in: Proc. international symposium on "Knowledge Retrieval, Use, and Storage for Efficiency", Santa-Cruz (CA US), pp69-73, 1995

Objects, types and constraints as classification schemes (abstract)

Cécile Capponi, Jérôme Euzenat, Jérôme Gensel

INRIA Rhône-Alpes — IMAG-LIFIA
46 avenue Félix Viallet, 38031 Grenoble Cedex 1 (France)
{Cecile.Capponi,Jerome.Euzenat,Jerome.Gensel}@imag.fr

Abstract. The notion of classification scheme is a generic model that encompasses the kind of classification performed in many knowledge representation formalisms. Classification schemes abstract from the structure of individuals and consider only a sub-categorization relationship. The product of classification schemes preserves the status of classification scheme and provides various classification algorithms which rely on the classification defined for each member of the product. Object-based representation formalisms often use heterogeneous ways of representing knowledge. In the particular case of the TROPES system, knowledge is expressed by classes, types and constraints. Here is presented the way to express types and constraints in a type description module which provides them with the simple structure of classification schemes. This mapping allows the integration into TROPES of new types and constraints together with their sub-typing relation. Afterwards, taxonomies of classes are themselves considered to be classification schemes which are products of more primitive ones. Then, this information is sufficient for classifying TROPES objects.
Keywords: Class — object — type — constraint — classification scheme — sub-type inference.

Classification has been formalized for specific languages such as terminological logics (as subsumption), types (as sub-typing) or object representation (as classification). However, when used for applications, practical implementation of such systems are confronted with alien formalisms which do not comply their particular rules. We designed a formalism, classification schemes (CS), aimed at comparing the way these many systems classify. For that purpose the specific syntactic features of the above formalisms are hidden focusing only on what is relevant to classification. Here we present how this can also be used for integrating different notions of classification in the same system. Types and constraints are shown to be CS, thus, provided with a set of types and constraints, objects may be constructed through product operations over CS, grounding the classification operation over objects.

First, the classification scheme formalism is briefly outlined (§1); then the introduction and management of types and constraints into TROPES are presented in order to show that they are classification schemes (§2).

1 Classification schemes

The classification scheme formalism [5] is an attempt to represent, in a very abstract way, the classification activity of any knowledge representation system (thus not only terminological systems). Instead of considering the classification activity to be a consequence of the structure of entities, it considers classification to be primitive and builds on it. CS are first defined. Then, the operations of classification are presented and the way CS can be defined through the operations of degeneration, restriction and product of other CS is presented.

1.1 Taxonomies and classification schemes

Hereafter, taxonomies and CS are defined with regard to a *language of categories* L_C and a *language of individuals* L_I. *Individuals* have an interpretation as individuals in a modeled domain. *Categories* are interpreted as sets of individuals.

DEFINITION (interpretation). There are two *interpretations* for a category c:

- the abstract interpretation $I_A(c)$ is the set of individuals described by the expression c;
- the real interpretation $I_R(c)$ is the set of individuals denoted by c in the modeled domain.

The category description must indeed cover its denotation; this is expressed by the constraint of *interpretation inclusion:* $I_R(c) \subseteq I_A(c)$.

The interpretation may depend on the considered knowledge model (for instance, $L_I = \mathbb{N}$ and L_C is the set of integer intervals). The categories are interpreted as representing sets of objects. However, the language in which they are expressed could be inadequate for characterizing them exactly. This is the reason for having two interpretations: the real interpretation corresponds to the set of individuals the category *is meant to* represent, while the abstract interpretation is the set of individuals the description of the category *can* represent (for instance, in the language of integer intervals, it is not possible to find a class which describes exactly the set {1, 3}, for that purpose the interval [1..3] whose abstract interpretation is {1, 2, 3} can be used for denoting, or be really interpreted as {1, 3}).

DEFINITION (taxonomy): A *taxonomy* $\langle C \leq \rangle$ with regard to an interpretation I (abstract or real) is made of a set C of categories and of a partial order relation named *sub-categorization* ($\leq$) over these categories so that the *extension inclusion property* is satisfied: $c' \leq c \Rightarrow I(c') \subseteq I(c)$.

