_n = \bigcup_{A,\theta} A\theta$ where A ranges over all connected subsets of γ which contain a declared predicate with no input arguments and θ ranges over all substitutions such that $A\theta \subseteq \mathcal{N}$. The interpretations can be computed efficiently e.g. using breadth-first-search, essentially resembling the way bottom clauses are computed in Progol (Muggleton 1995).

10.2 Feature construction

Here we present feature construction runtimes of RELF and RSD on the six datasets. Since RELF does not restrict size of features, we performed experiments using templates with varying complexity (depth). For RSD we used templates corresponding to the most complex templates used by RELF but we limited their maximum size. Figure 6 displays runtimes of RELF and RSD. For the molecular datasets, we report the number of *bond*-atoms in the longest features since this is a more intuitive measure of their complexity than the number of logical atoms. For the remaining two datasets, we report numbers of logical atoms in the longest features.

In all of the experiments, RELF was able to construct significantly larger features than RSD. The longest features discovered by RELF for the molecular datasets contained more than twenty *bond* atoms. This contrasts with results obtained by RSD, which was able to find features with at most four *bond* atoms. Only, in the experiment with the NCI 786 dataset using the most complex template, we also needed to set minimum frequency of RELF's features to 1%. A consequence of this was that even if we had not restricted RSD's bias to tree-like features, RSD did not find any cyclic features (only, it took longer to generate the tree-like ones). For the remaining two datasets, CAD and Mesh, RELF was again able to construct far larger features than RSD but in the case of the CAD data, RSD was also able to construct cyclic features. However, the predictive accuracy for the cyclic features was lower than for the acyclic case. We also measured memory consumption of RELF. Since our current implementation of RELF exploits properties of Java garbage collection mechanism, maximum memory consumption measured with a loose memory limit can be higher than memory consumption measured with some lower memory limit. Therefore we used RELF with maximum memory set to 128 MB, 256 MB etc. and we report the minimum memory that enabled RELF to finish its execution. RELF needed 256 MB for all four PTC datasets and for Mutagenesis. It needed 768 MB for the CPM dataset. On the CAD dataset, RELF was able to run with maximum memory set to just 64 MB and with 128 MB on the Mesh dataset. For the NCI 786 dataset, RELF needed 4 GB of memory on a 64-bit machine.

We also performed several experiments in order to assess the effect of the pruning methods implemented in RELF on its performance. First, we disabled filtering of redundant features. A result of this was that RELF was only able to construct features corresponding to the templates with lowest complexity (cf. Fig. 6) on molecular datasets. With turned off filtering of redundant features, RELF was unable to construct features on the CAD dataset. This clearly illustrates that filtering of redundant features is an essential part of RELF. To evaluate the effect of filtering of H-reducible blocks, we turned off explicit tests for H-subsumption in the phase where combinations are built. This resulted in about 40% slow-down on CAD dataset and negligible slow-downs on the rest of the datasets. When judging the impact of H-subsumption filtering integrated in RelF, it is important to appreciate that this method

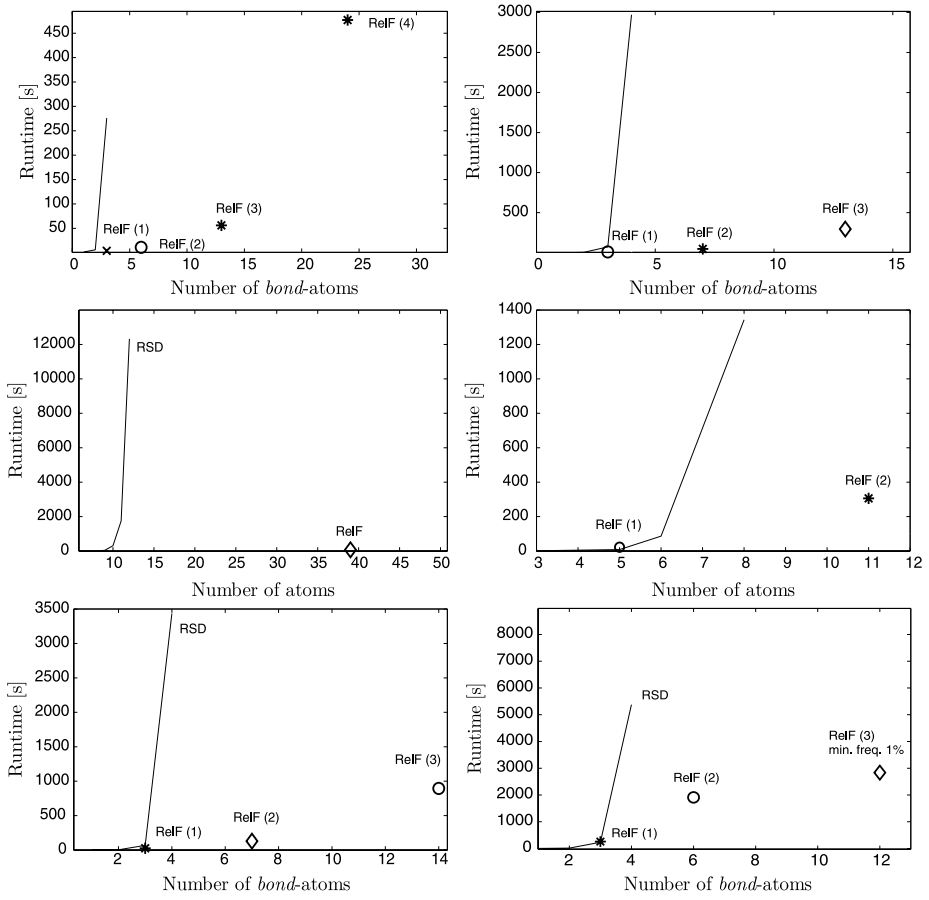

Fig. 6 Runtimes of RELF and RSD on Mutagenesis (*top left*), PTC-MR (*top right*), CAD (*middle left*) Mesh (*middle right*), CPM (*bottom left*) and NCI 786 (*bottom right*). The syntactical bias of RSD allows to specify the maximum allowed length of features. RELF only allows to specify a template and, implicitly, also the maximum depth of the features. Therefore RSD is used with template corresponding to the template of RELF with the greatest complexity and limited maximum length and RELF is used with several templates with increasing complexity

serves not only to perform these explicit tests but also to make RELF able to consider only combinations of blocks without repetitions.

