ZIPPERS-BASED EMBEDDING OF ATTRIBUTE GRAMMARS

PHD PROJECT

SUPERVISORS: *JOÃO SARAIVA & JOÃO PAULO FERNANDES*
DEPARTAMENTO DE INFORMÁTICA, UNIVERSIDADE DO MINHO

Abstract. The goal of this project is to define a powerful and elegant embedding of modern attribute grammar extensions to attribute grammars via the use of a zipper mechanism. Attribute grammars are a suitable formalism to express complex, multiple traversal algorithms. In recent years there has been a lot of work in attribute grammars, namely by defining new extensions to the formalism (like, forwarding and reference attribute grammars, etc), by proposing new attribute evaluation models (like lazy, circular evaluators) and by embedding attribute grammars (like first class attribute grammars). In this PhD project we will study how to design such extensions through a zipper-based embedding and we will study efficient evaluation models for this embedding. Finally, we will express several attribute grammars in our setting and we will compare the performance of our implementation to the evaluators produced by well-known attribute grammar based systems.

1. Goals of the Project

The aim of this project is to embed advanced features of the AG paradigm into a general purpose, lazy, and purely functional language. This goal builds upon the definition attribute grammars as a domain-specific embedded language in Haskell by João Paulo. That is, attribute grammars become a Haskell library of higher-order and lazy functions, and we want to model in this library new language concepts like circular attributes, aspects, higher-ordeness, forwarding, references, multiple inheritance, strategies, and, finally, incremental evaluation.

We also intend to conduct a systematic performance study of traditional attribute grammar evaluators *versus* their implementation as a library in Haskell. We want to verify with realistic examples if a highly optimized functional implementation of AGs (lazy, strict and deforested evaluators) is really faster than the simple Haskell library. The outcome of the performance experiments, will allow us to to recommend improvements both to existing compilers for Haskell, attribute grammar based systems, and incremental programming environments.

The phases described below constitute the main work areas for our work:

a) Design: For many applications, it is desirable to design and implement a special purpose language that is tailored to the characteristics of the problem domain. However, the design and implementation of a new programming language from scratch is usually costy. The idea

Date: October, 2010.

of embedded languages has been enthusiastically embraced by the functional programming community.

The first goal to achieve in this project is to enhance the zipper-based embedding of João Paulo with advanced AG features such as aspects, higher-order and circular attributes, references, multiple inheritance and incremental attribute evaluation. In order to achieve this, we will design and propose different implementations for each feature, and each proposal will possibly rely on a different advanced feature of the host language, Haskell. Then, we will conduct several experiments in order to realize which proposal achieves our goal as elegantly as possible.

b) Implementation: One of the uses of AGs today is in the creation of programming environments (also called language-based editing environments), that report on syntactic and semantic errors as the program is being constructed. In such an environment, it is important that the attribute values are computed *incrementally* after each edit action, re-using results of previous computations where possible. Reps and Teitelbaum were the first to demonstrate the feasibility of the idea. It is partly because of this emphasis on incremental computation that the embedding of attribute grammars into functional programming was ignored: it was not clear at all how incremental computation could be achieved. An important step was made by João Saraiva, who showed how the known attribute evaluation techniques could all be implemented in a strict, purely functional setting. He was however not able to apply these techniques as part of an implementation of attribute grammars as a software library in Haskell: a quite complex preprocessor was still required.

Acar *et al.* presented a new technique for incremental computation of functional programs, which is completely general as it can be used to make any program incremental. Furthermore, it is implemented as a library of functions in ML. A remarkable property of Acar's work is that it maintains a dynamic dependency graph, as opposed to the static dependency graph used in all previous work on attribute grammars. Due to this, it potentially requires few recomputations after the input has been changed. This theoretical advantage may however be outweighed in practice by the additional book-keeping required.

With our work we will study different execution models for our extensions. In particular, we will work on the development of incremental models of execution.

c) Benchmarks: We will apply the library developed in the previous phases to realistic examples (Java grammar, Pretty-printing optimal algorithm, etc). Then, we will compare the performance of the obtained implementations with equivalent implementations obtained by other AG systems. Firstly, we will compare the performance of our library against other functional AG embeddings, whenever such a comparison is possible (recall that most of the features we want to embed in our system are not available in functional embeddings. Secondly, we will compare our implementations against the ones derived by standard AG systems from AG expressed in special purpose languages.