0.1 UUAG

Report by:	Arie Middelkoop
Participants:	ST Group of Utrecht University
Status:	stable, maintained

UUAG is the *Utrecht University Attribute Grammar* system. It is a preprocessor for Haskell which makes it easy to write *catamorphisms* (i.e., functions that do to any data type what *foldr* does to lists). You define tree walks 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 (→ **??**), the Helium compiler (→ **??**), the Generic Haskell compiler, UUAG itself, and many master student projects. The current version is 0.9.29 (July 2010), is extensively tested, and is available on Hackage.

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.

Fixpoint evaluation We incorporated a fixed-point evaluation scheme for circular grammars. A cycle is broken by specifying an initial value for an attribute on the cycle, and repeating the evaluation with an updated value until it converges.

Step-wise evaluation We provide the possibility to evaluate AGs step-wise. The evaluation for a nonterminal may yield user-defined progress reports, and we can direct the evaluation until the next progress report. With this mechanism, we can resolve non-determinism and encode breadth-first search strategies.

Further reading

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