A taxonomy reflects the organization of categories in an inclusive structure which enables classification systems to draw inferences such as transitivity of non membership along the sub-categorization relation. Note that this definition does not consider sub-categorization and extension inclusion as equivalent (i.e. $c' \leq c \not\Leftarrow I(c') \subseteq I(c)$): this allows non-exclusive categories.

DEFINITION (classification scheme): A *classification scheme* $\langle C \leq \ll \rangle$ over a couple of languages L_C and L_I is given by a set of category (C), a sub-categorization relation ($\leq$) and a sub-categorization criterion ($\ll$), such that:

- $\langle L_C \ll \rangle$ is a taxonomy with regard to the abstract interpretation,
- $\langle C \leq \rangle$ is a taxonomy with regard to the real interpretation, and
- $\forall c, c' \in C, c \leq c' \Rightarrow c \ll c'$.

The sub-categorization criterion differs from $\leq$, which only ranges over C. Often, the sub-categorization relation is a "natural" order over L_C (sub-typing or set-inclusion for instance) while $\leq$ has to be built. Pursuing the example above, it means that the category c=[2..4] (with $I_R(c)$={2,3,4}) can be added in the taxonomy such that $c \leq$[1..4] because [2..4]$\subseteq$[1..4] (here, $\ll$ is $\subseteq$). So, constraining the sub-categorization relation to comply with $\ll$ — which is defined over L_C — ensures that, during the construction of $\leq$, the extension inclusion property is satisfied.

1.2 Classification

CS attempt to formalize the fact that a taxonomy has an interpretation and is carefully framed by a super-structure whose interpretation can be completely — in the formal meaning of completeness — taken into account by the implementation. They are sufficient to describe the classification operation.

DEFINITION (classification): From a set of categories C (such that $C \subseteq L_C$), the *classification* operation determines the set of categories $Cl(i,C)=\{c \in C;\ i \in I_A(c)\}$ under which the individual i is classified.

An individual i can be classified under a category c. Classification aims at finding categories whose real extension contains i; however, this is not mechanically accessible. Then, classification returns the set of categories whose abstract interpretation contains i. However, when abstract and real interpretations coincide, i belongs to the real interpretation of each of the categories. A particular taxonomy can be obtained by starting with an empty taxonomy and adding new categories and relationships respecting $\ll$, but also by starting from the largest possible taxonomy $\langle L_C \ll \rangle$ and simplifying it. This is achieved through restriction and degeneration.

DEFINITION (restriction and degeneration): A partial order relation $\leq$ over a set S can be defined by the graph whose nodes are elements of S and edges E are the subset of $S \times S$ containing all the couples of related elements. The *restriction* of $\leq$ to a set S' subset of S (noted $\leq|S'$) is the graph $\langle S'\ E' \rangle$ with E' the subset of E whose elements are elements of $S' \times S'$. A *degeneration* of $\leq$ is a graph $\langle S\ E' \rangle$ such that $E' \subseteq E$ is closed under transitivity and reflexivity.

PROPERTIES:
1) $\langle L_C \ll \rangle$ is a taxonomy with respect to I_A;
2) Any restriction of a taxonomy w.r.t. I is a taxonomy w.r.t. I;
3) Any degeneration of a taxonomy w.r.t. I is a taxonomy w.r.t. I.

Thus, many taxonomies can be constructed within a scheme and their interpretation is not restricted to be I_A.

1.3 Products

The definitions given so far do not account for the way classification is deduced from the structure of the terms. Products provide a foundation for such languages with constructors.

