Efficient Operations in Feature Terms using Constraint Programming

Santiago Ontañón and Pedro Meseguer

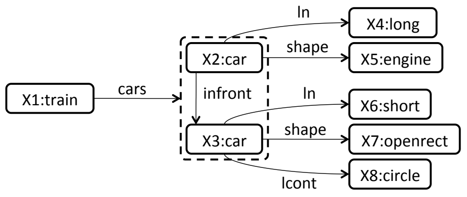
IIIA-CSIC, Artificial Intelligence Research Institute
Spanish Scientific Research Council,
08193 Bellaterra (Spain)
{santi,pedro}@iiia.csic.es

Abstract. Feature Terms are a generalization of first-order terms that have been introduced in theoretical computer science in order to formalize object-oriented capabilities of declarative languages, and which have been recently received increased attention for their usefulness in structured machine learning applications. The main obstacle with feature terms (as well as other formal representation languages like Horn clauses or Description Logics) is that the basic operations like subsumption have a very high computational cost. In this paper we model subsumption as constraint programming (CP), allowing us to solve those operations in a more efficient way than using traditional methods.

1 Introduction

Structured machine learning (SML) [8] focuses on developing machine learning techniques for rich representations such as feature terms [2, 7, 13], Horn clauses [10], or description logics [6]. SML has received an increased amount of interest in the recent years for several reasons, like allowing to handle complex data in a natural way (as illustrated by the success of these techniques in biomedical fields), or sophisticated forms of inference. One of the major difficulties in SML is that basic operations in structured representations like feature terms have a very high computational complexity. This paper focuses on feature terms, and presents a formalization of the subsumption operation based on constraint programming (CP), which allows also for unification and antiunification to be implemented in a more efficient way.

Feature Terms are a generalization of first-order terms that have been introduced in theoretical computer science in order to formalize object-oriented capabilities of declarative languages, and which have been recently received increased attention for their usefulness in structured machine learning applications [12, 3, 13]. The three basic operations among feature terms are subsumption, unification and antiunification, which are essential for defining machine learning algorithms. It is well known that those operations have a high computational cost if we allow set-valued features in feature terms [7] (necessary to represent most structured machine learning datasets).



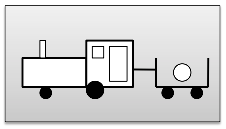


Fig. 1. A simple train represented as a feature term.

Constraint programming has been shown in the past to be a powerful framework that can be used for increasing the performance of relational machine learning algorithms. For example, it is well known that θ-subsumption can be efficiently computed using CP [11]. In this paper, we use constraint programming to model subsumption in set-valued feature terms, and show that this results in an implementation several orders of magnitude faster than standard approaches.

2 Preliminaries

Feature Terms. Feature terms [2, 7] are a generalization of first-order terms, introduced in theoretical computer science to formalize object-oriented declarative languages. Feature terms correspond to a different subset of first-order logics than description logics, although with the same expressive power [1].

Feature terms are defined by its *signature*: $\Sigma = \langle \mathcal{S}, \mathcal{F}, \leq, \mathcal{V} \rangle$. $\mathcal{S}$ is a set of sort symbols, including the most general sort ("any").$\leq$ is an order relation inducing a single inheritance hierarchy in $\mathcal{S}$, where $s \leq s'$ means s is more general than or equal to s', for any $s, s' \in \mathcal{S}$ ("any" is more general than any s which, in turn, is more general than "none"). $\mathcal{F}$ is a set of feature symbols, and $\mathcal{V}$ is a set of variable names. We define a feature term ψ as,

$$\psi ::= X : s \quad [f_1 \doteq \Psi_1, ..., f_n \doteq \Psi_n]$$

where ψ points to the *root* variable X (that we will note as $root(\psi)$) of sort s; $X \in \mathcal{V}$, $s \in \mathcal{S}$, $f_i \in \mathcal{F}$, and Ψ_i might be either another variable $Y \in \mathcal{V}$, or a set of variables $\{X_1, ..., X_m\}$. When Ψ_i is a set $\{X_1, ..., X_m\}$, each element in the set must be different. An example of feature term appears in Figure 1. It is a train (variable X_1) composed of two cars (variables X_2 and X_3). This term has 8 variables, and one set-valued feature (indicated by a doted line): *cars* of X_1.