The systems kFOIL and nFOIL do not resort to exhaustive search, instead, they are based on beam search. This means that by setting beam size and maximum size of searched clauses sufficiently low, these algorithms can find reasonably accurate classifiers in a short time. In order to test kFOIL and nFOIL, we set these parameters so that their running times would be in the same orders as the running time of RELF and then compare the obtained predictive accuracies. nFOIL was used with maximum clause length set to 20 for all datasets and with beam size set to 100 for Mutagenesis, 30 for CAD and PTC, 10 for Mesh and NCI 786 dataset and 20 for CPM. Parameter settings for kFOIL were as follows. Beam size was set to 100 for CAD, to 200 for Mutagenesis, to 10 for PTC and CPM and to 7 for NCI 786. Maximum clause length was set to 30 for Mutagenesis and CAD datasets, to 5 for P-MR P-FM, P-MM, P-FR, CPM and NCI 786. It is interesting to note that kFOIL's runtime varied

significantly even for very similar datasets. For example, while it took kFOIL just several minutes to finish on the P-FM dataset, it needed tens of hours to finish on the P-MR dataset.

10.3 Classification

Here we present classification accuracies obtained using RELF's or RSD's output in conjunction with propositional learners and accuracies obtained by kFOIL and nFOIL. Table 3 displays predictive 10-fold cross-validated accuracies obtained using one best rule, support vector machines and l_2-regularized logistic regression acting on features constructed by RELF and RSD. An exception was the NCI 786 dataset for which accuracy was assessed using a train-test split. RELF obtained highest accuracy on six out of nine datasets. RELF in conjunction with SVM or logistic regression also obtained higher predictive accuracy than nFOIL and kFOIL.

The higher accuracies achieved by RELF could have been attributed to the fact that RELF creates classifiers using several thousands of features whereas kFOIL and nFOIL try to build classifiers using only a small set of informative features. Thus, in the experiments with SVM and logistic regression, we were essentially trading interpretability for predictive accuracy. In order to assess whether RELF's ability to construct large features is beneficial also for the task of finding interpretable classifiers we performed an additional experiment. We followed the approach introduced in SAYU (Davis et al. 2005) with features generated by RELF. We started with an empty set of features and we greedily added features that improved accuracy of a naive Bayes classifier.[5] This method, marked as NRELF in Table 4, obtained higher predictive accuracy than kFOIL and nFOIL on seven out of nine datasets which shows that RELF's ability to construct large features is beneficial also for the task of finding small interpretable classifiers.

Table 3 Accuracies of RELF and RSD on Mutagenesis dataset (Muta), Predictive Toxicology Challenge datasets (P-FM, P-MM, P-FR, P-MR), CAD dataset (CAD), Mesh dataset (Mesh), Carcinogenesis + PTC + Mutagenesis dataset (CPM) and NCI 786 dataset (N 786))

	RelF			RSD		
	SVM	LR	OneR	SVM	LR	OneR
Muta	**87.4 ± 6.9**	81.1 ± 8.9	70.3 ± 4.8	79.9 ± 4.6	82.6 ± 6.8	72.3 ± 4.8
CAD	**96.8 ± 5.2**	**96.8 ± 5.2**	85.5 ± 8.6	94.9 ± 7.2	95.1 ± 9.7	81.4 ± 9.3
Mesh	**62.6 ± 6.9**	60.7 ± 10.3	40.9 ± 5.4	61.4 ± 7.3	58.9 ± 6.2	40.7 ± 4.2
CPM	**73.6 ± 4.5**	71.2 ± 4.5	56.9 ± 5.2	71.3 ± 3.7	68.7 ± 3.9	56.5 ± 1.7
P-FM	62.2 ± 5.7	57.9 ± 6.3	64.4 ± 6.0	59.3 ± 6.9	58.7 ± 5.3	**64.5 ± 3.6**
P-MM	59.9 ± 8.3	63.4 ± 10.5	**64.9 ± 5.8**	62.2 ± 4.9	59.1 ± 6.8	61.3 ± 9.6
P-FR	66.4 ± 3.0	65.5 ± 4.5	68.6 ± 3.4	65.3 ± 6.0	67.3 ± 6.0	**68.7 ± 4.1**
P-MR	62.0 ± 6.9	66.0 ± 5.6	57.8 ± 5.3	65.1 ± 6.1	**69.2 ± 6.0**	56.2 ± 5.8
N 786	68.3	**69.6**	56.0	69.3	69.3	55.2

[5]This was implemented using WEKA (Witten and Frank 2005) using its *AttributeSelectedClassifier* with feature selection method *ClassifierSubsetEval* and with search procedure *GreedyStepwise*. Since this turned out to be very time-consuming, we used only the features from the NCI 786 dataset that had frequency higher than 1%.

