Tractable Construction of Relational Features

Filip Železný

České vysoké učení v Praze
Fakulta elektrotechnická
Katedra kybernetiky
zelezny@fel.cvut.cz

Abstract. A popular technique for converting multi-relational data into a single-relational form is based on constructing truth-valued relational features of the data instances, where the features play the role of binary attributes in the resulting representation. Here I consider a simple relational feature language whose formulas correspond to conjunctions of first-order atoms where arguments are only variables and respect user-defined constraints on types and input/output modes. I show a sufficient condition for polynomial time construction of such formulas and give preliminary results on tractable enumeration of complete sets of such formulas.

1 Introduction

This paper is concerned with efficient construction of expressions such as

```
hasCar(C),hasLoad(C,L1),triangle(L1),hasLoad(C,L2),box(L2)
```

corresponding to conjunctions of constant-free, function-free, non-negated atoms and subject to some predefined syntactic constraints. The expression is an example of a *feature* related to an *individual* (here a train, refer to Fig. 1) meaning that the train has a car carrying a triangle-shaped load and a box-shaped load. Note, so far informally, that all variables appearing as *outputs*, eg. `C` in `hasCar(C)`, also appear as *inputs*, eg. `C` in `hasLoad(C,L1)` and vice versa. This will be a general requirement on correct features.

I am not concerned here with the semantics of features and do not treat the problem of determining their truth value for a specific individual. Note however, that for purposes of generating a single-relational representation of a database, a set of features is evaluated with respect to a given individual at a time, and this procedure is then conducted for all available individuals. Due to this item-wise process of feature evaluation, no special, 'key variable' is needed which would link the feature to some individual in the database. Below I will discuss more on the consequences of ignoring the key variable on the syntactic construction of features.

Let me define an atom and an expression (features will be searched among expressions) formally. I assume there are some infinite countable sets S ('predicate symbols') and V ('symbols of variables'). N stands for the set of naturals numbers.

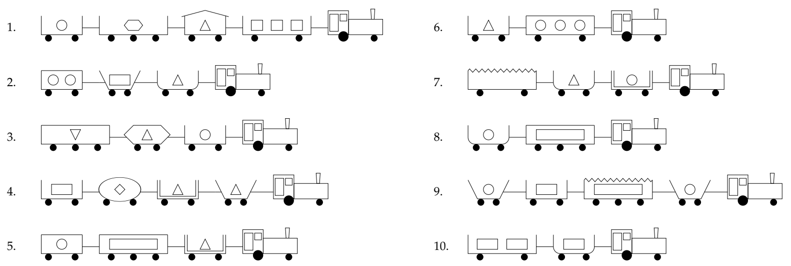

Fig. 1. A data base containing structured items.

Definition 1. *$x = s_x(v_{x,1}, v_{x,2}, \ldots, v_{x,a_x})$ is called an* atom *if $s_x \in S$, $a_x \in N$ and $v_{x,i} \in V$ $(1 \leq i \leq a_x)$. $\mathbf{L}$ denotes the set of all atoms. Any finite $e \subseteq \mathbf{L}$ is called an* expression*. Denote $Arg(e) = \{(x, i) \mid x \in e, 1 \leq i \leq a_x\}$ and $Var(e) = \{v_a \mid a \in Arg(e)\}$. $\mathbf{E}$ denotes the set of all expressions.*

From the semantic point of view, the atom order is irrelevant. This allows to view expressions simply as *sets* of atoms. Note that the indexation in $v_{x,i}$ and a_x should be understood as a functional notation, ie. $v_{l,j}$ (v_{l,a_l}) will always represent the variable at the j-th (last, respectively) argument in atom l (a_l is called the *arity* of l).

The concept of *substitution* also acquires a simple meaning in this constrained framework.

Definition 2. *Let $e \in \mathbf{E}$. A* substitution *is a mapping $\theta : V \to V$. For $x \in e$, $x\theta = s_x(\theta(v_{x,1}), \theta(v_{x,2}), \ldots, \theta(v_{x,a_x}))$ and $e\theta = \{x\theta \mid x \in e\}$. Expression e θ-*subsumes *expression f (denoted as $e \preceq_\theta f$) iff there is a substitution θ such that $e\theta \subseteq f$. Finally, e is* equivalent *to f (written $e \approx f$) iff θ is bijective and $e\theta = f$.*

A specific feature language is defined by a *template*, which is made of

– an expression t, determining the argument types and modes of all atoms which may appear in a feature,
– a subset M of argument places in the above expression, which are assigned the input-mode.