To make a uniform description, constants are treated as variables of a particular sort. For each X in a term with a constant value k of sort s, we consider that X is a regular variable of a special sort s_k. For each different constant k, we create a new sort s_k of s. Then, we can forget about constants and just treat all variables in the same way.The set of variables of a term ψ is $vars(\psi)$, the set of features of a variable X is $features(X)$, and $sort(X)$ is its sort.

Operations on Feature Terms. The basic operation between feature terms is *subsumption*: whether a term is more general than (or equal to) another one.

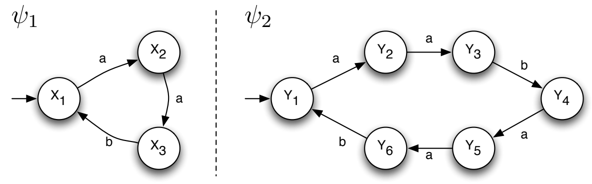


Fig. 2. An example where a bigger feature term subsumes a smaller feature term. In this case $\psi_2 \sqsubseteq \psi_1$ assuming that all variables in them have the same sort s.

Definition 1. *(Subsumption) A feature term ψ_1* subsumes *another one ψ_2 ($\psi_1 \sqsubseteq \psi_2$)* [1] *when there is a total mapping m: $vars(\psi_1) \to vars(\psi_2)$ such that:*

- $root(\psi_2) = m(root(\psi_1))$
- *For each $X \in vars(\psi_1)$*
  - $sort(X) \leq sort(m(X))$,
  - *for each $f \in features(X)$, where $X.f = \Psi_1$ and $m(X).f = \Psi_2$:*
    - $\forall Y \in \Psi_1, \exists Z \in \Psi_2 | m(Y) = Z$,
    - $\forall Y, Z \in \Psi_1, Y \neq Z \Rightarrow m(Y) \neq m(Z)$

    i.e. each variable in Ψ_1 is mapped in Ψ_2, and different variables in Ψ_1 have different mappings.

Subsumption induces a partial order among feature terms, i.e. the pair $\langle \mathcal{L}, \sqsubseteq \rangle$ is a *poset* for a given set of terms $\mathcal{L}$ containing the infimum $\perp$ and the supremum $\top$ with respect to the subsumption order, typically called the *subsumption graph*. We can see the subsumption graph as a directed graph where vertices are feature terms and directed edges indicate subsumption.

Since feature terms can be represented as labelled graphs, it is natural to relate the problem of feature terms subsumption to subgraph isomorphism. However, subsumption cannot be modeled as subgraph isomorphism by two reasons. First, edges and nodes in feature term graphs have labels and sorts, and second, and most important, larger feature terms can subsume smaller feature terms while the corresponding graphs are not isomorphic. See for example the two terms shown in Figure 2, where a term ψ_2 with six variables subsumes a term with three variables ψ_1 (with the mapping $m(Y_1) = m(Y_4) = X_1$, $m(Y_2) = m(Y_5) = X_2$, and $m(Y_3) = m(Y_6) = X_3$).

Given the partial order introduced by subsumption, we define the two other basic operations on feature terms: unification and antiunification. Both are operations over the subsumption graph: antiunification finds the most specific common "parent"; unification finds the most general common "descendant". In feature terms, unification and antiunification might not be unique. For space reasons, in this paper we will only show how subsumption can be modeled

[1] In description logics notation, subsumption is written in the reverse order since it is seen as "set inclusion" of their interpretations. In machine learning, $A \sqsubseteq B$ means that A is more general than B, while in description logics it has the opposite meaning.

using constraint programming. However, both antiunification and unification be accelerated as well thanks to our model.