Table 4 Accuracies of RELF, kFOIL and nFOIL on Mutagenesis dataset (Muta), Predictive Toxicology Challenge datasets (P-FM, P-MM, P-FR, P-MR), CAD dataset (CAD), Mesh dataset (Mesh), Carcinogenesis + PTC + Mutagenesis dataset (CPM) and NCI 786 dataset (N 786). The numbers in parentheses denote average number of features in the respective classifiers

	NRELF	nFOIL	kFOIL
Muta	**80.6** ± **9.8** (9.7)	76.6 ± 9.6 (4.6)	76.0 ± 5.6 (7.1)
CAD	**94.1** ± **6.7** (3)	92.7 ± 6.9 (3.6)	89.8 ± 10.6 (2)
Mesh	**59.4** ± **4.2** (16.2)	57.9 ± 11.6 (18.5)	n.a.
CPM	**65.1** ± **2.8** (65.3)	58.3 ± 5.8 (9)	62.2 ± 4.7 (16.9)
P-FM	**61.5** ± **9.3** (22.5)	60.2 ± 8.7 (5)	57.3 ± 5.6 (3.1)
P-MM	61.0 ± 6.9 (26.6)	**63.1** ± **8.8** (6.8)	62.2 ± 7.4 (4.6)
P-FR	**70.7** ± **5.6** (21.9)	67.0 ± 6.5 (9.1)	63.4 ± 8.3 (6.7)
P-MR	53.9 ± 10.0 (28.6)	57.3 ± 7.1 (6.7)	**59.3** ± **7.0** (7.2)
N 786	**64.4** (41)	63.7 (16)	63.1 (5)

11 Related work

The primary aim of the algorithm RELF presented in this paper is to enable exhaustive generation of large numbers of features in a limited subset of first order logic. Several pattern mining algorithms have been proposed for this task or for more or less similar tasks. A well-known algorithm working in the logical setting is the frequent pattern miner WARMR (Dehaspe and Toivonen 1999), which greatly exploits monotonicity of frequency to prune uninteresting patterns. Monotonicity of frequency can be easily exploited also in our algorithm by discarding infrequent pos blocks.[6] If a minimum frequency threshold is set in RELF, however, one has to refine the statement about RELF's completeness as only the irredundant features with frequency above the specified threshold are guaranteed to be found. Another algorithm for pattern discovery is the RSD algorithm (Železný and Lavrač 2006). RSD does not prune patterns using the minimum frequency constraint as WARMR does, instead, RSD relies on its expressive user-definable syntactical bias, which is also somewhat similar to WARMR's warmode declarations. RSD's language bias is also used by RELF which puts further requirements on syntax of correct features ensuring their tree-like structure. A common principle of RSD and WARMR is the level-wise approach to feature construction, which means that features are built by adding one logical atom at time. Unlike the block-wise approach presented in this paper, the level-wise approach to feature construction cannot be easily combined with filtering of reducible and redundant features, which we have already discussed in the respective sections about redundancy and relevancy.

A recent line of research, represented by algorithms nFOIL (Landwehr et al. 2007), kFOIL (Landwehr et al. 2006) and SAYU (Davis et al. 2005), tries to refrain from explicitly constructing the set of all *interesting* features (frequent, non-redundant etc.) by constructing only features that improve classification accuracy or some related scoring criterion when combined with a propositional learner such as Naive Bayes or SVM. Both nFOIL and kFOIL used in experiments in this paper are based on FOIL's (Quinlan 1990) search, although, in principle, any strategy for hypothesis search could be used instead of FOIL. A potential replacement for FOIL could be e.g. Progol (Muggleton 1995) or even some procedure searching over features generated by a propositionalization algorithm. An advantage of systems

[6] It is possible to set minimum frequency in the current implementation of RELF.

like nFOIL and kFOIL is that they can produce relatively accurate models within reasonably short runtimes. On the other hand, in our experiments, giving these algorithms more time did not increase their predictive accuracy sufficiently to enable them to outperform classifiers constructed from the whole set of RELF's features.

Frequent graph mining algorithms are also related to RELF. They are very well suited for molecular databases, however, they have limitations in other domains. For example, the currently fastest graph mining algorithm Gaston (Nijssen and Kok 2005) is able to mine frequent molecular structures from larger databases than RELF, but it cannot easily handle oriented graphs or even hypergraphs (Wörlein et al. 2005). The covering relation adopted by graph mining algorithms, which corresponds to subgraph isomorphism, differs significantly from the one used by RELF and other related algorithms such as nFOIL or kFOIL, whose covering relation corresponds to hypergraph homomorphism. As a consequence, number of frequent patterns in the respective frameworks may differ significantly as, for example, some patterns are expressible only with homomorphism. Conversely, there are tree patterns that can be represented with the graph mining approach but cannot be represented by tree-like features with their covering relation. Another notable system geared towards discovery of patterns in molecular databases is MolFea (Kramer and De Raedt 2001), which restricts the patterns to linear molecular fragments.

Recently, there has been a growing interest in removing various forms of *redundancy* from frequent pattern sets. The form of *redundancy* exploited by RELF was introduced in Lavrač et al. (1999) where it was defined for propositional data and for constrained Horn clauses, which are equivalent to propositional representation in their expressive power. RELF extends this framework to tree-like features by establishing monotonicity of *redundancy* w.r.t. grafting. There are also other forms of *redundancy* considered in literature. For example, a system called Krimp was introduced in Koopman and Siebes (2009), Van Leeuwen et al. (2006) which searches for a representative set of features that would allow for a lossless compression of a given set of examples, i.e. it relies on the minimum description length principle. Mining of closed frequent patterns is another approach that minimizes the number of discovered patterns by considering only representative candidates from certain equivalence classes. Finally, in Bringmann et al. (2006), it has been shown that simple patterns (e.g. linear fragments) work well when employed in classification of chemical compounds, which is directly related to our work as we also use only tree-like patterns.

12 Conclusions

We have introduced RELF, an algorithm for construction of acyclic relational features. We have shown that blockwise construction of acyclic features enables RELF to remove H-reducible and redundant pos features. In experiments, we have shown that RELF is able to construct *relevant* features with sizes far beyond the reach of state-of-the-art propositionalization systems. Of importance, we have also shown that, for nine relational learning benchmarks, restriction to acyclic features was not detrimental with respect to predictive accuracy. In fact, the results obtained with acyclic features were, in most cases, higher than predictive accuracies of state-of-the-art systems nFOIL and kFOIL. Importantly, the features constructed by the other tested algorithms were often also acyclic. Although there are definitely datasets where cyclic features could provide better predictive accuracies, one can always merge results from different propositionalization algorithms and feed the propositional learners with such merged propositionalized tables. An algorithm for construction of a limited class of features such as RELF would be useful also in such cases.