The role of typing will simply be that no variable can appear in a feature at two argument places with different types. The reason I use the expression t for specifying types is that any type-compliant expression will then θ-subsume t, as I will exemplify later.

The M subset will simply list the argument places which have to hold an input variable. Although the input-output moding of arguments has a clear intuitive role (refer to the initial train example), my only formal requirement will be that each variable present in a feature appears at both some input and some output argument place. This requirement leads to closed-form, well-interpretable features [3].

The following definition formalizes the above. Further it defines when a variable is *proper*, ie. when it has both an input and an output role in an expression, and defines which expression is a feature.

Definition 3. *A* template τ *is a pair* (t_τ, M_τ) *where* t_τ *is an expression and* $M_\tau \subseteq Arg(t_\tau)$. $\mathbf{T}$ *denotes the set of all templates. Let* $e \preceq_\theta t_\tau$. *Denote* $Arg^+_\tau(e) = \{(x,i) \in Arg(e) \mid (x\theta, i) \in M_\tau\}$ *and* $Arg^-_\tau(e) = Arg(e) \setminus Arg^+_\tau(e)$. *If some* $v \in V$ *satisfies the equivalence* $(v = v_{a^+}, a^+ \in Arg^+_\tau(e)) \Leftrightarrow (v = v_{a^-}, a^- \in Arg^-_\tau(e))$, *then* v *is said to be* τ-proper *in* e . *A non-empty expression* f *is a* τ-feature *iff* $f \preceq_\theta t_\tau$ *and each* $v \in Var(f)$ *is* τ*-proper in* f.

Example 1. Consider a template (t_τ, M_τ) suitable for the initial train example

$$t_\tau = \mathtt{hasCar(C)}, \mathtt{hasLoad(C,L)}, \mathtt{triangle(L)}, \mathtt{box(L)}$$
$$M_\tau = \{(\mathtt{hasLoad(C,L)}, 1), (\mathtt{triangle(L)}, 1), (\mathtt{box(L)}, 1)\}$$

The typing constraint is here defined by the t_τ expression, isolated from the *moding* constraint. This makes it explicit that verifying compliance to a typing constraint corresponds to a subsumption check. Indeed, consider an expression

$$e = \mathtt{hasCar(X)}, \mathtt{hasLoad(X,Y)}, \mathtt{triangle(Y)}, \mathtt{hasLoad(X,Z)}$$

e complies to the typing specified by t_τ because $e \preceq_\theta t_\tau$.[1]

Each variable of a feature must occur at both an argument contained in M_τ and an argument not in M_τ. Therefore, e above is not a feature, since `Z` is at no argument in M_τ.

Note that every template τ has a dual template τ^{-1} with inverse moding $M^{-1}_\tau = Arg(t_\tau) \setminus M_\tau$ and every τ-feature is also a τ^{-1}-feature. Through the inversion, every 'primary structural' (such as `hasCar(C)`) becomes a 'property' and every property (such as `triangle(L)`) becomes a primary structural. A theoretical consequence is that if a feature class (say features where atoms chained by variable sharing form a 'tree') can be efficiently enumerated, then the inverse class (here atoms forming a 'root') can also be efficiently enumerated. The symmetric properties commented above are a result of disregarding the key (individual-linking) variable, which would occur only at some 'input' arguments (those in M_τ). In a sense, the sole role of the key type lies in setting the *orientation* of the otherwise symmetric features, whereas the orientation is irrelevant for sakes of feature construction.

Note that also the notation (t_τ, M_τ) above is understood functionally, ie. for any template $\rho \in \mathbf{T}$, t_ρ represents the prescribed typing of ρ and M_ρ its moding. Let me now specify the main problem treated in the remainder of this paper.

[1] In general, there may be more than one substitution θ such that $e \preceq_\theta t_\tau$. A straightforward way to avoid this ambiguity is to constraint oneself, quite naturally, to templates where t_τ contains each symbol $s \in S$ (such as `hasCar`) at most once.

