HASKELL+ Program Transformation System

C. Lescher, M. Höff, R. Vater, A. Maletti,
A. Kühnemann, J. Voigtländer

January 13, 2003

email: voigt@tcs.inf.tu-dresden.de
letter: Janis Voigtländer
Department of Computer Science
Dresden University of Technology
01062 Dresden
GERMANY

1 Introduction

This program was developed to demonstrate the constructions presented in [VK01]. It may be used to automatically compose two macro tree transducers given certain restrictions. It uses an extension of HASKELL (namely HASKELL+) as specification language, but is able to output HASKELL code on demand. Furthermore it enables the user to apply some post-processing to the result and also allows for a direct comparison to classical deforestation, which is also implemented in the system.

2 Input syntax

Since HASKELL+ is an extension of HASKELL you can write standard HASKELL code as well as specify tree transducers using two new block constructs, namely the `data` block and the `mag` block.

2.1 The `data` block

Example:

```
begindata Data
  data Nat  = S Nat | Z
  data List = A List | B List | Nil
  data Tree = Node Tree Tree | Leaf
enddata
```

Obviously a data block is started with the keyword `begindata` followed by an identifier (block name) for that particular data block. Note that identifiers consists of letters solely and start with an uppercase character. Inside the block the syntax is very similar to the one of HASKELL. The keyword `data` is followed by another identifier (type name) and equals-sign (=) and a nonempty sequence of constructor declarations separated by bars (|). A constructor declaration consists of an identifier (constructor name) in front of type names (possibly none). Note that identifiers should be unique (i.e. an identifier should either be a type name or a constructor name). The whole block thus defines algebraic data types. The block is closed with the keyword `enddata`.

2.2 The `mag` block

Example:

```
beginmag Rev
  input Data
  syn rev :: List -> List -> List
  syn id  :: Nat -> Nat
```

```
    rev (A x) y = rev x (A y)
    rev (B x) y = rev x (B y)
    rev Nil   y = y

    id (S x) = S (id x)
    id Z     = Z
  endmag
```

The mag block uses a similar tagging technique, where the keywords are `beginmag` and `endmag`. The block name here is also the name of the tree transducer. Each mag block imports data type definitions using the keyword `input` at the beginning of the block followed by a data block name. Note that only one data block name is allowed. The next part of the mag block consists of state (function symbol) declarations using the keyword `syn` in front of a nonempty sequence of letters starting with a lowercase character. Optionally you separate the type of the function symbol by a double colon (::) sign from the function symbol. The type itself comprises of type names interleaved by arrow (−>) symbols.

The last part of a mag block contains the rewrite equations as defined in the accompanying paper (see introduction), where the arrow is replaced by the equals (=) sign.

3 Example session

Let's assume the following file `rev.hp`:

```
begindata Data
  data List = A List | B List | E
enddata

beginmag Rev
  input Data
  syn rev :: List -> List -> List

  rev (A u) y = rev u (A y)
  rev (B u) y = rev u (B y)
  rev E     y = y
endmag
```

also supplied with the examples.

The execution of the program will eventually yield the following menu:

```
(1) Just compose [nc-MAC with wsu-MAC; MAC with TOP; TOP with MAC]
(2) Compose [nc-MAC with wsu-MAC; MAC with TOP; TOP with MAC]
    and apply post-processing
(3) Apply deforestation

(0) Exit

Please select (0-3):
```

We select 2, input the filename without suffix (`rev`) and the names of the tree transducers to compose (in our case two times `Rev`). HASKELL output is according to the HASKELL language specification (found under *http://www.haskell.org/*).

The HASKELL+ output has some additional annotations.

```
beginmag Rev_Rev [Mac,Mat,Su,Wp,Wsu,Xlin,Xnd,Ylin]
```

The list behind the tree transducer name contains the detected properties of that particular tree transducer. Amongst the most important are the following:

`Top`	Top-down tree transducer
`Mac`	Macro tree transducer
`Wsu`	weakly single use
`Ylin`	non-copying
`Su`	single use

Note that the construction is only applied, if the input tree transducers meet the given requirements. Another unique feature is the presence of so-called Rules-pragmas. They are introduced in the Glasgow Haskell Compilation System (GHC) documentation of version 5 and later.

```
{-# RULES "REMOVE SUPERFLUOUS CONTEXT PARAMETERS" forall u y1 y2.
       rev_rev u y1 y2 = rev_rev' u y1
#-}
```

References

[VK01] J. Voigtländer and A. Kühnemann. Composition of functions with accumulating parameters. Technical Report TUD-FI01-08, Dresden University of Technology, August 2001.
http://wwwtcs.inf.tu-dresden.de/~voigt/TUD-FI01-08.ps.gz
Revised version to appear in *Journal of Functional Programming*.