DEFINITION (product of classification schemes): Let $S_1,\ldots S_n$ be a set of classification schemes (for $j\in[1,n]$, $S_j=\langle C_j\ \leq_j\ \ll_j\rangle$ is defined upon the languages L_C^j and L_I^j), the product of classification schemes $S=\times_{j=1}^n S_j$ is defined by $\langle \times_{j=1}^n C_j\ \leq\ \ll\rangle$ on the languages $L_C=\times_{j=1}^n L_C^j$ and $L_I=\times_{j=1}^n L_I^j$ such that:

$$\forall c=(c_1,\ldots c_n)\in\times_{j=1}^n L_C^j, \forall c'=(c'_1,\ldots c'_n)\in\times_{j=1}^n L_C^j, c\ll c'\Leftrightarrow \forall j\in[1\,n], c_j\ll_j c'_j$$
$$\forall c=(c_1,\ldots c_n)\in\times_{j=1}^n C_j, \forall c'=(c'_1,\ldots c'_n)\in\times_{j=1}^n C_j, c\leq c'\Leftrightarrow \forall j\in[1\,n], c_j\leq_j c'_j$$

The abstract and real interpretations for classes are simply the sets of individuals which are obtained by the product of the corresponding interpretations of each term of the product. The two constraints define ≤ and ≪ as the usual product of partial orders. Note that the languages L_I and L_C are implicitly defined: they are products of the initial languages. There is only one way to obtain a CS from the same set of initial CS.

PROPERTY: A product of classification schemes is a classification scheme.

The assertions above are proved in [5] in which several classification algorithms for products of CS can also be found.

2 Objects, types and constraints

So far, a notion of CS has been presented with a well-defined notion of classification over CS. Nevertheless, the quality of being a CS is preserved through the operations of product, degeneration and restriction. This must be related to the object-based knowledge representation system TROPES [9]. In TROPES, objects are simply product of values and objects, classes are product of type expressions, classes and constraints and concepts are product of abstract data type (ADT) and concepts. So, if ADT and constraints can be organized in classification schemes (with their classification operations) then so are TROPES concepts (with an immediate classification operation). The types and constraints used in TROPES are presented below.

2.1 Types

TROPES allows to use values which are not objects, but requires the typing of these values. These values are members of ADT defined into a type system named METEO [4]. Additionally, METEO provides some polymorphic type constructors such as List or Set.

An ADT T necessarily contains an equality predicate between values of this type ($x=_T y$ means that x and y denote the same value for type T), and a typing predicate ($x\in T$ means that x is a value of the ADT T). In addition, an ADT may define other operators on its values: the order predicates on ordered types ($x\leq_T y$ means that x is before y for T), the successor operation for countable types, etc. From such information, METEO performs the encoding of the new ADT and makes it available for TROPES, independently from its implementation.

For instance, a user may want to introduce the new ADT `Date` whose values are triplets of integers: `(year, month, day)`. `Date` can be explicitly declared as a *countable* set of values:

- The user has to specify the two required predicates $=_{\texttt{Date}}$ and $\in$`Date`.
- Since `Date` is ordered, the user can specify the order relation $\leq_{\texttt{Date}}$ between two values of `Date`, so that METEO can represent domains of `Date` values as ranges.
- In order to express the fact that `Date` is countable, the user has to specify some elementary operations and properties, such as the successor operation.

So far, ADT provide a set of objects but no taxonomy. However, the TROPES description language allows to restrict a slot type through domain descriptors: *domain* restrict the possible values to a particular sub-set, *range*, to a particular set of intervals, *card*, to a particular cardinality, and so on [9]. For dealing with these expressions METEO carries out two main operations:

- normalization (or equality computation) which establishes a unique form from a particular type expression;

- sub-type computation which establishes wether the extension of the first type expression includes that of a second one.

The internal representation of type expressions depends on the kind of ADT considered: for instance if `Integer` is defined as enumerated, the type expressions will be sets, but if it is defined as ordered, they will be list of closed-bound intervals. Similarly, normalization and sub-type computation between type expressions of an ADT, depend on the necessary predicates of the ADT ($\in_T$ and $=_T$), and eventually from other predicates such as $\leq_T$. For instance, sub-type computation is obtained by checking the inclusion between two lists of closed-bound intervals.