Definition 4. *Let $T \subseteq \mathbf{T}$ and $E : T \to 2^{\mathbf{E}}$. The* feature existence problem *for T and E is defined as follows. The problem instance is the tuple $n \in N$, $\tau \in T$. The instance size is n. The instance solution is a τ-feature f such that $|f| \leq n$ and $f \approx f'$ for some $f' \in E(\tau)$ (if such f exists), or "NO" otherwise.*

The function E takes a template $\tau \in T$ and produces a set of expressions in which τ-features are searched (due to the $f \approx f' \in E(\tau)$ requirement, a solution may be not be in $E(\tau)$ but must be equivalent to some expression in $E(\tau)$; this avoids dependency on variable naming). The reason for specifying the problem class this way is that different complexity results can be proved for different functions E's. For example, $E(\tau)$ may consist of *connected* expressions (where all atoms are linked via the transitive closure of the variable sharing relation) and then be independent of τ. Less trivially, $E(\tau)$ may be such that all τ-features therein are *loop-free* (edges between atoms given by variable sharing, orientation given by moding) and thus obviously depend on the moding of the specific τ.

2 The Bottom Set Theorem

I will first show that the problem of constructing an expression out of a finite set of available atoms, such that a given variable is proper (has both an input and an output occurrence) in that expression, is equivalent to a problem of satisfying a set of propositional Horn clauses.

Lemma 1. *Let τ be a template, $e = \{l_1, l_2, \ldots, l_p\}$ and $e' \subseteq e$ two expressions, $P = \{P_1, P_2, \ldots P_p\}$ a set of propositional variables and $v \in Var(e)$. Further let*

$$\{in_1, in_2, \ldots, in_r\} = \{1 \leq in \leq p \mid \exists i \; (l_{in}, i) \in Arg^{+}_{\tau}(e), v_{l_{in},i} = v\} \tag{1}$$

$$\{out_1, out_2, \ldots, out_s\} = \{1 \leq out \leq p \mid \exists i \; (l_{out}, i) \in Arg^{-}_{\tau}(e), v_{l_{out},i} = v\} \tag{2}$$

be two index sets.[2] *Let further $C_{in}(v)$ denote the following set of Horn clauses*

$$P_{in_1} \vee \neg P_{out_1} \vee \neg P_{out_2} \vee \ldots \vee \neg P_{out_s} \tag{3}$$
$$P_{in_2} \vee \neg P_{out_1} \vee \neg P_{out_2} \vee \ldots \vee \neg P_{out_s} \tag{4}$$
$$\vdots \tag{5}$$
$$P_{in_r} \vee \neg P_{out_1} \vee \neg P_{out_2} \vee \ldots \vee \neg P_{out_s} \tag{6}$$

Similarly, let $C_{out}(v)$ be the following Horn clause set

$$P_{out_1} \vee \neg P_{in_1} \vee \neg P_{in_2} \vee \ldots \vee \neg P_{in_r} \tag{7}$$
$$P_{out_2} \vee \neg P_{in_1} \vee \neg P_{in_2} \vee \ldots \vee \neg P_{in_r} \tag{8}$$
$$\vdots \tag{9}$$
$$P_{out_s} \vee \neg P_{in_1} \vee \neg P_{in_2} \vee \ldots \vee \neg P_{in_r} \tag{10}$$

[2] The former index set thus addresses those of atoms in e, which contain v at some input argument while the latter set corresponds to atoms with v acting as an output.

Let $C(v) = C_{in}(v) \cup C_{out}(v)$ and $\xi_{e'} : P \rightarrow \{true, false\}$ be a truth assignment

$$\xi_{e'}(P_i) = \begin{cases} false, \text{ if } l_i \in e'; \\ true, \quad \text{if } l_i \notin e'. \end{cases} \tag{11}$$

Then v is τ-proper in e' iff $\xi_{e'}$ satisfies all clauses in $C(v)$.

Proof. (Sufficiency) Let $\xi_{e'}$ be an arbitrary truth assignment to the variables $P_1, P_2, \ldots, P_p$, such that $\xi_{e'}$ satisfies all clauses in $C(v)$. Let further $u_1, u_2, \ldots u_\mu$ $(1 \leq \mu \leq p)$ be the indexes of those of the variables which are assigned the *false* value by $\xi_{e'}$, ie. $e' = \{l_{u_1}, l_{u_2}, \ldots, l_{u_\mu}\}$. I need to show that $v = v_{a^+}, a^+ \in Arg^+_\tau(e')$ (v appears at an input argument) iff $v = v_{a^-}, a^- \in Arg^-_\tau(e')$ (v appears at an output argument). Let me first show the implication