Acknowledgements The authors are supported by the Czech Science Foundation through project 201/08/0509. F.Ž. is further supported by the Czech Ministry of Education through project MSM6840770038. The authors would like to thank Stefan Kramer for advising on the correspondences between conjunctions and hypergraphs, and the anonymous reviewers of ICML'09, ILP'09 and the Machine Learning journal for helpful remarks, especially on the connections between features and acyclic database queries.

Appendix: Theorems and proofs

For sakes of the following proofs we first need to define a few concepts we did not need in the main text. The notation used in this section is summarized in Table 5.

Definition 12 (pos-neg τ_θ-block) Let B be a τ_θ-conjunction. If there is exactly one τ_θ-positive variable ($p(B)$) and exactly one τ_θ-negative variable ($n(B)$), we say that B is a pos-neg τ_θ-block.

Definition 13 (Descendant, Induced pos τ-block) Let F be a $(\gamma,\mu)_\theta$-feature or a pos τ-block and let $a \in F$ be an atom. We say that an atom $b \in F$ is a *descendant* of a (denoted by $a \prec_d b$) if there is a sequence of atoms $a_1, \ldots, a_n$ such that $a_1 = a$, $a_n = b$ and every two consecutive atoms a_i, a_{i+1} share a variable which has an output in $a_i\theta$ and an input in $a_{i+1}\theta$. Then $Induced_F(a) = \{a\} \cup \{b | a \prec_d b\}$ is called a pos τ-block *induced* in F by a.

Definition 14 (Parent, Child) Let F be a $(\gamma,\mu)_\theta$-feature or a pos τ-block. If two atoms $a, b \in F$ share a variable V, which has an output in $a\theta$ and an input in $b\theta$, we call *a parent of b* (denoted by $P_F(a)$) and *b child of a* (denoted by $C_F(a)$).

Lemma 1 *Let F' be the result of the reduction from Definition 4 applied on a τ-feature F. Then the number of atoms in F' is the same as the number of variables in F'. Futhermore, each atom in F' contains an output and exactly one input.*

Proof Denote A (V) the number of atoms (variables) in F'. Every variable in F' must have at least two occurrences (otherwise it would have been removed by rule 2) and one of them must be input as each variable has at most one output in F (due to the neutrality requirement) and thus also in F'. Each atom in F, and thus in F', contains at most one input due to condition (i) in Definition 1, so $A \geq V$. Furthermore, each atom in F' contains more than one term (otherwise it would have been removed by rule 1) and, as we have seen already, contains at most one input. So each atom in F' contains an output. Again realize there is at most one output per variable in F', therefore $V \geq A$. We have seen that $A \geq V$, so indeed $A = V$ as the lemma says. We have also seen that each atom contains an output which was to be shown. Lastly, we have seen that each of the $A = V$ atoms contains at most

Table 5 Summary of notation introduced in Definitions 13, 14

Notation	Meaning	Note
$b \in C_F(a)$	*b is a child of a*	a and b are atoms
$b = P_F(a)$	*b is a parent of a*	a and b are atoms
$a \prec_d b$	*a is a descendant of b*	a and b are atoms
$B = Induced_F(a)$	*B is an induced pos block of a in F*	a is an atom, B, F are blocks/features

one input and each of the $A = V$ variables must have one input in F' so each atom has exactly one input. □

Theorem 1 *Every τ-feature F is tree-like.*

Proof Since F is a τ-feature, $\tau = (\gamma, \mu)$, there must be a substitution θ such that $F\theta \subseteq \gamma$. By Definition 1 there is a partial order $\prec$ on $\{v_i\theta \,|v_i \in vars(F)\}$ such that $v_j\theta \prec v_k\theta$ whenever v_j (v_k) has an input (output) in an atom $a \in F$. Let F' be the result of the reduction from Definition 4 applied on F. Denote $\prec'$ the suborder of $\prec$ on $vars(F') \subseteq vars(F)$. Since $\prec'$ is also a partial order, we can sort variables in F' topologically as $v_1, v_2, \ldots, v_n$ ($n = |vars(F')|$) such that

$$v_j\theta \prec' v_k\theta \quad \Rightarrow \quad j < k \tag{4}$$

According to Lemma 1 there are exactly n atoms in F' each containing exactly one input. Assume for contradition that F is not tree-like and thus $n > 0$. We sort atoms in F' by their single inputs, i.e. atom a_i has v_i as input. Atom a_n that has v_n as input must, by Lemma 1, have another variable v_m ($1 \leq m \leq n$) as output. Since v_n (v_m) has an input (output) in atom a_n, it must be that $v_n \prec' v_m$. This however contradicts implication 4 above as $m \leq n$. Therefore F is tree-like. □

Lemma 2 *Let A, B be $(\gamma, \mu)_\theta$-features or pos (γ, μ)-blocks, let ϑ be an H-substitution such that $A\vartheta \subseteq B$ (i.e. $A \preceq_H B$). If $b \notin A\vartheta$ and $b' \in Induced_B(b)$, then $b' \notin A\vartheta$.*

Proof Since an application of a substitution on A preserves its connectedness, there are two possible cases: (i) $A\vartheta \subseteq B \setminus Induced_B(b)$ or (ii) $A\vartheta \subseteq Induced_B(b) \setminus \{b\}$. The latter case is not possible because in the definition of H-subsumption, the substitutions were required to respect depth of variables and every variable contained in $Induced_B(b) \setminus \{b\}$ has depth w.r.t. B greater than 1. Therefore the only possibility is $A\vartheta \subseteq B \setminus Induced_B(b)$. □

Lemma 3 *Let A, B be $(\gamma, \mu)_\theta$-features or pos (γ, μ)-blocks and let ϑ be an H-substitution such that $A\vartheta \subseteq B$. Then if for some $a \in A$, $b \in B$ it holds $a\vartheta = b$ then $Induced_A(a)\vartheta \subseteq Induced_B(b)$.*

