PURe: Program Understanding and Re-engineering
(under FCT contract `POSI/ICHS/44304/2002`)

BIC 2: Research Plan on **Point-free Program Transformation**

Theme:	**Point-free Program Transformation**
Relevant Tasks:	1 and 6
Grant Holder:	*to be designated*
Start Date:	2005.01.01
End Date:	2005.06.30

1 Program Calculation in the Point-free Style

The calculational approach is particularly appealing in the area of functional programming, since referential transparency ensures that expressions in functional programs behave as ordinary expressions in mathematics. This leads to a program calculus – a collection of equational laws – that can be used to prove semantic equivalence between programs, or to derive efficient implementations from specifications.

In order to make this approach more effective, we should use a style of programming with special characteristics. First, we must avoid the use of arbitrary recursive definitions. Instead we should restrict ourselves to a limited set of higher-order functions that encode typical patterns of recursion, such as folds and unfolds, characterized by a useful set of properties. Second, we should use a *point-free* style of programming, where programs are built by composing simpler ones using a limited set of combinators – again characterized by a nice set of equational laws – without ever mentioning their arguments. Although the usefulness of recursion patterns is already widely recognized, the same can not be said about point-free programming. Even if some introductory books on functional programming already argue that this style leads to simpler reasoning methods, most authors still use the point-wise style, both for programming and reasoning. Due to the absence of variables in the point-free expressions, derivations correspond to reductions in a first-order rewriting system. The use of the point-free style of writing functional programs is particularly appealing when one is interested in deriving program transformations in a purely equational form. In the typical notation for presenting such derivations, the relevant equational law is annotated with each proof step:

$$\begin{array}{l} f_1 \\ \quad = \quad \{\text{by law} \ldots\} \\ f_2 \\ \quad = \quad \{\text{by law} \ldots\} \\ f_3 \\ QED \end{array}$$

2 Program Transformation with the MAG System

There have been some attempts to develop automatic systems for point-wise program transformation. One of the most successful is the MAG system developed at Oxford by Sittampalam and de Moor. This system is not fully automatic, but relies on the notion of *active source*, that is, the original (inefficient) definitions are stored together with sufficient hints (typically the creative steps of the derivation) that enable the system to derive the efficient version. It has been applied to perform transformations using, for example, the well known accumulation and tupling strategies.

At the core of this system lies a term rewriting mechanism that, given a set of transformation rules, tries to apply them from left to right in the order in which they appear in the active source, repeating this process until no rule can be applied. In order to cope with side-conditions with uninstantiated variables, this mechanism includes a higher-order matching algorithm that is capable of deriving mechanically new function definitions.

3 Research Objectives

To implement a tool for program transformation in the "point-free" style. The tool will manipulate programs written using the **Pointless Library** in the Uminho Haskell Libraries.

- Writing a parser for point-free programs;
- Implementing a conditional-rewriting system inspired by the MAG system, where each program is annotated with a sequence of transformation rules to be applied, in order. The novelty here is the use of a purely "point-free" style for the programs.

4 Integration in the PURe Project

In the context of the PURe project, a number of libraries and tools have already been developed allowing to manipulate point-free expressions; in particular

- *Pointless:* A library providing support for "point-free" programming, where recursion patterns are defined as hylomorphisms;
  (`http://wiki.di.uminho.pt/bin/view/Alcino/PointlessHaskell`)
- *DrHylo:* A tool for deriving hylomorphisms from (a restricted) Haskell syntax;
  (`http://wiki.di.uminho.pt/bin/view/Alcino/DrHylo`)
- A tool for converting programs from a restricted form of Haskell code with variables to "point-free" form (currently being developed).

This tool will continue this research line, providing PURe with a mechanism to automatically reason about point-free programs. This will be directly relevant to task 1, since it will provide the means to classify algorithms more precisely (by converting hylomorphisms derived with **DrHylo** into more specific recursion patterns), and to task 6, since it will also cover deforestation (it can be implemented in a conditional rewrite system by reversing the order of the fusion rule).

5 Deliverables

- *Tool:* A tool for program transformation as described above, to be integrated in the UMinho Haskell Libraries.
- *Report:* A comprehensive final research report documenting the results achieved.

6 Schedule

T1 : Acquaintance with the **Pointless Library** in the Uminho Haskell Libraries, and with the **MAG** program transformation system.
$\boxed{\text{Month 1}}$

T2 : Acquaintance with term rewriting techniques. Implementation of a simple, non-conditional, rewriting system.
$\boxed{\text{Months 2 to 3}}$

T3 : Implementation of the conditional features, and explore the possibility of using narrowing to solve unknowns.
$\boxed{\text{Months 4 to 5}}$

T3 : Extensive testing, development of transformation tactics, and writing of a final research report.
$\boxed{\text{Month 6}}$