v appears at an input argument $\Rightarrow$ v appears at an output argument

As the implication assumes, one of the atoms $l_{in_1}, l_{in_2}, \ldots l_{in_r}$ containing the input occurrences of v (see Eq. 1) must be present in e', ie. there must be a κ $(1 \leq \kappa \leq r)$ such that $in_\kappa = u_k$. Then P_{in_κ} is assigned the *false* value by $\xi_{e'}$. Because all clauses of $C_{in}(v)$ must be satisfied by $\xi_{e'}$, so must be its κ-th clause

$$P_{in_\kappa} \vee \neg P_{out_1} \vee \neg P_{out_2} \vee \ldots \vee \neg P_{out_s} \tag{12}$$

Since P_{in_κ} is *false*, at least one of $P_{out_1}, P_{out_2}, \ldots P_{out_s}$ must hold the *false* value to keep the clause satisfied; let it be P_{out_λ} $(1 \leq \lambda \leq s)$. If P_{out_λ} is *false* then $l_{out_\lambda} \in e'$. But l_{out_λ} is a atom containing (see Eq. 2) an output occurrence of v. Thus I have proved the above implication. The inverse implication can be shown analogically, using the fact that all clauses in $C_{out}(v)$ must be satisfied.

(Necessity) Let e' $(e' \subseteq e)$ be an arbitrary expression in which v appears as both an input and an output (there are $a^+ \in Arg^+_\tau(e')$ and $a^- \in Arg^-_\tau(e')$ such that $v = v_{a^+} = v_{a^-}$). Hence at least one of the atoms $l_{out_1}, l_{out_1}, \ldots, l_{out_s}$ containing an output occurrence of v (see Eq. 2) must be present in e'; let it be l_{out_κ} $(1 \leq \kappa \leq s)$. Then $\xi_{e'}$ assigns the *false* value to P_{out_κ}. Since all clauses in $C_{in}(v)$ contain $\neg P_{out_\kappa}$, all clauses in $C_{in}(v)$ are satisfied. At the same time, e' must also contain at least one of the atoms $l_{in_1}, l_{in_1}, \ldots, l_{in_r}$ where v is at an input argument (see Eq. 1). Let it be l_{in_λ} $(1 \leq \lambda \leq r)$. Then P_{in_λ} is *false* and all clauses in $C_{out}(v)$ are also satisfied as they all contain the propositional atom $\neg P_{in_\lambda}$. Consequently, all clauses in $C(v) = C_{in}(v) \cup C_{out}(v)$ are satisfied by $\xi_{e'}$. □

It is now straightforward to extend the previous lemma to the problem of finding a non-empty expression with all variables proper.

Lemma 2. *Let all assumptions of Lemma 1 hold. Let further $C = \bigcup_{v \in Var(e)} C(v)$ and $C_\emptyset = \{\neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_p\}$ Then the following assertions are equivalent:*

1. *Expression e' is non-empty and each $v \in Var(e)$ is τ-proper in e'.*
2. *Assignment $\xi_{e'}$ satisfies all clauses in the Horn clause set $C \cup C_\emptyset$.*

Proof. ($1 \Rightarrow 2$) If e' is non-empty then some $l_i \in e'$, which means that P_i is assigned the *false* value by $\xi_{e'}$. Thus $C_\emptyset$ is clearly satisfied. Since each $v \in Var(e)$ is τ-proper in e', every clause in each $C(v)$ ($v \in Var(e)$) is satisfied due to Lemma 1. Therefore all clauses in $C \cup C_\emptyset = \bigcup_{v \in Var(e)} C(v) \cup C_\emptyset$ are satisfied.

($2 \Rightarrow 1$) Since $C_\emptyset$ must be satisfied, there is some i ($1 \leq i \leq p$) such that P_i is false in the assignment $\xi_{e'}$. Then $l_i \in e'$, ie. e' is non-empty. As all clauses in each $C(v)$ ($v \in Var(e)$) are satisfied, all $v \in Var(e)$ are τ-proper in e' due to Lemma 1. □

Lemma 3. *Let all assumptions of Lemma 2 hold. A maximal assignment (ie. one assigning the* false *value to the smallest number of variables) satisfying all clauses in C can be found (or decided that no satisfying assignment exists) in time polynomial in r and s.*

Proof. P-completeness of satisfying a set of propositional Horn clauses (the 'HORNSAT' problem) is shown in [5]. Finding a maximal (or minimal) assignment in polynomial time is shown in [2]. □

I am finally in the position to show the main result of this paper, which is informally as follows. If a polynomial set $\perp$ can be constructed such that all acceptable features (up to variable renaming) are subsets thereof (let me call $\perp$ a *bottom set*), one can find a feature (if it exists) in polynomial time. This is despite the fact that there is of course an exponential number of subsets of the bottom set.