Constraint Satisfaction. A Constraint Satisfaction Problem (CSP) involves a finite set of variables, each taking a value in a finite discrete domain. Subsets of variables are related by constraints that specify permitted value tuples. Formally,

Definition 2. *A CSP is a tuple (X, D, C), where $X = \{x_1, \ldots, x_n\}$ is a set of n variables; $D = \{D(x_1), \ldots, D(x_n)\}$ is a collection of finite discrete domains, $D(x_i)$ is the set of x_i's possible values; C is a set of constraints. Each constraint $c \in C$ is defined on the ordered set of variables $var(c)$ (its scope). Value tuples permitted by c are in $rel(c) \subseteq \prod_{x_j \in var(c)} D(x_j)$.*

A *solution* is an assignement of values to variables such that all constraints are satisfied. CSP solving is NP-complete.

3 Subsumption

It is easy to see that testing subsumption between two feature terms ψ_1 and ψ_2 (testing if there exists a mapping m, as defined in Section 2, to detect whether ψ_1 is more general than or equal to ψ_2) can be formalized as a CSP as follows:

- CSP Variables: for each feature term variable $X \in vars(\psi_1)$ there is a CSP variable x that contains its mapping $m(X)$ in ψ_2. To avoid confusion between the two types of *variables*, feature term variables are written uppercase while CSP variables are written lowercase, the same letter denotes corresponding variables (x is the CSP variable that represents feature term variable X).[2]
- CSP Domains: the domain of each CSP variable is the set $vars(\psi_2)$, except for the CSP variable of $root(\psi_1)$, whose domain is the singleton $\{root(\psi_2)\}$.
- CSP Constraints: three types of constraints are posted
  - Constraints on sorts: for each $X \in vars(\psi_1)$, $sort(X) \leq sort(x)$.
  - Constraints on features: for each variable $X \in vars(\psi_1)$ and feature $f \in features(X)$, for each variable $Y \in X.f$ there exists another variable $Z \in x.f$ such that $y = Z$.
  - Constraints on difference: If $X.f = \{Y_1, ..., Y_k\}$, where all Y_i's are different by definition, the constraint *all-different*$(y_1, ...y_k)$ must be satisfied.

Since ψ_1 and ψ_2 have a finite number of variables, it is direct to see that the CSP has a finite number of CSP variables and all their domains are finite. Constraints on sorts can be easily tested using the sort ontology O; constraints on features and of diference are directly implemented since they just involve the basic tests of equality and difference.

The proposed CSP model of feature term subsumption $\psi_1 \sqsubseteq \psi_2$ is feasible. The number n of CSP variables is exactly $|vars(\psi_1)|$. The domain size of $n-1$ CSP variables is $|vars(\psi_2)|$, while for the remaining CSP variable is 1. Regarding constraints, denoting by m the maximum number of features, the maximum number of constraints required is:

[2] For X we use "feature term variable" or simply "variable". For x we always use "CSP variable".

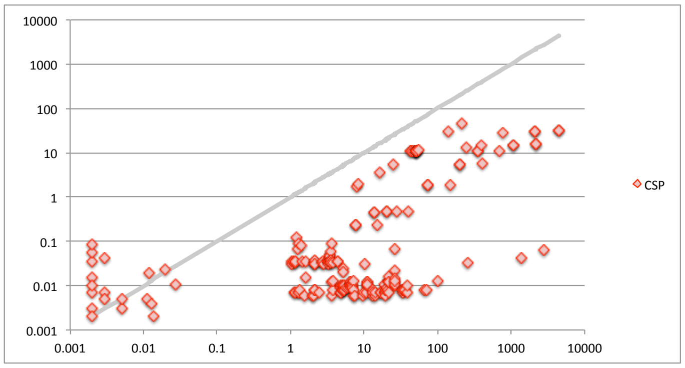


Fig. 3. Time required to compute subsumption using CSPs compared to a standard approach.

- n binary constraints on sorts (one per CSP variable),
- $O(n^2m)$ binary constraints on features (number of possible pairs of variables times the maximum number of features),
- $O(nm)$ n-ary constraints on difference (number of variables, each having one *all-different* constraint, times the maximum number of features).

