CHALMERS | GÖTEBORG UNIVERSITY

Applications of Functional Programming in Processing Formal and Natural Languages

Markus Forsberg

Thesis to be defended in public at **10:15, December 3, 2004**
in ES52, EDIT building, Göteborg
for the Degree of Licentiate of Engineering.

The defense will be held in English.

Opponent: Viggo Kann, Nada KTH,
Stockholm, Sweden

Department of Computing Science
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden
Telephone +46-31-772 10 00

Abstract

This thesis describes two applications of functional programming to process formal and natural languages. The techniques described in this thesis are closely connected to compiler construction, which is obvious in the work on BNF Converter.

The first part of the thesis describes the BNFC (the BNF Converter) application, a multi-lingual compiler tool. BNFC takes as its input a grammar written in Labelled BNF (LBNF) notation, and generates a compiler front-end (an abstract syntax, a lexer, and a parser). Furthermore, it generates a case skeleton usable as the starting point of back-end construction, a pretty printer, a test bench, and a LaTeX document usable as a language specification. The program components can be generated in Haskell, Java, C and C++, and their standard parser and lexer tools. BNFC itself was written in Haskell.

The methodology used for the generated front-end is based on Appel's books on compiler construction. BNFC has been used as a teaching tool in compiler construction courses at Chalmers. It has also been applied to research-related programming language development, and in an industrial application producing a compiler for a telecommunications protocol description language.

The second part of the thesis describes Functional Morphology, a toolkit for implementing natural language morphology in the functional language Haskell. The main idea behind is simple: instead of working with untyped regular expressions, which is the state of the art of morphology in computational linguistics, we use finite functions over hereditarily finite algebraic data types. The definitions of these data types and functions are the language-dependent part of the morphology. The language-independent part consists of an untyped dictionary format which is used for translation to other morphology formats and synthesis of word forms, and to generate a decorated trie, which is used for analysis.

Functional Morphology builds on ideas introduced by Huet in his computational linguistics toolkit Zen, which he has used to implement the morphology of Sanskrit. The goal has been to make it easy for linguists who are not trained as functional programmers, to apply the ideas to new languages. As a proof of the productivity of the method, morphologies for Swedish, Italian, Russian, Spanish, and Latin have already been implemented.