Theorem 1. *Let $T \subseteq \mathbf{T}$, $E : T \to 2^{\mathbf{E}}$ and let there be $\perp : T \times N \to \mathbf{E}$ such that for all $\tau \in T$, $n \in N$: $\perp(\tau, n)$ is computable in time polynomial in n, $\perp(\tau, n) \preceq_\theta t_\tau$, and for all τ-features f it holds that $f \approx f' \in E(\tau)$ iff $f \subseteq \perp(\tau, |f|)$. Then the feature existence problem for T and E can be solved in polynomial time.*

Proof. Given the bottom set $\perp(\tau, n)$, the feature existence problem is equivalent to deciding if there is a subset f of the bottom set such that $|f| \leq n$ and f is a τ-feature. $\perp(\tau, n) \preceq_\theta t_\tau$ implies[3] $f \preceq_\theta t_\tau$ for arbitrary $f \subseteq \perp(\tau, n)$. Also, as $\perp(\tau, n)$ is finite, so is every subset thereof. Considering Def. 3, it thus remains to decide whether there exists a non-empty $f \subseteq \perp(\tau, n)$, $|f| \leq n$ where every variable is τ-proper, that is, it appears both at some input argument (ie. $v = v_{a^+}$, $a^+ \in Arg_\tau^+(f)$) and some output argument (ie. $v = v_{a^-}$, $a^- \in Arg_\tau^-(f)$). For brevity denote $e = \perp(\tau, n)$. As Lemma 2 shows, finding a non-empty subset $f \subseteq e$ where each $v \in Var(e)$ (and thus each $v \in Var(f)$) is τ-proper in f, is equivalent to finding a satisfying assignment to a set of propositional Horn clauses $C = \bigcup_{v \in Var(e)} C(v) \cup C_\emptyset$ with propositional variables $P_1, P_2, \ldots, P_{|e|}$. As e is computable in time polynomial in n, $|Arg(e)|$ is at most polynomial in n and so are the numbers r, s in Eq.'s 1 and 2 ($r = Arg_\tau^+(e) \leq |Arg(e)| \geq Arg_\tau^-(e) = s$). Confronting this with the clause set 3-10, each $C(v)$ has a polynomial (in n) number of clauses and atoms. As $|Var(e)| \leq |Arg(e)|$ (due to the existence of the function $v : Arg(e) \to Var(e)$ defined as $v((x, i)) = v_{x,i}$), also C has a

[3] Clearly, if $e' \subseteq e' \preceq_\theta e''$ then $e' \preceq_\theta e''$ for any expressions e, e', e''.

polynomial number of clauses and atoms. Due to Lemma 3, if there is a satisfying assignment to C, one can find in polynomial time a maximal one, which corresponds to the smallest expression $f_{min} \subseteq e$ where each $v \in Var(f_{min})$ is τ-proper in f_{min}. If f_{min} exists and $|f_{min}| \leq n$, the feature existence problem is answered with f_{min}. Otherwise it is answered with "NO". □

Example 2. The bottom set for τ specified in Example 1 and $n = 3$ is

$$\perp(\tau, n) = \texttt{hasCar(C),hasLoad(C,L),triangle(L),box(L)}.$$

Clearly, all τ-features (up to variable renaming) of length up to 3 atoms are subsets of this bottom set (incidently equivalent to t_τ). Let the propositional variables assigned to the bottom atoms (in the order of their appearance) be P_1, P_2, P_3, P_4. The corresponding HORNSAT instance is the union of the following Horn clause sets

$C_{in}(\texttt{C}) = \{P_2 \vee \neg P_1\}$,
$C_{out}(\texttt{C}) = \{P_1 \vee \neg P_2\}$,
$C_{in}(\texttt{L}) = \{P_3 \vee \neg P_2, P_4 \vee \neg P_2\}$,
$C_{out}(\texttt{L}) = \{P_2 \vee \neg P_3 \vee \neg P_4\}$
$C_\emptyset = \{\neg P_1 \vee \neg P_2 \vee \neg P_2 \vee \neg P_4\}$.