Proof Suppose for contradiction that there is some $a' \in Induced_A(a)$ such that $a'\vartheta \notin Induced_B(b)$. There is then some longest sequence of atoms $a_1, \ldots, a, \ldots, a'$ ($a_i \in A$) such that every two consecutive atoms share a variable, which is an output in the first one and an input in the second one. It follows from definition of depth and from definition of H-subsumption that ϑ maps atoms from this sequence to atoms from sequence $b_1, \ldots, b, \ldots, b'$, where every two consecutive atoms share a variable, which is an input in the first one and an output in the second one, and b' is an atom in the same depth as a'. One such sequence must obviously exist. If there were more such sequences then B would not be tree-like. This is a contradiction with the assumption that $a'\vartheta \notin Induced_B(b)$ because $b' \in Induced_B(b)$. □

Lemma 4 *Let F be a τ_θ-feature or a pos τ-block. If there is an H-substitution ϑ such that $F\vartheta \subseteq F$ and there are atoms $a, b \in F$ and their respective induced pos τ-blocks $A, B \subseteq F$ such that $A\vartheta \subseteq B$ then either $\mathrm{p}(A) = \mathrm{p}(B)$ or there are pos τ blocks $A', B' \subseteq F$ such that $A \subsetneq A'$ and $B \subsetneq B'$ and $A'\vartheta \subseteq B'$.*

Proof If $A\vartheta \subseteq B$ then $\mathcal{P}(b) \in F\vartheta$ by application of Lemma 2, from which $\mathcal{P}(a)\vartheta = \mathcal{P}(b)$. If $\mathrm{p}(\mathcal{P}(a)) = \mathrm{p}(\mathcal{P}(b))$, we are done. Otherwise, since $F\vartheta \subseteq F$ we have $Induced_F(\mathcal{P}(a))\vartheta \subseteq F$ therefore $Induced_F(\mathcal{P}(a))\vartheta \subseteq Induced_F(\mathcal{P}(b))$ by application of Lemma 3 because $\mathcal{P}(a)\vartheta = \mathcal{P}(b)$. □

Theorem 2 *Let B be a pos τ-block and let B^- be a neg τ-block. Then the following holds:* (i) *B is H-reducible if and only if B contains pos τ-blocks B_1, B_2 such that $B_1 \neq B_2$, $\mathrm{p}(B_1) = \mathrm{p}(B_2)$ and $B_1 \preceq_H B_2$.* (ii) *If B is H-reducible, then $B^- \oplus (S \cup \{B\})$ is also H-reducible for any set of pos blocks S.*

Proof (i $\Rightarrow$) Let B_r be H-reduction of B and let θ_1, θ_2 be H-substitutions such that $B_r\theta_1 \subseteq B$ and $B\theta_2 \subseteq B_r$. Substitution $\theta_3 = \theta_2\theta_1$ is a mapping $\theta_3 : vars(B) \to vars(B)$. Since $B\theta_2 \subseteq B_r$, $|B\theta_2| \leq |B_r|$ and consequently $|B\theta_3| \leq |B_r| < |B|$, because applying a substitution to a τ-conjunction cannot increase its size. Therefore there is an atom $a \in B \setminus B\theta_3$ and, by Lemma 2, also a whole pos τ-block $B_1 \subseteq B \setminus B\theta_3$. Thus, there is a pos τ-block B_2 ($B_2 \neq B_1$) such that $B_1\theta_3 \subseteq B_2$. It remains to show that for some such B_1, B_2, $\mathrm{p}(B_1) = \mathrm{p}(B_2)$. If $\mathrm{p}(B_1) \neq \mathrm{p}(B_2)$, then there must be pos τ-blocks G_1, G_2 such that $B_1 \subsetneq G_1$, $B_2 \subsetneq G_2$ and $G_1\theta_3 \subseteq G_2$ (by Lemma 4). For such G_1, G_2 with maximum size, $\mathrm{p}(G_1) = \mathrm{p}(G_2)$. (i $\Leftarrow$) Let θ be a H-substitution such that $B \setminus B\theta = B_1$, then $B\theta \approx_H B$ and $|B\theta| = |B| - |B_1| < |B|$. (ii) This follows directly from (i). □

Lemma 5 *Let F be a (γ, μ)-feature. Then $d_F \leq |\gamma|$.*

Proof Assume $d_F > |\gamma|$. Let θ be a substitution such that $F\theta \subseteq \gamma$ and $a_1, \ldots, a_{d_F}$ be a sequence of F's atoms such that for $1 \leq i < d_F$, a_i contains variable v_i as output and a_{i+1} contains v_i as input. By transitivity of the partial order $\prec$ assumed by Definition 1, it must hold $v_i\theta \prec v_j\theta$ for $1 \leq i < j \leq d_F$. Since $d_F > |\gamma|$, the sequence must contain two atoms a_k and a_l, such that $k < l$, $a_k\theta = a_l\theta$ and thus $v_k\theta = v_l\theta$. Since $k < l$, $v_k\theta \prec v_l\theta$. But given that $v_k\theta = v_l\theta$, this contradicts the assumption of irreflexivity of the order $\prec$. □

Theorem 3 *Let $\tau = (\gamma, \mu)$ be a template. Then there is only a finite number of τ-features, which are not H-reducible.*

Proof First, we show that for any template, the set P_d of pos blocks with depth at most d is finite. (i) For $d = 1$, there is at most $|\gamma|$ pos τ-blocks because any pos τ-block with depth 1 is just a single atom. Therefore P_1 is finite. (ii) Let us suppose that we have proved the theorem for maximum depth at most d. We need to show that then it holds also for $d + 1$. We can take all atoms $a \in \gamma$ and for each of them create the set of τ-conjunctions

$$P^a_{d+1} = \{a\theta_a \wedge_{i=1}^{arity(a)} C_i\theta_i | C_i \in 2^{P_d}\} \cup P_d,$$

where θ_a is only meant as *variabilization* of a (i.e. as substitution that replaces constants by suitable variables) and θ_i maps $\mathrm{p}(S)$ of every pos τ-block $S \in C_i$ to i-th output argument of $a\theta_a$. Not all elements of sets P^a_{d+1} are correct pos τ-blocks, but it is easy to check that all pos τ-blocks with depth at most $d + 1$ are contained in $\bigcup_{a\in\gamma} P^a_{d+1}$. To see this, it is important to notice that it suffices to create combinations of pos τ-blocks without repetition since any combination with repetitions would necessarily lead to H-reducible pos τ-blocks (Theorem 2). Now, validity of the theorem follows from the fact that $P_d = \bigcup_{a\in\gamma} P^a_d$ is finite for any d and that for any template there is a maximum depth of τ-features (Theorem 5). □

