Cases as terms:
A feature term approach to the structured representation of cases

Enric Plaza*
Institut d'Investigació en Intel·ligència Artificial, CSIC
Campus de la Universitat Autònoma de Barcelona
08193 Bellaterra, Catalunya, Spain
Email: enric@iiia.csic.es
URL: http://www.iiia.csic.es

1 Motivation

In our research work, we have come to represent cases as complex, structured data structures that we will formally described as feature terms (see [Arcos] for a description of the NOOS language for CBR and multistrategy learning). The advantage of using structured representations are twofold. Firstly, it offers a natural way to describe composite objects that if described by attribute-value representations cause some problems like dealing with irrelevant attributes, not-applicable values, etc (and this leads to problems when comparing similitude among cases in this descriptions). Secondly, structured-representation cases offer the capability of treating subparts of cases also as full-fledged cases: they can be stored, and retrieved and used to solve (sub)problems of new cases; also, a case may be solved using (subparts of) multiple cases retrieved from the system's memory. We will present here a formalization of structured (sometimes called object-centered) representations as feature terms and we will present how can we assess similarity between feature terms (cases) and determine the preferred (most similar) case from a set of cases.

1.1 Background

This approach is different from some usual notions of case-based reasoning (CBR). Usually, cases are (represented as) tuples of attribute value pairs for a known, finite vocabulary of attributes and values (except for numeric values that need not be finite). Then, similarity is estimated using some metrics (that may involve differential weighting of attributes, dynamic adjustment of the weights, etc). The usual thing is to have some attribute-wise distance measure, i.e. if attributes are \(a_i (i = 1, \ldots, n)\) and we use the notation \(t.a_i\) to denote the value of attribute \(a_i\) in case \(t\), then attribute-wise distance is some function \(d(t_1.a_i, t_2.a_i)\) that may be defined according to the \(a_i\) attribute's type. Then some aggregation function \(D\) is defined as the weighted distance of two cases \(t_1, t_2\), such as:

\[
D(t_1, t_2) = \sum_{i=1}^{n} w_i \cdot d(t_1.a_i, t_2.a_i)
\]

There are some problems that have to be solved, like missing values, irrelevant values, etc. Next, some similarity function is defined as \(S(t_1, t_2) = N(1 - D(t_1, t_2))\) where \(N\) is a normalization function. Other CBR approaches do not use similarity but symbolic indices (sometimes learned by explanation-based generalization). However, this approach deal only with the what we call [Armengol] retrieval task, and not with the task of selection task of the retrieved cases [Plaza]; usually this CBR systems use some similarity-based or domain-specific strategy for ranking the retrieved cases. We will see how our approach deals with both retrieval and selection of structured cases.

* This work has been performed in the framework of Project ANALOG, CICYT grant TIC-122/93.
Indeed, the structured representation of cases we propose has the advantage of a wider expressive power than attribute-value representations. However, the classical approach of distance among tuples is not directly applicable. One option is that of extending distance metrics to structures representations as trees (see [Bunke]).

1.2 Towards similitude descriptions

Our approach is slightly different. Let's ask ourselves the reason of using distances: it is a way of estimating similarity between cases, and we want this because later the selection process in the CBR system will pick up as most relevant the most similar case. What we need then is a method for assessing similarity between cases (represented as structured terms) and obtaining a preference ranking among the cases based on that similarity. We do not need to use a distance (although it provides a complete ordering) if we have a way to obtain a preference ordering (albeit a partial ordering) based on a similarity estimate.

In the following we will formalize structured representation of cases as feature terms and we will show how this formalization allows us to define a natural characterization of similarity and how it can be used in the retrieval and selection of cases. This formalization is general and hopefully useful for other CBR systems, but we will explain specifically how NOOS, the language for CBR and multistrategy learning we have developed, is able to represent and implement all the concepts and methods we introduce. Feature terms can thus be considered as the foundation of CBR representation in NOOS-based applications [Arcos]. The structure of the paper is as follows: first feature terms are introduced and formalized, next the notions of subsumption and antiunification of feature terms are defined. Antiunification provides a description of the similitude among terms (cases). These notions are then used to realize a strategy for selection of retrieved cases using on a preference ordering based on a ranking of similitude descriptions.

2. Feature terms

Before defining formally feature terms, we will introduce intuitively some basic notions. A structured representation of cases is one where \( t_i.a_i = t_j \), i.e. where the value of an \( a_i \) attribute of a tuple (case) \( t_i \) is another tuple \( t_j \). This fact allows to treat subparts of cases as full-fledged cases. The intuition behind feature terms is that the basic power of this representation is that of path equality. A path is a concatenation of attributes, e.g. if we have: \( t_i.a_i = t_2, t_2.a_j = t_3, t_3.a_k = t_4 \) then \( p = t_i.a_i.a_j.a_k \) is a path. (with value \( t_4 \).) Path equality \( (p = q) \) is a restriction upon the values the term's attributes appearing in two paths may take. For instance a path equality such as: \( t_p.a_i.a_j.a_k = t_q.a_m.a_n \) constrains the term's attributes to have the same value. That is to say, they may have any value as far as among them the following relationship holds: following those paths they lead to the same term.

In fact, feature terms are just a generalization of first order terms that are useful to have a declarative representation of record-like structures without the loss of the convenient instantiation ordering unification operation [Ait-Kaci]. Using first order terms we can represent a person in this way:

\[
person(John, Smith, 34, Jack-Smith, NYC\text{city})
\]

where we have to know the meaning of the arguments by its position, i.e. writing that predicate \( person \) is:

\[
person(firstname, lastname, age, father, city)
\]

Using feature terms each argument is denoted by a symbol identifier instead of position. This change allows to easily express incomplete information. We could not do that using the above first order term for \( person \). Of course there is a way of representing incomplete