## 0.1 UUAG

| | |
|---|---|
| Report by: | Jeroen Bransen |
| Participants: | ST Group of Utrecht University |
| Status: | stable, maintained |

UUAG is the *Utrecht University Attribute Grammar* system. It is a preprocessor for Haskell that makes it easy to write *catamorphisms*, i.e., functions that do to any data type what *foldr* does to lists. Tree walks are defined using the intuitive concepts of *inherited* and *synthesized attributes*, while keeping the full expressive power of Haskell. The generated tree walks are *efficient* in both space and time.

An AG program is a collection of rules, which are pure Haskell functions between attributes. Idiomatic tree computations are neatly expressed in terms of copy, default, and collection rules. Attributes themselves can masquerade as subtrees and be analyzed accordingly (higher-order attribute). The order in which to visit the tree is derived automatically from the attribute computations. The tree walk is a single traversal from the perspective of the programmer.

Nonterminals (data types), productions (data constructors), attributes, and rules for attributes can be specified separately, and are woven and ordered automatically. These aspect-oriented programming features make AGs convenient to use in large projects.

The system is in use by a variety of large and small projects, such as the Utrecht Haskell Compiler UHC (→ **??**), the editor Proxima for structured documents (http://www.haskell.org/communities/05-2010/html/report.html#sect6.4.5), the Helium compiler (http://www.haskell.org/communities/05-2009/html/report.html#sect2.3), the Generic Haskell compiler, UUAG itself, and many master student projects. The current version is 0.9.50 (August 2013), is extensively tested, and is available on Hackage. There is also a Cabal plugin for easy use of AG files in Haskell projects.

Some of the recent changes to the UUAG system are:

**OCaml and Clean support.** We have added OCaml and Clean code generation such that UUAG can also be used in OCaml projects and in Clean projects.

We are currently working on the following enhancements:

**Evaluation scheduling and fixed point computation.** We are running a project to improve the scheduling algorithms in combination with fixed point computations. The currently implemented algorithms for scheduling AG computations do not fully satisfy our needs; the code we write goes beyond the class of OAGs, but the algorithm by Kennedy and Warren (1976) results in an undesired increase of generated code due to non-linear evaluation orders. However, because we know that our code belongs to the class of linear orderable AGs, we would like to find and algorithm that can find this linear order, and thus lies in between the two existing approaches.

Fixed point computations and ordered AG computations do not easily mix. In some cases however we would like to include fixed point computations inside an ordered AG, so the second aim of this project is to introduce way of letting the user specify such computations, and evaluate these in the right way.

**Incremental evaluation.** We are currently also running a Ph.D. project that investigates incremental evaluation of AGs. In this ongoing work we hope to improve the UUAG compiler by adding support for incremental evaluation, for example by statically generating different evaluation orders based on changes in the input.

### Further reading

- http://www.cs.uu.nl/wiki/bin/view/HUT/AttributeGrammarSystem
- http://hackage.haskell.org/package/uuagc