Theorem 4 *Algorithm 1 correctly decides whether a τ_θ-feature (pos τ-block) F is H-reducible in time $O(|F|^2)$.*

Proof First, we show, using an induction argument, that (a_i, a_j) gets to the list *Open* if and only if $Induced_F(a_i) \preceq_H Induced_F(a_j)$. (i) This is trivial for pos features with depth 1. (ii) We suppose that the claim holds for depth d. The induction argument implies that $Subsumed(P_F(a_i^{d+1}), P_F(a_j^{d+1})) = |C_F(l_i^{d+1})|$ if and only if every child c of a_i^{d+1} can be mapped by H-substitution to some child c' of a_j^{d+1} while preserving $I_F(c) = I_F(c')$. Validity of the induction step then follows from this together with $predicate(a_i^{d+1}) = predicate(a_j^{d+1})$. Now, from Theorem 2 we know that existence of two pos τ-blocks B_1, B_2 with $p(B_1) = p(B_2)$ contained in a feature F such that $B_1 \preceq_H B_2$ is equivalent to H-reducibility of F, which finishes the proof of correctness of the algorithm.

Clearly, each pair of atoms can be added only once to the list *Open* because it happens only once that $Subsumed(P_F(a_i), P_F(a_j)) = |C_F(a_i)|$ (because $Subsumed(P_F(a_i), P_F(a_j))$ is only incremented and $|C_F(a_i)|$ is a constant). It follows that the *repeat-until*-loop will run for at most $|F|^2$ steps. Since all the other operations can be performed in $O(1)$ time (in the unit-cost RAM model), we have the bound $O(|F|^2)$. □

Lemma 6 *Let I be an interpretation, τ a template and let $\mathcal{S}_1 = \{B_1, \ldots, B_m\}$ and $\mathcal{S}_2 = \{C_1, \ldots, C_n\}$ be standardized-apart sets of pos τ-blocks such that $\bigcap_{i=1}^m \mathrm{dom}_I(B_i) \subseteq \bigcap_{i=1}^n \mathrm{dom}_I(C_i)$, then for any pos-neg τ_θ-block or neg τ_θ-block B^- $\mathrm{dom}_I(B^- \oplus \mathcal{S}_1) \subseteq \mathrm{dom}_I(B^- \oplus \mathcal{S}_2)$*

Proof Let us first consider the case when $d_{B^-}(\mathrm{n}(B^-)) = 1$. The only place, where domains of $B_i \in \mathcal{S}_1$ ($C_i \in \mathcal{S}_2$ respectively) are used, is line 5 in Algorithm 2. Clearly $argDom_{\mathcal{S}_1} = \bigcap_{i=1}^m \mathrm{dom}_I(B_i) \subseteq argDom_{\mathcal{S}_2} = \bigcap_{i=1}^n \mathrm{dom}_I(C_i)$ and consequently also $atomsDom_{\mathcal{S}_1} \subseteq atomsDom_{\mathcal{S}_2}$ and therefore also $\mathrm{dom}_I(B^- \oplus \mathcal{S}_1) \subseteq \mathrm{dom}_I(B^- \oplus \mathcal{S}_2)$. The general case of lemma may be proved by induction on depth of $\mathrm{n}(F^-)$. (i) The case for depth 1 has been already proved. (ii) Let us suppose that the lemma holds for depth d. Now, we may take the pos-neg τ_θ-block $T \subseteq B^-$ which contains $\mathrm{n}(B^-)$ such that $(B^- \setminus T) \oplus \{T\} = B^-$ and $d_T(\mathrm{n}(B^-)) = 1$, and *graft it* with $\mathcal{S}_1$ ($\mathcal{S}_2$, respectively). We have $\mathrm{dom}_I(T \oplus \mathcal{S}_1) \subseteq \mathrm{dom}_I(T \oplus \mathcal{S}_2)$ and by induction argument finally also $\mathrm{dom}_I(B^- \oplus \mathcal{S}_1) = \mathrm{dom}_I((B^- \setminus T) \oplus \{T \oplus \mathcal{S}_1\}) \subseteq \mathrm{dom}_I((B^- \setminus T) \oplus \{T \oplus \mathcal{S}_2\}) = \mathrm{dom}_I(B^- \oplus \mathcal{S}_2)$. □

Theorem 5 *Let E^+ (E^-) be a set of positive (negative) examples. Let $\mathcal{F}$ be a set of pos τ-blocks with equal types of input arguments and let $\mathcal{B} = \{B_i\}_{i=1}^n \subseteq \mathcal{F}$. Let $\mathrm{dom}_I(B_1) \subseteq \bigcap_{i=2}^n \mathrm{dom}_I(B_i)$ for all $I \in E^+$ ($I \in E^-$) and let $\bigcap_{i=2}^n \mathrm{dom}_I(B_i) \subseteq \mathrm{dom}_I(B_1)$ be true for all $I \in E^-$ ($I \in E^+$). Then for any neg τ-block B^-: $B^- \oplus (\{B_1\} \cup \mathcal{S})$ is E^+-redundant (E^--redundant), where $\mathcal{S} \subseteq \mathcal{F}$.*

