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 0.9.42.1 (November 2012), is extensively tested, and is available on Hackage. There is also a Cabal plugin for easy use of AG files in Haskell projects. Recently, we have improved the building procedure to make sure that the UUAGC can both be built from source as well as from the included generated Haskell sources, without the need of an external bootstrap program. Also, we added code generation for Ocaml.

We are working on the following enhancements of the UUAG system:

First-class AGs. We provide a translation from UUAG to AspectAG (→ **??**). AspectAG is a library of strongly typed Attribute Grammars implemented using type-level programming. With this extension, we can write the main part of an AG conveniently with UUAG, and use AspectAG for (dynamic) extensions. Our goal is to have an extensible version of the UHC.

Ordered evaluation. We have implemented a variant of Kennedy and Warren (1976) for *ordered* AGs. For any absolutely non-circular AGs, this algorithm finds a static evaluation order, which solves some of the problems we had with an earlier approach for ordered AGs. A static evaluation order allows the generated code to be strict, which is important to reduce the memory usage when dealing with large ASTs. The generated code is purely functional, does

not require type annotations for local attributes, and the Haskell compiler proves that the static evaluation order is correct.

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