Information Integration through Unification of Feature Structures

Antonio Badia
Computer Science and Computer Engineering Department
University of Arkansas, Fayetteville, AR, 72701
Email: `abadia@godel.uark.edu`

1 Introduction

Description Logics (DLs) have strong relationships with other formalisms -among them, feature grammars ([SHI]). Both can be seen as variants of the same type of *terminological logics*; however, the application areas are different (knowledge representation and reasoning, for DLs; linguistic analysis, for feature grammars); therefore the algorithms developed in each case are quite different ([NS2]). In particular, DLs have been used for database research, while feature grammars have not.

Heterogeneous information integration is an important area of research in databases ([ERS]). Its aim is to access and merge, in a meaningful way, data from several sources. Work on the area has successfully used DLs ([CGLNR]). In this paper we propose to use a typical operation of functional grammars, *unification*, as the main operation for heterogeneous information integration. This is a natural match, since intuitively unification *puts together* the information in two concepts as far as the concepts do not contradict each other.

2 Feature Structures

Consider given a set `L` of labels and a set `V` of *atomic values*. A feature is either the special value `NULL`, an atomic value `a` (for some `a` $\in$ `V`), or a set of feature structures, where each feature structure is of the form `(l, v)`, with `l` $\in$ `L` and `v` a feature. We define the collection of labels of a feature `f` (in symbols `LABELS(f)`) as follows: `LABELS(NULL)` = `LABELS(a)` = $\emptyset$; and `LABELS`($\{$`fs`$_1$`,...,fs`$_n\}$) = `LABELS(fs`$_1$`)` $\cup \ldots \cup$ `LABELS(fs`$_n$`)`, where each `fs`$_i$ ($1 \leq i \leq n$) is a feature structure. Finally, `LABELS(fs)` = $\{$`l`$\}$ $\cup$ `LABELS(v)`, for a feature structure `fs = (l, v)`[1].

Let `f` denote a feature. In the following, we require thar `LABELS(f)` be a set. If `l` $\in$ `LABELS(f)`, there is a feature structure `(l v)` in `f`; the value `v` is denoted by `f/l`, and the whole feature structure by `f` $\rightarrow$ `l`.

[1]We abuse the notation by extending the definition to both feature and feature structures.

Feature structures can be seen as (descriptions of) directed graphs, which in turn can be interpreted as finite state automata (FSA) ([PER]). A *structure* A is a FSA $< S, s_0, \delta, \pi >$, where S is a set of states, $s_0 \in S$ is the initial state (which corresponds to the root of the graph), $\delta : S \times F \rightarrow S$ is the transition (next state function) (where the transition labels are features), and π is a (partial) function mapping states to elements in `V`, and whose domain is included in the set of *final states* (states for which δ is undefined). Given automaton A and label `l`, we say that A/`l` is the subautomaton that starts at state s_i[2], if $\delta(s_o, \texttt{l}) = s_i$ (whenever $\delta(s_o, \texttt{l})$ is undefined, A/`l` is undefined). Intuitively, structure A satisfies feature `f` if `f` is some `v` $\in$ `V` and A is the automaton with a single state labeled `v` and no transitions (ie. A = $< \{s\}, s, \emptyset, \{(s, \texttt{v})\} >$); or if `f` is a set of feature structures, and for each feature structure `(l v)` in `f`, A has a transition labeled `l` such that `A/l` satisfies `v`. Given an automaton A and an feature `(l v)`, we distinguish between three situations. First, `(l v)` is satisfied by A if A/`l` is defined and satisfies `v`. Second, `(l v)` is (merely) compatible with A if A/`l` is undefined. Third, `(l v)` is incompatible with A if A/`l` is defined and has value `x` $\neq$ `v`.

3 Unification

The unification algorithm proceeds in three steps. First, given two feature structures `f`, `g` to unify, the algorithm would divide them into *definite* and *indefinite* parts (the indefinite part covers all negations, disjunctions and conditionals). Second, the definite parts of both features are unified with an algorithm extended to check for relationships among features. Third, if there is no failure, the indefinite parts are processed by splitting them into negations, disjunctions and conditionals, and trying to unify with each part in turn with the result of the basic unification as reference[3].

[2]That is, the subgraph reachable from s_i.
[3]This approach is already outlined in [KA].

