Zen and the Art of Symbolic Computing: Light and Fast Applicative Algorithms for Computational Linguistics

Gérard Huet

INRIA Rocquencourt,
BP 105, 78153 Le Chesnay Cedex, France,
`Gerard.Huet@inria.fr`,
`http://pauillac.inria.fr/~huet`

Abstract. Computational linguistics is an application of computer science which presents interesting challenges from the programming methodology point of view. Developing a realistic platform for the treatment of a natural language in its phonological, morphological, syntactic, and ultimately semantic aspects demands a principled modular architecture with complex cooperation between the various layers. Representing large lexical data bases, treating sophisticated phonological and morphological transformations, and processing in real time large corpuses demands fast finite-state methods toolkits. Analysing the syntactic structure, computing anaphoric relations, and dealing with the representation of information flow in dialogue understanding, demands the processing of complex constraints on graph structures, with sophisticated sharing of large non-deterministic search spaces.

The talk reports on experiments in using declarative programming for the processing of the sanskrit language, in its phonological and morphological aspects. A lexicon-based morphological tagger has been designed, using an original algorithm for the analysis of euphony (the so-called *sandhi* process, which glues together the words of a sentence in a continuous stream of phonemes). This work, described in [2], has been implemented in a purely applicative core subset of Objective Caml [5]. The basic structures underlying this methodology have been abstracted in the Zen toolkit, distributed as free software [3]. Two complementary techniques have been put to use. Firstly, we advocate the systematic use of *zippers* [1] for the programming of mutable data structures in an applicative way. Zippers, or *linear contexts*, are related to the interaction combinators of linear logic. Secondly, a *sharing functor* allows the uniform minimisation of inductive data structures by representing them as shared dags. This is similar to the traditional technique of bottom-up hashing, but the computation of the keys is left to the client invoking the functor, which has two advantages: keys are computed along with the bottom-up traversal of the structure, and more importantly their computation may profit of specific statistical properties of the data at hand, optimising the buckets balancing in ways which would be unattainable by generic functions. These two complementary technologies are discussed in [4].

The talk discusses the use of these tools in the uniform representation of finite state automata and transducers as *decorated* lexical trees (also

called *tries*). The trie acts as a spanning tree of the automaton search space, along a preferred deterministic skeleton. Non deterministic transitions are constructed as choice points with virtual addresses, which may be either absolute words (locating the target state by a path from the starting state) or relative *differential words* (bytecode of the trie zipper processor, representing the shortest path in the spanning tree of the state graph). Sharing such automata structures gives uniformly an associated equivalent minimal automaton. For instance, the lexicon is itself represented by its characteristic minimal recognizer. But this applies as well to possibly non-deterministic transducers. Thus our segmenting sandhi analyser compiles a lexicon of 120000 flexed forms with a data base of 2800 string rewrite rules into a very compact transducer of 7300 states fitting in 700KB of memory, the whole computation taking 9s on a plain PC.

We believe that our experiment with functional programming applied to lexical and morphological processing of natural language is a convincing case that direct declarative programming techniques are often superior to more traditional imperative programming techniques using complex object-oriented methodologies. Our programs are very short, easy to maintain and debug, though efficient enough for real-scale use. It is our belief that this extends to other areas of Computational Linguistics, and indeed to most areas of Symbolic Computation.

References

1. Gérard Huet. “The Zipper”. J. Functional Programming 7,5 (Sept. 1997), pp. 549–554.
2. Gérard Huet. “Transducers as Lexicon Morphisms, Segmentation by Euphony Analysis, And Application to a Sanskrit Tagger”. Draft available as `http://pauillac.inria.fr/~huet/FREE/tagger.pdf`.
3. Gérard Huet. “The Zen Computational Linguistics Toolkit”. ESSLLI 2002 Lectures, Trento, Italy, Aug. 2002. Available as: `http://pauillac.inria.fr/~huet/PUBLIC/esslli.pdf`.
4. Gérard Huet. “Linear Contexts and the Sharing Functor: Techniques for Symbolic Computation”. Submitted for publication, 2002.
5. Xavier Leroy et al. “Objective Caml.” See: `http://caml.inria.fr/ocaml/index.html`.