Proof We will prove only the case for E^+-redundancy because the proof for E^--redundancy is analogous. Let us first ignore the set $\mathcal{S}$. Let B^- be a neg τ-block. By application of Lemma 6, if $\mathrm{dom}_I(B_1) \subseteq \bigcap_{i=2}^n \mathrm{dom}_I(B_i)$ for all $I \in E^+$, then $\mathrm{dom}_I(F^- \oplus \{B_1\}) \subseteq \mathrm{dom}_I(F^- \oplus \{B_2, \ldots, B_n\})$ for all $I \in E^+$. Similarly, if $\bigcap_{i=2}^n \mathrm{dom}_I(B_i) \subseteq \mathrm{dom}_I(B_1)$ for all $I \in E^-$, then $\mathrm{dom}_I(B^- \oplus \{B_2, \ldots, B_n\}) \subseteq \mathrm{dom}_I(B^- \oplus \{B_1\})$ for all $I \in E^-$. Therefore if $I \models F^- \oplus \{B_1\}$, then $I \models F^- \oplus \{B_2, \ldots, B_n\}$ for all $I \in E^+$ and similarly if $I \models B^- \oplus \{B_2, \ldots, B_n\}$, then $I \models B^- \oplus \{B_1\}$ for all $I \in E^-$. This means that $B^- \oplus \{B_1\}$ must be E^+-redundant. Now, we consider also the set $\mathcal{S} = \{H_1, \ldots, H_o\}$. Notice that

$\mathrm{dom}_I(B_1) \bigcap_{i=1}^{o} \mathrm{dom}_I(H_i) \subseteq \bigcap_{i=2}^{n} \mathrm{dom}_I(B_i) \bigcap_{i=1}^{o} \mathrm{dom}_I(H_i)$ must hold for all positive examples and $\bigcap_{i=2}^{n} \mathrm{dom}_I(B_i) \bigcap_{i=1}^{o} \mathrm{dom}_I(H_i) \subseteq \mathrm{dom}_I(B_1) \bigcap_{i=1}^{o} \mathrm{dom}_I(H_i)$ must hold for all negative examples. The rest of the proof is obvious; it is an analogy of the argument used to prove the first part of the theorem. □

Theorem 6 *Let $\mathcal{F} = \{B_i\}_{i=1}^{n}$ be a set of pos τ-blocks with equal types of input arguments such that no two pos τ-blocks in the set $\mathcal{F}$ have equal domains for all examples. Let $\mathcal{F}'$ be a set of pos τ-blocks obtained from $\mathcal{F}$ by repeatedly removing redundant pos τ-blocks. Set $\mathcal{F}'$ is unique.*

Proof Let $\prec_R$ be a relation defined as follows: $B \prec_R C$ if and only if $\mathrm{dom}_I(B) \subseteq \bigcap_{k \in A} \mathrm{dom}_I(B_k) \cap \mathrm{dom}_I(C)$ for all $I \in E^+$ and $\bigcap_{k \in A} \mathrm{dom}_I(B) \cap \mathrm{dom}_I(C) \subseteq \mathrm{dom}_I(B)$ for all $I \in E^-$ and $B \neq C$ and $\forall i : B \neq B_i$. It is not hard to check that $\prec_R$ is a partial order (also due to the fact that no two elements of $\mathcal{F}$ have identical domains). Set $\mathcal{F}'$ contains all maximum elements w.r.t. $\prec_R$ and is therefore unique. □

Theorem 7 *Let $\tau = (\gamma, \mu)$ be a template and let $E = E^+ \cup E^-$ be the set of examples. Further, let $\mathcal{F}_\tau$ be the set of all non-H-reducible τ-features and let $\mathcal{F}_{RelF}$ be the set of conjunction of atoms constructed by* RELF. *Then $\mathcal{F}_{RelF} \subseteq \mathcal{F}_\tau$ and for every τ-feature $F_\tau \in \mathcal{F}_\tau$, which is not strictly redundant, there is a feature F_{RelF} such that $\mathrm{ext}_E(F_{RelF}) = \mathrm{ext}_E(F_\tau)$.*

Proof (Sketch) It is not hard to see that $\mathcal{F}_{RelF} \subseteq \mathcal{F}_\tau$. For each conjunction of atoms $F \in \mathcal{F}_{RelF}$, we just construct a substitution θ such that θ maps each atom $a \in F$ to an atom from γ that was used when a was created in line 2 in Algorithm 4. Then it is not hard to show that such a substitution is consistent (i.e. there are no two conflicting substitutions of one variable) and that $F\theta \subseteq \gamma$ and that also the conditions on input and output arguments stated in Definition 3 are satisfied.

In order to justify the theorem, we need to show that if RELF did not remove redundant pos τ-blocks or pos τ-blocks that do not cover any positive example, it would construct all τ-features contained in $\mathcal{F}_\tau$. The case with filtering of redundant pos τ-blocks will follow from Theorems 2 and 5. First, recall that from earlier theorems we know that all features are tree-like and $|\mathcal{F}_\tau|$ is finite. For contradiction, let us assume that there is a substitution θ and a non-H-reducible τ_θ-feature $F \in \mathcal{F}_\tau$ such that $F \notin \mathcal{F}_{RelF}$. Let B be a shortest pos τ_θ-block such that $F = B^- \oplus \{B\}$ and such that B was not constructed by RELF (i.e. the shortest block which caused that F could not be constructed). If no such block exists, let us set $B = F$. Further, let $a \in \gamma$ be an atom that corresponds to B's root. By a brief analysis of the algorithm RELF, we may conclude that if all pos τ_θ-blocks, which can be composed with a using the graft operator, had been generated, then procedure *Combine*(*Atom*, *PosBlocks*, E) would have also generated B on line 5 of Algorithm 4. Since the necessary blocks must be generated at some point in the feature construction process (because otherwise we would have a contradiction with the minimality of B), the only possibility is that they were generated only after the atom a had been processed. This possibility is, however, ruled out by the requirement on the partial irreflexive order on constants in γ given in Definition 1 (recall that $F\theta \subseteq \gamma$) and by the fact that RELF sorts the atoms from γ according to this order before it starts the feature construction process. Thus we see that B and consequently also F had to be constructed by RELF, which finishes the proof. □

References

Blockeel, H., Dehaspe, L., Demoen, B., Janssens, G., Ramon, J., & Vandecasteele, H. (2002). Improving the efficiency of inductive logic programming through the use of query packs. *The Journal of Artificial Intelligence Research*, *16*(1), 135–166.