The standard unification algorithm only checks for clashes (i.e. the same label present on both structures but with different values on each). This behavior has some strong shortcomings for information integration. First, unification would succeed for two structures with a completely disjoint set of labels. But in unifying information, there is the question of whether two descriptions are about the same concept at all. There are two basic strategies to follow. The first one is to consider the problem as *external* to unification (i.e. consider unification only as deciding whether two descriptions are compatible) and leave the decision of what pairs of descriptions to unify to another part of the system. The second one is to extend the unification algorithm to decide whether the structures not only are compatible, but whether they are (likely to be) about the same concept. We say that two descriptions `f` and `g` are *connected* if they share some labels, that is, if `LABELS(f)` $\cap$ `LABELS(g)` $\neq \emptyset$. If, furthermore, all features involving these labels can be unified, we say that both descriptions are *compatible*. A unification algorithm can be easily modified to return `failure` if the descriptions given as arguments are not connected. The second problem is that the values of labels may be related in some way, and different labels may also be related[4]. For instance, features like `(at least x)` and `(at least y)` with `x` $\neq$ `y` would fail an straightforward unification test. However, these features can clearly be combined. In order to avoid this problem, we use the *lcs* algorithm introduced in [CBH][5]. Every pair of features is classified as *connected* or *not connected* ([BIG]). The connected features are classified as *compatible* and *incompatible*. Two features are connected if they have the same label or there is an operation that relates the two labels; otherwise they are not connected. To decide if two connected features are compatible, some basic semantic relations must be defined. *lcs* defines the unification for compatible features based on those semantic relations; in the example above, *lcs* would produce the feature `(at least (min(x,y)))`. For example, `(range x y)` and `(range z w)` are connected; their *lcs* is given by `(range (max x z) (min y w))`.

The semantics given above determines the behavior of the unification algorithm with respect to negation and conditionals. Unification for two features `f` and `g` will fail only (as far as negation is concerned) if one contains a feature structure `(l v)` and the other contains a feature structure `(l (not v))`. Also, given condition `(cond f g)`, the unification algorithm will check that when A satisfies `f`, it must satisfy `g`. When A is incompatible with `f`, no constraint is put on the relation between A and `g`, and when A is (merely) compatible with `f`, `g` must be (merely) compatible with A or `(not f)` must be (merely) compatible with A.

[4]Some labels would correspond with well-known *constructors* used in DLs, like `not, at least`,... These are assumed to have the standard semantics.

[5]Note that *lcs* is introduced in the context of DLs, which use a different syntax. In particular, a *constructor* (label) may have several arguments, including roles (binary relations). However, *lcs* is defined in [CBH] "ignoring role identity", i.e. the definition depends basically on the constructor, and therefore the idea is still applicable in our case.

4 Conclusion and Further Research

This paper reports work in progress; many issues are being currently explored. Chief among them is the need to extend the formalism to deal with the full power of DLs, by adding quantification, roles and *paths* (expressed in DLs with the constructor `SAME-AS`) to the language under consideration.

References

[BIG] Blanco, J., Illarramendi, A. and Goni, A., *Building a federated Relational Database System: An Approach Using a Knowledge-Based System*, International Journal of Intelligent and Cooperative Information Systems, vol. 3, n. 4, 1994.

[CGLNR] Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D. and Rosati, R. *Description Logic Framework for Information Integration*, Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning, Morgan-Kaufmann, 1994

[CBH] Cohen, W., Borgida, A. and Hirsh, H., *Computing Least Common Subsumers in Description Logics*, in Proceedings of AAAI-92.

[ERS] *Management of Heterogeneous and Autonomous Database Systems*, Elmagarmid, Rusinkiewicz, and Seth, editors, Morgan Kauffman Publishers, 1999.

[KA] Kasper, R. *Feature Structures: A Logical Theory with Applications to Language Analysis*, Ph.d Thesis, Univ. of Michigan, 1987.

[NS2] Nebel, B. and Smolka, G. *Representation and Reasoning with Attribute Descriptions*, in Bläsius, Hedstück, Rollinger, eds. *Sorts and Types in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, 418, Springer-Verlag, 1990.

[PER] Pereira, F. *Grammars and Logics of Partial Information*, in Proceedings of the 4th International Conference on Logic Programming, MIT Press, 1987.

[SHI] Shiebert, S. *An Introduction to Unification-Based Approaches to Grammar*, CSLI Lecture Notes, number 4, Stanford University, 1986.