One of the maximal solutions assigns the *false* value to P_1, P_2 and P_3 (the reader will check that all mentioned clauses are satisfied), which corresponds to the set `hasCar(C),hasLoad(C,L),triangle(L)`, which therefore forms a correct feature.

3 Bottom Set Existence and Feature Enumeration

The bottom set theorem leaves two crucial questions open:

– When can a polynomial bottom set be constructed?
– When does tractability of the feature existence problem entail tractability of the enumeration of *all* τ-features in $E(\tau)$?

The following results are given without proofs, which will appear in an extended version of the paper.

3.1 Bottom Set Existence

Let me first formalize a few structural notions about templates and features.

Definition 5. *Let $T \subseteq \mathbf{T}$, $E : T \to 2^{\mathbf{E}}$, $\tau \in \mathbf{T}$ and $e \in \mathbf{E}(\tau)$. $x, y \in e$ are said to be* connected *in e iff they share a variable or some $z \in e$ is connected with both x and y. e is is said to be* connected *(or* undecomposable*) iff any atom in e is connected to all other atoms in e. There is a* path *in a τ-feature f from $x \in f$ to $y \in f$ of length 1, iff for some $a, b \in Arg(f)$ it holds $v_a = v_b$, $a \notin M_\tau$, $b \in M_\tau$, or a path of length $l + 1$, iff for some $z \in f$ there is a* *path from x to z of length l and a path from z to y of length* 1. *The* distance $\delta_\tau(x, y)$ *is the length of the shortest path from x to y, if one exists. The* depth *of f is defined as* $\Delta_\tau(f) = \max_{u,v \in f} \delta_\tau(u, v)$. *$f$ is said to be* loop-free *if for no x there is a path in f from x to x. A loop-free τ-feature f is called a* semi-root (semi-tree) *iff no variable has two input (output) occurrences in f w.r.t τ; f is called a* root (tree) *iff it is a semi-root (semi-tree) and no atom in f has two output (input) arguments w.r.t τ. Further, f is called a* semi-chain (chain) *iff it is simultaneously a semi-root (root) and a semi-tree (tree). Finally, τ is said to be hierarchical iff there is a partial irreflexive order $\prec$ on $Var(t_\tau)$ such that $v_{l,i} \prec v_{l,j}$ whenever there are l, i, j such that $(l, i) \in Arg^+_\tau(t_\tau)$ and $(l, j) \in Arg^-_\tau(t_\tau)$.*

The next theorem gives three sufficient conditions, each of which allows for the construction of a polynomial-size bottom set, and therefore also the polynomial-time construction of a feature.

Theorem 2. *Assumptions of Theorem 1 are satisfied if for each $\tau \in T$: every $e \in E(\tau)$ is connected and any of the following holds:*

1. *each τ-feature in $E(\tau)$ is either a tree, root or chain,*
2. *there is a $\Delta_{max} \in N$ such that every τ-feature f in $E(\tau)$ is loop-free and $\Delta_\tau(f) \leq \Delta_{max}$,*
3. *τ is hierarchical (this implies Cond. 2).*

Let me illustrate the spirit of the proof with an example.

Example 3. Consider again the continued train example. For each template τ, let $E(\tau)$ be the set of all *connected* τ-features, ie. expressions such as

$$\texttt{hasCar(C),triangle(C),hasCar(D),triangle(D)}$$

are disallowed. Let further T consist of *hierarchical* templates $\tau = (t_\tau, M_\tau)$, such as

$$t_\tau = \texttt{hasCar(C)}, \texttt{hasRoof(C)}, \texttt{hasLoad(C,L)}, \texttt{triangle(L)}, \texttt{box(L)} \tag{13}$$

$$M_\tau = \{(\texttt{hasLoad(C,L)}, 1), (\texttt{triangle(L)}, 1), (\texttt{box(L)}, 1)\} \tag{14}$$

There is of course no τ-feature of size $n = 1$. For $n = 2$, a bottom set (coinciding with the only correct τ-feature) is

$$\perp(\tau, 2) = \texttt{hasCar(C),hasRoof(C)}$$

For $n = 3$, a bottom set is

$$\perp(\tau, 3) = \texttt{hasCar(C),hasRoof(C),hasLoad(C,L),box(L),triangle(L)}$$