Types constructed with the same constructor from the same primitive type are disposed in a lattice whose partial order relation is sub-typing and whose greater element is the referred ADT [3]. METEO performs the dynamic management of the type lattice associated to the primitive type (e.g. insertion and deletion of type, computation of the greatest lower bound). Furthermore, it provides algorithms in order to manage links of composition established between different type lattices, thus type modifications are dynamically propagated along these links.

Types introduced in TROPES provide elementary CS from which more complex ones are built. For a primitive type T, L_I corresponds to the values of the type and L_C corresponds to the set of equivalence classes of type expressions for normalization. The set of categories C is the result of the normalization of the type expressions used in concept expressions augmented in order to be a lattice. The sub-categorization criterion is the sub-typing relation over the whole L_C while the sub-categorization relation is its restriction to C. Then types are interpreted as sets (domains) and sub-typing as set inclusion. The real and abstract interpretation are the same and range over the normalized type expressions. Thus, METEO is able to associate with a primitive type T the behavior of a CS, i.e. it provides the computation of the sub-categorization criterion and relation.

For instance, the type `Integer` determines a CS in which L_I is $\mathbb{N}$ and L_C is the set of sets of intervals interpreted as its powerset. The sub-categorization criterion is sub-typing and the normalization operation ensures that the taxonomy is exactly the restriction of $\langle 2^{\mathbb{N}} \subseteq\rangle$ to the sets which are necessary to provide it with a lattice structure (i.e. the theoretical lattice over the powerset of integers restricted to sets relevant to the application).

2.2 Constraints

A constraint is interpreted as in Constraint Satisfaction Problems (CSP) [8]. A CSP considers a set of n variables v_j with associated domains d_j and a set of relations r_i. A relation $r_i\ (v_{i1}, v_{i2}, \ldots, v_{im})$ is a sub-set of the Cartesian product of the domains $D_{i1}\times\ldots\times D_{im}$ (where D_{ik} is the domain of a variable v_{ik} constrained by r_i). A constraint in a class is a declarative statement which expresses a relation that holds between several fields of this class.

In TROPES, both unary constraints and n-ary (where $n > 1$) constraints can be defined and set up among the fields of one or several objects. A unary constraint consists of a predicate which extends the field type description. A n-ary constraint corresponds to a relation which must hold between n fields of a class. These two kinds of constraints are necessary conditions that every member of the class must satisfy. Constraints of different kinds are expressed in intension and correspond to basic constraint functions. These basic constraint functions are combined to form more complex constraint expressions. Besides classical constraints, i.e. numerical constraints based on a comparison (=, <, >, ≤, ≥, ≠) of arithmetic expressions built with the operators (+, -, *, /, ^, log, exp, sin, cos, etc.), more specific ones are available like constraints on sets or lists of values of any types, or like conditional constraints.

A constraint programming module, called MICRO [7], has been designed for TROPES. It handles CSP whose domains are enumerated sets or intervals. Thus, domains may be finite or infinite. MICRO takes advantage of the presence of METEO to propose generic constraints (constraints whose predicate depends on the type of the variables). It includes a constraint propagation algorithm (which achieves arc-consistency [8] for monotonous operators used in constraint expressions).

In order to guarantee sub-typing of constrained fields between a class and its sub-classes, constraints must be inherited from one class to its sub-classes. Then, the final set of constraints of a class results from the union of the proper set of constraints of this class and of the set of constraints of its super-classes. However, this constraint inheritance should not be systematic, particularly when redundant constraints of class are defined. A constraint to be added to a class is said to be redundant with regard to the current set of constraints when, either it can be obtained after simplification or derivation from the current set of constraints, or its predicate is implied by a conjunction of predicates stemming from the current set of constraints. Indeed, redundancy can be expressed as a set-inclusion between relation extensions. If two n-ary predicates P and Q defined on the same Cartesian product of primitive domains $D=D_1\times\ldots\times D_n$ are so that the set of n-uples satisfying P is a sub-set of those satisfying Q, then, a sub-typing relation $\leq_D$ between P and Q (such that $P \leq_D Q$) can be considered on D. This sub-typing relation allows to organize the structures representing constraints in a taxonomy.

