A Functional Language for the Specification of Complex Tree Transformations

Reinhold Heckmann

Universität des Saarlandes
D-6600 Saarbrücken, W. Germany

The implementation of the language TrafoLa was developed on a VAX 11/780 computer with operating system UNIX. It consists of two processes running together and linked to a circle by two UNIX pipes. The first process is the front end for TrafoLa, and the second one an ML interpreter acting as TrafoLa interpreter.

The front end is concerned with file handling and user input/output. It reads concrete TrafoLa text and translates it into abstract syntax trees that are submitted to the real interpreter as concrete ML text through the first pipe.

The ML interpreter evaluates this ML expression and converts the result into a string by unparsing, and sends it back to the front end that refines and displays it. Example:

```
TrafoLa input:   [1, 2] . [3]
ML expression sent to the ML interpreter:
  ip(valof(edot(eseq([enum 1 ,enum 2 ]),eseq([enum 3 ]))));
  where 'ip' is the function implementing the TrafoLa interpreter
  and its argument is the abstract form of the TrafoLa expression
  to be evaluated
Answer of the ML interpreter:   > "[1, 2, 3]" : string
Output of the whole system:     > [1, 2, 3]
```

The front end was mainly generated by a combined scanner, LL (2) parser, and tree generator generator from a specification of the concrete syntax of TrafoLa and of its translation into abstract syntax. To this generated code, some functions for converting the abstract trees into ML text, for user interaction and file handling were added. The front end is completely written in the programming language C.

The denotational semantics of TrafoLa was translated into an ML program and compiled into abstract ML machine code. When the TrafoLa interpreter is started, the ML interpreter is invoked with this code as argument, and is then able to evaluate ML expressions standing for TrafoLa terms.

The ML program consists of modules for

- the semantic values (sequences, trees, and functions) of TrafoLa and their operations (concatenation, insertion, application, etc.)
- environments implemented as sequences of pairs of variables and values, and their operations (look up, superposition etc.)
- sets of environments with union and superposition
- the abstract syntax of TrafoLa as ML data type
- the interpreting functions 'match' and 'eval'
- the initialization and update of the global environment

Both components of the TrafoLa interpreter are essentially generated: the front end from a specification of the concrete syntax of TrafoLa and its translation into abstract syntax, and the real interpreter from the denotational semantics of TrafoLa. This is advantageous for an experimental implementation since both syntax and semantics of TrafoLa may easily be updated or extended, but implies that the interpreter is quite slow.

We intend to design an abstract machine for TrafoLa and to compile it to code of this machine. The emphasis will lie on the patterns of TrafoLa since the expressions are similar to those in other functional languages.