Attribute Grammar assignment

ariem@cs.uu.nl

October 13, 2008

Introduction. Consider the accompanying program text, which represents a compiler that generates a HTML visualization of a LaTeX/XML-like text. The details of this program were explained during the AG introduction. This program text starts with a definition of the concrete syntax by means of a scanner and parser. The scanner and parser take the concrete syntax, and produce abstract syntax, defined by some data type definitions. The main part of the program text is the semantics of the abstract syntax in terms of attributes and rules. Finally, there is some Haskell code that runs the scanner, parser and AG code, and some helper code for pretty-printing to HTML.

Task. The task is now to extend the semantics in such a way that syntax of a marker for an index, and syntax to list keywords for this index, are supported:

```
\begin{paragraph}
  ...
\end{paragraph}

\keyword importante \end
\keyword interessante \end

\index
```

This syntax is interpreted as follows. At the place of the `index`-marker, a list of words of different sizes is inserted. These words are exactly those words specified with the `keyword` syntax. The size of each word is determined by the frequency of the occurrences of the word in the text of the paragraphs.

Procedure. The concrete syntax (scanner and parser) has already been extended to support the additional syntax. It is now up to you to extend the abstract syntax with suitable extra alternatives, and to give the semantics of the existing attributes for these extra alternatives. For this you need to add extra attributes as well, in order to transfer information from one part of the tree to another part.

One way to do this is to first gather a list of all defined keywords, using a synthesized attribute using the `USE` syntax, and generating a singleton list at a `keyword` node. At the root of the tree, transform this list into an environment mapping a keyword to a frequency (`Map String Int`), which is initialized with `0`. Then, create a threaded attribute for threading this environment through the tree, adding to the frequencies at each `paragraph` node. Back at the root of the tree, you get the resulting environmen containing the total frequencies, and you can create an inherited attribute to transfer this information to the `Index` node, where you can use this environment to generate the appropriate HTML.