In *Proc. 11th International Workshop on Functional and (Constraint) Logic Programming, WFLP 2002* (M. Falaschi, Ed.), Grado (Italy), pp. 225–228, 2002

CurryDoc: A Documentation Tool for Declarative Programs

Michael Hanus*

Institut für Informatik, Christian-Albrechts-Universität Kiel
D-24098 Kiel, Germany, `mh@informatik.uni-kiel.de`

Abstract. In this system demonstration we present CurryDoc, a tool for the automatic generation of documentation manuals in HTML format from programs written in the declarative multi-paradigm language Curry. The documentation is generated by combining comments in the source program with information extracted from the program. It extends other tools with a similar goal (e.g., `javadoc`, `lpdoc`) by the inclusion of information in the generated documents which has been computed by analyzing the structure and approximating the run-time behavior of the program. CurryDoc is completely implemented in Curry and is used to generate the documentation of the libraries included in PAKCS, a freely available implementation of Curry.

1 Overview

Curry [3, 8] is a declarative multi-paradigm programming language that combines in a seamless way the most important features from functional, logic, constraint, and concurrent programming paradigms. Curry has been used in a number of non-trivial applications, like GUI programming [4], web programming [5], implementing graphical programming environments [7], or partial evaluators [1]. As usual in application programming, the implementation of such systems require the use of many libraries to avoid programming everything from scratch. This demands for an adequate documentation of such libraries, which is the motivation for this work.

In order to produce up-to-date program documentation with a modest effort, it is preferable to generate such documentation automatically from source programs, as done, for instance, in the tools `javadoc`[1] or `lpdoc` [9]. The general idea of such tools is to put some documentation information into the source files (e.g., special comments in `javadoc`, or special directives or clauses in `lpdoc`). Then, the documentation tool extracts this documentation information and restructures it in a specific format (e.g., HTML pages with hyperlinks, man pages, info files). The CurryDoc tool follows a similar idea but extends this functionality by the inclusion of information that is often relevant to the programmer but not directly present in the program. Beyond type information (note that, like

* This research has been partially supported by the German Research Council (DFG) under grant Ha 2457/1-2, by the DAAD/NSF under grant INT-9981317, and by the DAAD under the programme Acciones Integradas Hispano-Alemanas.

[1] http://java.sun.com/j2se/javadoc/

in Haskell, function types need not be written in Curry programs but can be inferred by a type inferencer), this includes information about the overlapping of patterns in function definitions (which might cause unexpected nondeterministic computations), complete pattern matching (is there a matching rule for all ground calls?), solution completeness (are all non-ground calls to this function solvable in the sense of logic programming, or do some calls suspend?), indeterministic computations due to external communication or committed choice, etc. Since the computation of some of this information requires the global analysis of the program, the CurryDoc tool also includes a program analyzer.

In general, the CurryDoc tool generates the documentation for a Curry program (i.e., the main module and all its imported modules) in HTML format. The generated HTML pages contain information about all data types and functions exported by a module as well as links between the different entities, and the information about functions mentioned above combined with documentation comments provided by the programmer.

A *documentation comment* starts with "`---`" (note that standard comments start with "`--`") before the definition of the documented entity. The comments can also contain several special tags like:

`@author` - the author of a module

`@version` - the version of a module

`@cons` *id* - a comment for the constructor *id* of a datatype (in datatype comments)

`@param` *id* - a comment for a parameter *id* of a function (in function comments)

`@return` - a comment for the return value of a function (in function comments).

The following example shows a Curry program with documentation comments:

```
--- This is an example module.
--- @author Michael Hanus
--- @version 0.1
module example where
--- The function conc concatenates two lists. It is defined as
--- flexible so that it can also be used to split a given list.
--- @param xs - the first list
--- @param ys - the second list
--- @return a list containing all elements of xs and ys
conc eval flex
conc []     ys = ys
conc (x:xs) ys = x : conc xs ys
-- this comment will not be included in the documentation
--- The function last computes the last element of a given list.
--- @param xs - the given input list
--- @return last element of the input list
last xs | conc ys [x] =:= xs  = x   where x,ys free
--- This datatype defines polymorphic trees.
--- @cons Leaf - a leaf of the tree
--- @cons Node - an inner node of the tree
data Tree a = Leaf a | Node [Tree a]
```

The documentation for this module is generated by simply typing the command "`currydoc example`". This command puts the HTML documentation files for the module and all its imported modules into the directory `DOC_example` together with some index files (e.g., all functions or constructors). A part of the documentation generated by CurryDoc from this input program is shown in Fig. 1.

2 Implementation

CurryDoc is completely implemented in Curry, where the program analysis part is based on the library `Flat` for meta-programming in Curry [2] and partially reused from the programming environment CIDER [7]. Basically, CurryDoc processes the main module and all imported modules and produces the documentation for each single module. The module documentation is generated by reading the program source file to extract all documentation comments and by loading an intermediate representation of the program as data using the meta-programming library `Flat`. The latter representation is used to extract the lists of all exported data types and functions. The rules of the functions are analyzed in order to compute the information about the behavior of functions. Finally, this information is combined with the documentation comments to generate the corresponding HTML documentation files by the use of Curry's HTML library [5].

The implementation of CurryDoc is freely available and included in the latest distribution of PAKCS [6].

Acknowledgements. The author is grateful to Sergio Antoy for fruitful discussions and suggestions that led to the development of the CurryDoc tool.

