# Parametricity

## A Property of Polymorphism

Rasmus Ejlers Møgelberg
Rasmus Lerchedahl Petersen
Lars Birkedal

IT-University of Copenhagen

## Polymorphism

Strict typing discipline has proven to be an efficient tool for developing structured programs

## Limitations

However, distinguishing programs with different types, the typing discipline would consider the following two programs different

$$\texttt{int}-\texttt{reverse} : \texttt{int}-\texttt{list} \to \texttt{int}-\texttt{list}$$
$$\texttt{string}-\texttt{reverse} : \texttt{string}-\texttt{list} \to \texttt{string}-\texttt{list}$$

although they both simply reverse a list, and thus perform essentially the same operation.

## Solution

Polymorphism allows one to define a general reverse function

$$\texttt{reverse} : \forall\alpha.\alpha-\texttt{list} \to \alpha-\texttt{list}$$

which yields the needed functions upon instantiation:

$$\texttt{int}-\texttt{reverse} = \texttt{reverse}(\texttt{int})$$
$$\texttt{string}-\texttt{reverse} = \texttt{reverse}(\texttt{string})$$

`reverse` is then called a *polymorphic* function.

POLYMORPHISM: *the quality or state of being able to assume different forms*

## Parametric Polymorphism -intuitively

Intuitively parametricity is the statement that

"Polymorphic functions behave the same on all type-instantiations"

For example, a function $t$ of type $\forall\alpha.\alpha \to \alpha$ can not instantiate to the identity on some types and to the successor function on the natural numbers. In fact, it can not instantiate to the sucessor function on natural numbers at all, since the successor function is specific to the natural numbers, and is not available on all other types. It is actually provable that the program $t$ can only be the polymorphic identity function.
Likewise, functions of type $\forall\alpha.\alpha-\texttt{list} \to \alpha-\texttt{list}$ (like `reverse`) can only manipulate the order of lists, throw away elements and duplicate elements. Thus, we may conclude on the behaviour of polymorphic functions simply from their types.

# Consequences

## Inductive Datatypes

Restricting the set of functions of a given polymorphic type, parametricity allows inductive datatypes to be defined through polymorphic types, as only functions that truly behave in the spirit of the datatype are admitted as members of the polymorphic type.
For instance, parametricity ensures that all functions of type

$$\sigma-\texttt{lists} = \forall\alpha.\alpha \to (\sigma \to \alpha \to \alpha) \to \alpha$$

behave as lists.
Other definable types:

$$\begin{aligned}\texttt{Natural Numbers} &= \forall\alpha.(\alpha\to\alpha)\to\alpha\to\alpha\\ \sigma-\texttt{trees} &= \forall\alpha.\alpha\to(\sigma\to\alpha\to\alpha\to\alpha)\to\alpha\\ \sigma\times\tau &= \forall\alpha.(\sigma\to\tau\to\alpha)\to\alpha\\ \sigma+\tau &= \forall\alpha.(\sigma\to\alpha)\to(\tau\to\alpha)\to\alpha\\ \texttt{Empty} &= \forall\alpha.\alpha\end{aligned}$$

## Data abstraction

Suppose programmer A is writing a program using a datatype implemented by programmer B. A will view this datatype as a type plus some operations on that type, and programmer B will implement a concrete type plus concrete operations.
Using parametricity, one can prove that the program written by A can only access the concrete implementation through the given operations, and thus one can prove that the result of the program given by A will be independent of the concrete implementation of the datatype given by B.

## Models

There is a correspondence between languages with polymorphism and certain mathematical structures called $\lambda_2$-fibrations modelling polymorphism. A good understanding of parametricity for languages with polymorphism should be based on a mathematical understanding of parametricity in these mathematical structures.
Many models of polymorphism are known, but for most of these it is unknown whether they are parametric. Providing models of parametric polymorphisms in combination with many of the features of modern programming languages is yet to be done, and will provide mathematical foundation for applying the consequences above, such as data abstraction to reasoning principles for programming languages.

## On-going research at ITU

Aims of our work:

1. Give a precise definition of parametricity for $\lambda_2$-fibrations.
2. Provide domain-theoretic models of parametric polymmorphism, since mixing domain-theory with parametricity provides powerful principles of reasoning enabling the solution of recursive domain-equations. This is also important because Turing complete programming languages have fixpoints.
3. Study these models.
4. Apply this knowledge to obtain reasoning principles for programming languages.

## References

[1] R. Møgelberg and L. Birkedal. Categorical models of parametric polymorphism. 2004. `www.itu.dk/people/mogel`

[2] J.C. Reynolds. Types, abstraction, and parametric polymorphism. *Information Processing*, 83:513–523, 1983.

[3] P. Wadler. Theorems for free! In *4'th Symposium on Functional Programming Languages and Computer Architecture, ACM, London*, September 1989.