Information, Unification and Locality

Fernando C. N. Pereira
Artificial Intelligence Center
SRI International

October 14, 1986

1 Unification Technology

In this discussion, I am wearing the humble hat of a "symbolic systems engineer" who has been involved in building artifacts, unification-based grammar formalisms, that might be used to describe certain aspects of some natural languages. In the same way a physicist might see the calculus as one of many technologies she needs in her scientific pursuits, and mathematicians as excessively glorified engineers, so it may well be reasonable for a linguist to look upon computational linguists of my formal persuasion as engineers building generic language-description tools.

One needs to be clear about the role of mathematical tools in scientific pursuits. Most differential equations one might write correspond to no physically-realizable system. The calculus imposes only very broad, weak constraints on the class of systems it can describe, eg. differentiability of state functions and trajectories. Similarly, there is no reason to believe grammar formalisms can impose strong lingustically-motivated constraints on the classes of languages they can describe.

Unification-based grammar formalisms are thus a subject of inquiry related to but independent from linguistic theory. The kinds of questions one asks about a formalism are then those from formal language theory (generative power, recognition complexity) and programming language design (semantics, expressive capabilities, implementation algorithms and data structures).

2 Unification in Abstract

In a unification framework we deal with a domain of *descriptions* $\mathcal{D}$ that are used to classify objects from some class under analysis $\mathcal{O}$, utterances and their fragments in the case that concerns us here. Classification is given by a *description* relation $\models$ between objects in $\mathcal{O}$ and elements of $\mathcal{D}$. If d is a [partial] description of e, we write $e \models d$, e *satisfies* (or is described by) d. The set of all objects that satisfy a description d is written $[\![d]\!]$.

Descriptions are in general *partial*, that is, a description d may in general be extended to a more specific (more informative) description d'. With suitable technical assumptions, this gives a partial order $d \sqsubseteq d'$ on descriptions. In terms of the description relation $\models$, $d \sqsubseteq d'$ iff for every object e, $e \models d$ whenever $e \models d'$.

Two descriptions d and d' are *compatible* if there is a description d'' such that $d \sqsubseteq d''$ and $d' \sqsubseteq d''$, that is if d and d' can both be extended to a single description more informative than both.

If two descriptions d and d' are compatible, it is reasonable to assume that there is a least specific description $d \sqcup d'$ more specific that both d and d'. In other words, $d \sqcup d'$ contains all the information in d and d', but no more. For historical reasons, $d \sqcup d'$ is called the *unification* of d and d'. In more standard mathematical terminology, $d \sqcup d'$ is the *join* of d and d'.

In terms of the description relation, if $e \models d \sqcup d'$, $e \models d$ and $e \models d'$. Furthermore, we want unification to behave like logical conjunction: if $e \models d$ and $e \models d'$, $e \models d \sqcup d'$. Thus $[\![d \sqcup d']\!] = [\![d]\!] \cap [\![d']\!]$ holds for any compatible descriptions d and d'.

The domains of objects, descriptions and the description relation are usually infinite, even though there may be some way of finitely characterizing the description relation. Such a characterization is a grammar.

To write grammars, we need to be able to constrain entities to satisfy arbitrarily complex descriptions without giving the descriptions in full. Our main instruments for this are *parameterized descriptions* and *rules*.

A parameterized description $d(p_1, \ldots, p_k)$ is not a description itself, but rather an encoding of a function from k-tuples of descriptions to descriptions. An object e satisfies such a parameterized description iff there are descriptions $f_1, \ldots, f_k$ such that e satisfies $d(f_1, \ldots, f_k)$. Given a family of parameterized descriptions $(d_i)_{i \in I}$ with parameters $(p_j)_{j \in J}$ and a set C of *constraints* involving the parameters, a family of objects $(e_i)_{i \in I}$ satisfies the parameterized descriptions relative to the constraints iff there are descriptions $(f_j)_{j \in J}$ that can be uniformly replaced for the parameters in such a way that $e_i \models d_i((f_j)_{j \in J})$ and all the constraints in C are satisfied.

In current unification-based formalisms, a grammar is a set of inductive rules with the general form

$$\frac{E_1 : d_1 \cdots E_n : d_n}{f(E_1, \ldots, E_n) : d} \quad \text{if} \quad C$$

where the E_i are object variables, d and the d_i are parameterized descriptions, C is a set of conditions on the parameters, and f is a ***structural composition*** function on objects. Given an appropriate notion of allowed derivation — a tree with nodes labelled by object-description pairs $e : d$ in which each node satisfies a rule — the grammar is *sound* with respect to a description relation $\models$ iff the root $e : d$ of every derivation allowed by the grammar obeys the condition $e \models d$.