Bringmann, B., Zimmermann, A., Raedt, L. D., & Nijssen, S. (2006). Don't be afraid of simpler patterns. In *PKDD '06: 10th European conference on principles and practice of knowledge discovery in databases* (pp. 55–66). Berlin: Springer.

Davis, J., Burnside, E., Page, D., Dutra, I., & Costa, V. S. (2005). Learning Bayesian networks of rules with SAYU. In *Proceedings of the 4th international workshop on Multi-relational mining*. New York: ACM.

Dechter, R. (2003). *Constraint processing*. San Mateo: Morgan Kaufmann.

Dehaspe, L., & Toivonen, H. (1999). Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, *3*(1), 7–36.

Dolsak, B., & Muggleton, S. (1992). The application of inductive logic programming to finite element mesh design. In *Inductive logic programming* (pp. 453–472). San Diego: Academic Press.

Fagin, R. (1983). Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the Association for Computing Machinery*, *30*(3), 514–550.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., & Lin, C.-J. (2008). Liblinear: a library for large linear classification. *Journal of Machine Learning Research*, *9*, 1871–1874.

Helma, C., King, R. D., Kramer, S., & Srinivasan, A. (2001). The predictive toxicology challenge 2000–2001. *Bioinformatics*, *17*(1), 107–108.

Koopman, A., & Siebes, A. (2009). Characteristic relational patterns. In *KDD '09: proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 437–446). New York: ACM.

Kramer, S., & De Raedt, L. (2001). Feature construction with version spaces for biochemical applications. In *ICML '01: proceedings of the eighteenth international conference on machine learning* (pp. 258–265). San Mateo: Morgan Kaufmann.

Krogel, M. A., & Wrobel, S. (2001). Transformation-based learning using multirelational aggregation. In *ILP '01: proceedings of the 11th international conference on inductive logic programming* (pp. 142–155). Berlin: Springer.

Krogel, M.-A., Rawles, S., Železný, F., Flach, P. A., Lavrač, N., & Wrobel, S. (2003). Comparative evaluation of approaches to propositionalization. In *International conference on inductive logic programming (ILP 03')*. Berlin: Springer.

Kuželka, O., & Železný, F. (2009). Block-wise construction of acyclic relational features with monotone irreducibility and relevancy properties. In *ICML 2009: the 26th int. conf. on machine learning*.

Landwehr, N., Passerini, A., De Raedt, L., & Frasconi, P. (2006). kFOIL: learning simple relational kernels. In *AAAI'06: proceedings of the 21st national conference on artificial intelligence* (pp. 389–394). Menlo Park: AAAI Press.

Landwehr, N., Kersting, K., & De Raedt, L. (2007). Integrating naïve bayes and FOIL. *Journal of Machine Learning Research*, *8*, 481–507.

Lavrač, N., & Flach, P. A. (2001). An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic*, *2*(4), 458–494.

Lavrač, N., Gamberger, D., & Jovanoski, V. (1999). A study of relevance for learning in deductive databases. *Journal of Logic Programming*, *40*(2/3), 215–249.

Lodhi, H., & Muggleton, S. (2005). Is mutagenesis still challenging. In *International conference on inductive logic programming (ILP '05), late-breaking papers* (pp. 35–40).

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing, Special Issue on Inductive Logic Programming*, *13*(3–4), 245–286.

Nienhuys-Cheng, S.-H., & de Wolf, R. (1997). *Foundations of inductive logic programming*. New York: Springer.

Nijssen, S., & Kok, J. N. (2005). The Gaston tool for frequent subgraph mining. *Electronic Notes in Theoretical Computer Science*, *127*(1), 77–87.

Perkins, S., & Theiler, J. (2003). Online feature selection using grafting. In *ICML* (pp. 592–599). Menlo Park: AAAI Press.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*(3), 239–266.

Scheffer, T., & Herbrich, R. (1997). Unbiased assessment of learning algorithms. In *15th international joint conference on artificial intelligence (IJCAI '97)* (pp. 798–803).

Srinivasan, A., King, R. D., Muggleton, S., & Sternberg, M. J. E. (1997). Carcinogenesis predictions using ILP. In *ILP '97: proceedings of the 7th international workshop on inductive logic programming* (pp. 273–287). Berlin: Springer.

Swamidass, S. J., Chen, J., Bruand, J., Phung, P., Ralaivola, L., & Baldi, P. (2005). Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. *Bioinformatics*, *21*(1), 359–368.

Van Leeuwen, M., Vreeken, J., & Siebes, A. (2006). Compression picks item sets that matter. In *PKDD '06: 10th European conference on principles and practice of knowledge discovery in databases* (pp. 585–592). Berlin: Springer.

Vapnik, V. N. (1995). *The nature of statistical learning theory*. Berlin: Springer.

Železný, F., & Lavrač, N. (2006). Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, *62*, 33–63.

Witten, I. H., & Frank, E. (2005). *Data mining: practical machine learning tools and techniques* (2nd ed.). San Francisco: Morgan Kaufmann.

Wörlein, M., Meinl, T., Fischer, I., & Philippsen, M. (2005). A quantitative comparison of the subgraph miners MoFa, gSpan, FFSM, and Gaston. In *LNCS. PKDD 2005, 9th European conference on principles and practice of knowledge discovery in databases* (pp. 392–403). Berlin: Springer.

Yannakakis, M. (1981). Algorithms for acyclic database schemes. In *International conference on very large data bases (VLDB '81)* (pp. 82–94).

Žáková, M., Železný, F., Garcia-Sedano, J., Tissot, C. M., Lavrač, N., Křemen, P., & Molina, J. (2007). Relational data mining applied to virtual engineering of product designs. In *International conference on inductive logic programming (ILP '07)*. Berlin: Springer.