Generalizing Large-Scale Unification Grammars Leads to Improving the Design of an Abstract Machine

Liviu Ciortuz

CS Department, University of Wales, Aberystwyth, UK. E-mail: vvc@aber.ac.uk

Abstract. We give here the *guidelines for further significant improvement* of the LIGHT Abstract Machine [3], the core of a deduction-based parsing system [2].
The *basic idea* behind this extension of the LIGHT AM design is that unification may be substantially improved if it will partition terms (or: feature structures), by focusing first on the feature paths which lead to unification failures.

Background: The opening/possibility for such a re-thinking of LIGHT AM follows naturally the finding that on a large-scale unification grammar like LinGO [4], there are only a reduced number of paths (we called them GR-paths) which contribute to selection of syntactic rules through unification, during parsing. This finding was the result of the application on LinGO of a learning technique called Generalized Reduction (GR, [1]). Inspired by Inductive Logic Programming (ILP, [5]), the GR technique searches for a (local) minimum of atomic constraints in the rule feature structures (FSs) which are responsible for the parsing results delivered by the grammar on a training test-suite/corpus of sentences.[1] Following learning, the GR-paths can be assumed to be known at the (re)compilation time.

Fact: As it is now the case in LIGHT AM — and in all other AMs performing compiled unification —, during the execution of a function which is the compilation of a feature structure, a set of X registers temporarily store the addresses of the cells which are the nodes of the tree representing that FS.

Critique: Unlike WAM/Prolog, in large-scale lexicalized unification grammars there are usually big feature structures and a reduced number of rules.[2] There is no explicit link between the X variables and the feature paths whose values are designated by these variables. On the other side, the content/value of those variables is discarded immediately after the FS gets constructed on the heap. The path values of a FS resident on the heap are computed again (and again!) during unification with other FSs, sometimes immediately after they have been discarded (at the end of another program/query function execution)!

Solution:

These recomputations can be very conveniently avoided through an elegant trade off with the consumption of memory resources. One one side the compiled form of rules will take the

[1] First, the grammar runs the test suite, and then annotates it with the correct (number of) parses. Then, the GR technique tries to gradually eliminate constraints from all rules, while keeping the parsing results unchanged. For each sentence in the test suite, the parses produced by the given grammar are taken as positive examples in the GR learning process. All other parses are considered negative examples. The paths (in rule FSs) which are retained by GR are called GR-paths. For LinGO, a large HPSG grammar for English implemented at CSLI, University of Stanford, we identified 187 GR-paths, out of a total of 494.

[2] For instance, in LinGO, there are 62 rules, and the average rule size is above one hundred nodes. The GR learning technique shows that in a first parsing phase, the size of the rules can be reduced with about 60%. A second, very efficient parsing phase eliminates the parses which have been over-generated by the first phase, due to the reduced/generalized rules. (The second phase may eventually be cancelled.)

GR-paths into account and will assigned them (and the corresponding X variables) fixed indices, and on the other side the X registers corresponding to GR-paths will be transformed (by replication) into persistent data structures stored in environments associated to FSs representing rule partitions — the rule LHS, and its RHS arguments. The X variables corresponding to the GR-paths will be directly addressed in the environments, and will not be destructed if unification succeeds. Therefore, there will be no need to recompute (or save and restore) them.

Non-GR paths will be dealt with as done in the current version of LIGHT AM, except that their treatment will follow that of GR-paths. Therefore FS unification (in both approaches: compiled or interpreted/on-line will be divided between the GR-partition (better: the GR-bounded part) of the FS, and the non-GR partition, i.e the feature (multi-rooted) substructure made of the values of non-GR paths, some of which are not known at compilation time.[3]

On the GR-partition, FS unification will then amount to three parts/steps:

1. the *glb*-sort test on the GR-path values corresponding to the GR-bounded partitions of the two FSs,
2. typed-checking (on-line expansion) when/where the above *glb* operations refines the root sort of non-leaf nodes of the resulting FS,
3. unification (interpreted/on-line) operations corresponding to coreferences.[4]

Concerning interpreted/on-line unification: As shown by a recent extension of the GR learning procedure, work which consists mainly to copying feature-value pairs can be cut off for those paths which are neither GR-paths nor extensions of maximal GR-paths.[5]

References

1. L. Ciortuz. Learning attribute values in typed-unification grammars: On generalised rule reduction. In D. Roth and A. van den Bosch, editors, *Proceedings of the 6th Conference on Natural Language Learning (CoNLL–2002)*, pages 70–76, Taipei, Taiwan, 2002. Morgan Kaufmann Publishers and ACL.
2. L. Ciortuz. LIGHT — a constraint language and compiler system for typed-unification grammars. In *KI-2002: Advances in Artificial Intelligence*, volume 2479, pages 3–17, Berlin, Germany, 2002. Springer-Verlag. Proceedings of the 25th Annual German Conference on AI, Aachen, Germany.
3. L. Ciortuz. LIGHT – another abstract machine for feature structure unification. In S. Oepen, D. Flickinger, J. Tsujii, and H. Uszkoreit, editors, *Efficiency in Unification-based Processing*, pages 167–194. CSLI Publications, The Center for the Studiy of Language and Information, Stanford University, 2002.
4. Daniel P. Flickinger, Ann Copestake, and Ivan A. Sag. HPSG analysis of English. In Wolfgang Wahlster, editor, *Verbmobil: Foundations of Speech-to-Speech Translation*, Artificial Intelligence, pages 254–263. Springer-Verlag, 2000.
5. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.

[3] It is only for this partition that traditional/regular X variables will be used (if the second unification/parsing phase is to be performed).

[4] The last two unification parts/steps will use look-up tables.

[5] In this extension of the GR procedure, not only the rule FSs (partitions, i.e. LHS and RHS arguments) are reduced to the GR-paths (projection), but also the FSs produced through unification can be limited to the projection of the GR-paths and the (open) extension of maximal GR-paths.