The main technical question at this level is whether a given vocabulary of constraints and rules really define the description relation, that is, whether the rules are well-founded with respect to the structural decomposition of elements of $\mathcal{O}$ given by the structural composition functions f and whether there is a least description relation compatible with the derivations allowed by the grammar. This question has been answered in detail for definite-clause-based grammar formalisms (in which descriptions are just logical terms), but the situation is much less clear for more complicated domains (eg. those with disjunctive constraints) and classes of constraints (eg. LFG's constraint equations).

The preceding discussion may be summarized as follows. Descriptions classify objects (strings, utterance fragments). Rules give the description of a compound object in terms of descriptions of its parts. Grammatical analysis is the derivation of a description for an object according to the rules of a grammar that axiomatizes the description relation.

3 Computation

The discussion of the previous section concerned the *denotational semantics* of a grammar formalism. The denotational semantics gives a rigorous specification of *what* a grammar does. However, from a computational point of view we are also interested in *how* a grammar does what it does. This question can be posed at two levels: at the more abstract level of *operational semantics*, the problem is how to give abstract procedures (proof procedures) that construct derivation trees yielding classifications of objects (utterances); at the more concrete level, what data structures and algorithms are required for efficient analysis (classification, proof generation).

In some cases, for example in DCGs or PATR-II, the formalism itself is only semi-decidable, so the procedure given by the operational semantics may not terminate when it is asked to classify an object with a description it does not satisfy. However, for reasonable classes of grammars in those formalisms, as well as for all grammars in LFG, the classification (analysis) problem is decidable.

Even though LFG as well as all offline parsable DCGs and PATR-II grammars are decidable, it has been shown that they are intractable (NP-complete). The sources of this intractability are rather interesting, and lead to the question of *locality* which I will discuss in the next section.

We should not take the undecidability or intractability of unification-based formalisms as fatal flaw any more than we take the same properties as fatal flaws in a programming language. From an engineering point of view, grammar formalisms are just programming languages for grammar writing in which it is of course possible to write intractable or even nonterminating programs.

Finally, at the most concrete level, we have the question of what data structures and algorithms are in practice most useful for analysis in unification-based formalisms. The problem here is rather more difficult than that for simpler formalisms such as CFGs.

At an abstract operational level, derivations classifying a given string can be build very much in the same way as derivations in a context-free grammar. However, in contrast with CFGs, we have to satisfy not only local identity conditions between nonterminals but also rule constraints, which may have global consequences for the choice of descriptions in the derivation. In the typical incremental derivation procedures currently in use, rule applications assign to objects parameterized descriptions whose parameters may then be filled in by by unifications as specified by constraints and other rule applications. Alternative derivation paths assign different values to parameters, requiring some mechanism to segregate the assignments. It is easy to construct grammars that require space exponential on input length for storing alternative descriptions of the parts of the input even though the recognition problem for the grammars in question is clearly linear time with a specialized algorithm. This problem may well be related to the question of locality which I discuss below.

4 Locality

The intractability of existing unification-based formalisms seems to have something to do with *lack of locality*: constraints on description parameters are weak enough to allow the construction of (exponentially many) partial derivations for a string, but strong enough to conspire in rejecting almost all of those derivations. Basically, constraints chain together to constrain parameters arbitrarily far apart in a derivation. Lack of locality impinges also on the implementation data structures, as it allows rule applications to give alternative values to parameters arbitrarily deep inside a derivation, thus requiring a costly copying or structure-sharing mechanism to keep separate the alternative derivations.

Most grammars written in practice do not seem to suffer from lack of locality. However, it is not very clear why this is so and it is not easy to recognize lack of locality in a grammar. Even from an engineering point of view, it would be useful to have precise criteria for locality in a grammar, with suitable consequences in the complexity department. If locality criteria could be embodied in formal constraints on allowable grammars, we would be able to design restricted formalisms with good complexity properties.

The locality question has a more philosophical angle. Nonlocal grammars lead to nondeterministic recognizers. The description of a part of an utterance is supposed to represent the information contributed by that part to the utterance interpretation process. Lack of locality means that the processor is unable to identify the exact informational contribution of an utterance element without considering the whole utterance. This is rather unsatisfactory, as one might expect that the informational contribution of an object is precisely what can be learned from the presence of the object without regard to context. Lack of locality in a grammar thus suggests that we have not been successful in our classification of objects as to their information-carrying properties. Somehow, correct classification and determinism seem to go together. Current unification-based formalisms are expressive enough to describe a great variety of languages, but they do not seem to make all the classificatory distinctions that are needed to pin down the informational contributions of objects. Alternatively, there may well be a negative result lurking here that shows the impossibility of exact classification of informational content; in this case we should be able to find natural situations in which nonlocality emerges.