References

1. E. Albert, M. Hanus, and G. Vidal. A Practical Partial Evaluator for a Multi-Paradigm Declarative Language. *Journal of Functional and Logic Programming*, Vol. 2002, No. 1, 2002.
2. FlatCurry: An intermediate representation for Curry programs. Available at http://www.informatik.uni-kiel.de/~curry/flat, 2001.
3. M. Hanus. A Unified Computation Model for Functional and Logic Programming. In *Proc. of the 24th ACM Symposium on Principles of Programming Languages (Paris)*, pp. 80–93, 1997.
4. M. Hanus. A Functional Logic Programming Approach to Graphical User Interfaces. In *International Workshop on Practical Aspects of Declarative Languages (PADL'00)*, pp. 47–62. Springer LNCS 1753, 2000.
5. M. Hanus. High-Level Server Side Web Scripting in Curry. In *Proc. of the Third International Symposium on Practical Aspects of Declarative Languages (PADL'01)*, pp. 76–92. Springer LNCS 1990, 2001.
6. M. Hanus, S. Antoy, J. Koj, P. Niederau, R. Sadre, and F. Steiner. PAKCS: The Portland Aachen Kiel Curry System. Available at http://www.informatik.uni-kiel.de/~pakcs/, 2002.
7. M. Hanus and J. Koj. An Integrated Development Environment for Declarative Multi-Paradigm Programming. In *Proc. of the International Workshop on Logic*

Programming Environments (WLPE'01), pp. 1–14, Paphos (Cyprus), 2001. Also available at http://arXiv.org/abs/cs.PL/0111039.
8. M. Hanus (ed.). Curry: An Integrated Functional Logic Language (Vers. 0.7). Available at http://www.informatik.uni-kiel.de/~curry, 2000.
9. M. Hermenegildo. A Documentation Generator for (C)LP Systems. In *Proc. of the 1st Int. Conf. on Computational Logic*, pp. 1345–1361. Springer LNAI 1861, 2000.

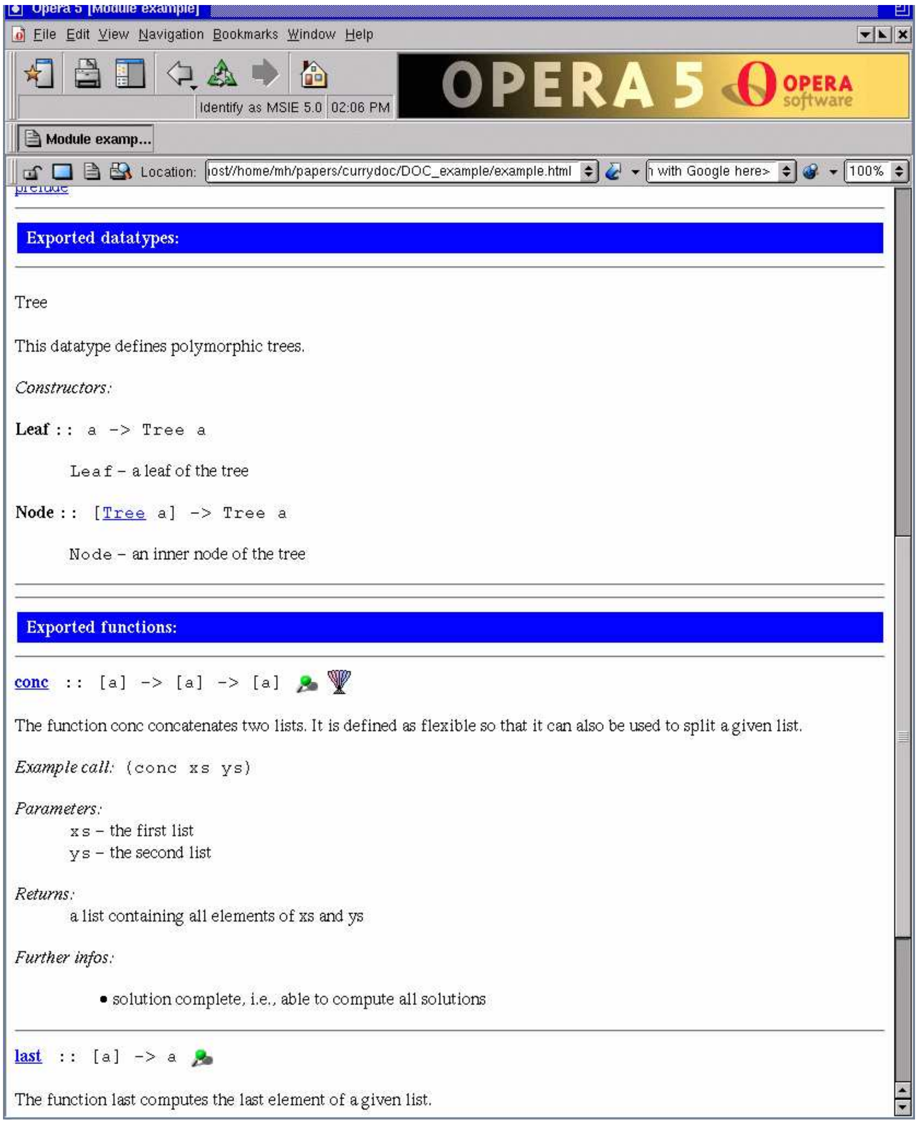

Fig. 1. Example for documentation generated by CurryDoc