Since TROPES cannot compare extensions of predicates (simply because they might be infinite) nor their intension, it has to be provided with information concerning sub-categorization of constraints. The user does it by declaring the taxonomy of constraints. TROPES does not automatically maintain the sub-categorization between these constraints. The system is only able to check these constraints for satisfaction. The user is free to organize the constraints with a sub-categorization relation.

This constitutes the classification schemes tied to constraints: a particular constraint r over the variables $v_1, \ldots v_n$ with domains $D_1, \ldots D_n$ is easily included in a CS whose L_I is the Cartesian product of domains $D = D_1 \times \ldots D_n$ and L_C is a set of predicate which recognize sub-sets of D (thus sub-sets of L_I). C is simply the set of expressed predicates (thus containing c). As for types, the abstract and real interpretations coincide and range over D. The sub-categorization criterion is interpreted as the inclusion of the predicate extension (note that this is not accessible by the program) and the sub-categorization relation its restriction to C expressed by $\leq_D$. Thus, it is possible to use this CS in the classification operation and even if the constraints use classic constraint satisfaction algorithms for deciding if c is satisfied of a tuple of variable (resp. if that individual belongs to the abstract interpretation of the c category), the external behavior of the CS remains.

3 Conclusion

The extension facilities of the TROPES knowledge representation system have been presented. Type definitions through ADT are very straightforward with regard to the kind of type expressions and inference they allow. So, TROPES can manipulate external expressions in a minimal fashion without having to access the semantics of these expressions. Types and constraints have been characterized as classification schemes. More elaborate CS such as TROPES concepts are justified only because they result from the combination (product, restriction, weakening) of these more simple CS which always produces new CS. Thus, thanks to the CS formalism, TROPES is able to perform classification (but also class classification which has not been presented here).

The problem has been addressed by others in the framework of terminological logics [1,2,6]. But, the extendibility mechanisms are not as general as CS. For instance, organizing constraints into a taxonomy as presented here requires the possibility of having non complete relationship between constraints: this is not possible with definitional categories. Implementing the CS principles allow both easy extension and simple classification algorithms. This could affect performances, however the algorithm relies, finally, on the algorithms provided for each type and constraint satisfaction. It is expected that this has better performances than the complete expression of the basic types in the representation formalism.

References

1. Franz Baader, Philipp Hanschke, A scheme for integrating concrete domains into concept languages, Proc. 12th IJCAI, Sydney (AU), pp452-457, 1991
2. Alex Borgida, Ronald Brachman, PROTODL: a customizable knowledge base management system, Proc. 1st Conference on Information and Knowledge Management, Baltimore (MA US), pp482-490, 1992
3. Cécile Capponi, Interactive class classification using types, in Edwin Diday, Yves Lechevallier, Martin Schader, Patrice Bertrand, Bernard Burtschy (eds.), New approaches in classification and data analysis, Springer-Verlag, Berlin (DE), pp204-211, 1994
4. Cécile Capponi, Identification et exploitation des types dans un modèle de représentation des connaissances par objets, PhD thesis, université Joseph-Fourier, Grenoble (FR), 1995 to appear
5. Jérôme Euzenat, Classification schemes: definitions, properties, algorithms, in preparation
6. Brian Gaines, A class library implementation of a principled open architecture knowledge representation server with plug-in data types, Proc. 13th IJCAI, Chambéry (FR), pp504-509, 1993
7. Jérôme Gensel, Integrating constraints in an object-based knowledge representation system, Proc. International Workshop on Constraint Processing at CSAM'93, St Petersburg (RU), Document D-93-14, DFKI, Kaiserslautern (DE), pp51-64, 1993
8. Alan Mackworth, Consistency in networks of relations, *Artificial intelligence* 8(1):99-118, 1977
9. Sherpa project, TROPES 1.0 reference manual, internal report, INRIA Rhône-Alpes, Grenoble (FR), 1995 `ftp://ftp.imag.fr/pub/SHERPA/rapports/tropes-manual.ps.gz`