In general, for

$$3 + 3p \leq n < 3 + 3(p + 1), \; p = 1, 2, \ldots \tag{15}$$

$\perp(\tau, n)$ can be obtained by adding the atoms

$$\texttt{hasLoad(C,L}_{p+1}\texttt{),box(L}_{p+1}\texttt{),triangle(L}_{p+1}\texttt{)}$$

to $\perp(\tau, n-1)$. This is better seen from a *directed graph* representation of a bottom set (below), where the nodes correspond to atoms and left-to-right edges correspond one variable being an output in the left node and an input in the right node.[4]

$$\text{branching factor} \leq n \left\{ \underbrace{\begin{array}{lll} \texttt{hasCar(C)} & -\texttt{hasRoof(C)} & \\ & -\texttt{hasLoad(C,L1)} & -\texttt{box(L1)} \\ & & -\texttt{triangle(L2)} \\ & -\texttt{hasLoad(C,L2)} & -\texttt{box(L2)} \\ & & -\texttt{triangle(L2)} \\ \ldots & \ldots & \ldots \end{array}}_{\Delta \leq \Delta_{max}} \right.$$

Here every $\tau-$feature is some connected subgraph of the above graph without orientation. Under the specific choice of τ (Eq. 13), the graph is a tree, which is because all atoms in t_τ have at most one input variable. Due to the assumed hierarchical ordering of types $\prec$, the depth of the graph is bounded by a constant[5] Δ_{max}. Also its branching factor can be upper-bounded by n (eg. no feature of size at most n can address more loads than n). The number of nodes, ie. the size of the bottom set is thus of order $n^{\Delta_{max}}$, that is, polynomial in n. Due to Theorem 2, this guarantees a polynomial-time construction of features.

However, a different choice of τ in Eq. 13 may produce a non-tree graph if atoms with multiple inputs appear in t_τ. Without elaborating the detailed proof, the atoms can still be organized in 'layers' (like above) using the assumed type hierarchy $\prec$. The branching factor is then of order n^{InpAr} where $InpAr$ is the maximum number of inputs in an atom, among atoms in t_τ. In this case $|\perp(\tau, n)|$ is of order $n^{InpAr \times \Delta_{max}}$ which again preserves polynomial-time constructibility of features.

3.2 Enumeration of Feature Sets

The *feature enumeration problem* is the same as the problem of feature existence, except that a solution to the former problem is the set of all distinct (ie. mutually non-equivalent) solutions to the latter problem. An auxiliary, yet important lemma for this problem is as follows.

Lemma 4. *Let the feature existence problem for T and E be decidable in polynomial time and let there be a polynomial number of distinct solutions to every instance thereof. Then the feature enumeration problem for T and E can be decided in polynomial time.*

[4] Formally, there is an edge between nodes of atoms x and y in this representation of $\perp(\tau, n)$, if $v_1 \in Var(x)$ and $v_2 \in Var(y)$ and $\theta(v_1) \prec \theta(v_2)$ where $\prec$ is the assumed hierarchy-defining order, and θ is a substitution mapping the bottom set variables to the variables in t_τ (types) such that $\perp(\tau, n)\theta \subseteq t_\tau$.

[5] assuming implicitly a fixed size of t_τ

Proof. Direct consequence of the enumeration-tractability result in [1]. □

A preliminary analysis indicates that satisfying the conditions stipulated by Theorem 2 implies a polynomial number of solutions to the existence problem and hence the polynomial decidability of the enumeration problem.

Acknowledgement

I am grateful for the support by the Czech Ministry of Education through the grant 1K4108.

References

1. R. Dechter and A. Itai. Finding All Solutions if You can Find One *AAAI-92 Workshop on Tractable Reasoning*, 1992
2. W. F. Dowling and J. H. Gallier. Linear time algorithm for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 3:267-284, 1984.
3. N. Lavrač and P. A. Flach. An extended transformation approach to inductive logic programming *ACM Transactions on Computational Logic* 2:4, 2001
4. N. Lavrač, F. Železný and P. A. Flach. RSD: Relational Subgroup Discovery through First-Order Feature Construction *12th Int. Conf. on Inductive Logic Programming*, Springer 2002
5. Thomas J. Schaefer. The complexity of satisfiability problems. *Tenth Annual Symposium on Theory of Computing*, 1978.