In practice, n varies from a few variables in simple machine learning problems to up to hundreds or thousands for complex biomedical datasets. Most machine learning datasets do not have more than a few different feature labels, and thus m usually stays low. Moreover, in practice, the actual number of constraints is far below its maximum number as computed above.

4 Experimental Results

In order to evaluate our model, we compared the time required to compute subsumption by a standard implementation of subsumption in feature terms [4] with our implementation based on constraint programming. We generated 200 pairs of feature terms using the examples in two relational machine learning data sets as the source of terms: the trains data set [9], and the predictive toxicology data set [5].

Figure 3 shows the results of our experiments, where each dot represents one of the 200 pairs of terms used for our evaluation. The vertical axis (in a logarithmic scale), shows the time in seconds required to compute subsumption by the traditional method, and the horizontal axis (also in a logarithmic scale), shows the time in seconds required to compute subsumption using constraint programming. Points that lay below the grey line correspond to problems where constraint programming is faster. For example, we can see there was one problem where a traditional approach required 4413 seconds vs 31 seconds for the CP approach. The results show that the harder the problems, the larger the benefits, and that the CP approach is always faster, except for very small problems (that have little practical impact).

5 Conclusions

A key obstacle when applying relational machine learning techniques to complex domains is that the basic operations like subsumption have a very high computational cost. In this paper we presented a method for assessing subsumption in set-valued feature terms using constraint programming (CP), allowing us to solve those operations in a more efficient way than using traditional methods. As part of our ongoing work we are analyzing the effect of our constraint programming model of subsumption in unification and antiunification of feature terms.

References

[1] Aït-Kaci, H.: Description logic vs. order-sorted feature logic. In: Description Logics (2007)

[2] Aït-Kaci, H., Podelski, A.: Towards a meaning of LIFE. Tech. Rep. 11, Digital Research Laboratory (1992)

[3] Aït-Kaci, H., Sasaki, Y.: An axiomatic approach to feature term generalization. In: Proceedings of the 12th European Conference on Machine Learning (EMCL '01). Lecture Notes in Computer Science, vol. 2167, pp. 1–12. Springer-Verlag, London, UK (2001)

[4] Arcos, J.L.: The NOOS representation language. Ph.D. thesis, Universitat Politècnica de Catalunya (1997)

[5] Armengol, E., Plaza, E.: Lazy learning for predictive toxicology based on a chemical ontology. In: Dubitzky, W., Azuaje, F. (eds.) Artificial Intelligence Methods and Tools for Systems Biology, vol. 5, pp. 1–18. Springer-Verlag (2005)

[6] Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)

[7] Carpenter, B.: The Logic of Typed Feature Structures, Cambridge Tracts in Theoretical Computer Science, vol. 32. Cambridge University Press (1992)

[8] Dietterich, T., Domingos, P., Getoor, L., Muggleton, S., Tadepalli, P.: Structured machine learning: the next ten years. Machine Learning pp. 3–23 (2008)

[9] Larson, J., Michalski, R.S.: Inductive inference of vl decision rules. SIGART Bull. (63), 38–44 (1977)

[10] Lavrač, N., Džeroski, S.: Inductive Logic Programming. Techniques and Applications. Ellis Horwood (1994)

[11] Maloberti, J., Sebag, M.: Theta-subsumption in a constraint satisfaction perspective. In: Proceedings of the 11th International Conference on Inductive Logic Programming. pp. 164–178. ILP '01, Springer-Verlag, London, UK (2001)

[12] Ontañón, S., Plaza, E.: On similarity measures based on a refinement lattice. In: Wilson, D., McGinty, L. (eds.) In ICCBR-2009. p. to appear. Springer-Verlag (2009)

[13] Plaza, E.: Cases as terms: A feature term approach to the structured representation of cases. In: Veloso, M., Aamodt, A. (eds.) Case-Based Reasoning, ICCBR-95, pp. 265–276. No. 1010 in Lecture Notes in Artificial Intelligence, Springer-Verlag (1995)

[14] Plotkin, G.D.: A note on inductive generalization. In: Machine Intelligence. No. 5 (1970)