# Glinda: A Meta-circular Interpreter for Oz

Martin Henz and Michael Mehl

*Abstract*— **Executing computer programs means** *interpreting* **the instructions coded in a programming language. Most implementations of high-level languages, such as DFKI Oz, use an intermediate step of compilation: The source code is compiled to a machine code which is then interpreted by the hardware or a more abstract machine. We show the advantages and problems that occur when one tries to directly interprete Oz source code. We present an implementation of a meta-circular Oz interpreter, i.e. an interpreter for Oz written in Oz and demonstrate its application to source-level profiling, execution visualization and language design.**

*Keywords*— **Oz, interpreter, programming environments, profiling**

## 1. Introduction

Interpreters are computer programs that execute a suitable representation of a source program. Examples of interpreters are (1) every computer itself since it executes source programs in form of binary code, and (2) a BASIC interpreter that executes a given BASIC program line-by-line. Meta-circular interpreters of a given language are interpreters in which source and implementation language coincide.

During the development of concurrent logic programming languages, interpreters were often used to judge the quality of the language design [2]. The following questions were used as criteria:

- Does a proposed language have a simple meta-interpreter? A positive answer is usually considered a plus for a given language design.
- Does a simple interpreter for programs of a language A written in language B exist? A positive answer is usually considered as an indication that language A is at least as expressive as B.

Interpreters are used for language implementation since they typically require less programming effort than compilers. Other applications include program debuggers and profilers. For example, Prolog debuggers are often based on a meta-cirular interpreter for Prolog.

In this work, we present *Glinda*[1], a simple meta-circular interpreter for Oz [3], a concurrent language, providing for higher-order functional, object-oriented and constraint programming. We show how *Glinda* can be modified to serve as a program profiler and visualizer.

## 2. Current Implementation

The core of *Glinda* interpretes a ground-term representation of Kernel Oz programs. *Glinda*'s user interface allows to enter full Oz programs, which are automatically translated to Kernel Oz for interpretation. *Glinda* allows to call Oz builtins and compiled procedures from interpreted programs. This is useful for interpreting only certain modules of a program and compiling others to run at full speed.

The static scoping rules of Oz are implemented in *Glinda* by interpreting every construct with respect to an environment, in which bindings of variable names to Oz variables are stored. Every variable declaration extends the environment. The interpreter uses the extended environment for interpreting the program fragment in the scope of the declaration.

All interpreted procedures are unary so that procedures can be constructed and applied regardless of their arity.

Conditionals with $n$ parallel clauses are implemented by parallel conditionals with two clauses. This becomes possible by using deep guards and passing around the bodies of the conditionals in form of program code.

The source code of *Glinda* is available through WWW under `http://ps-www.dfki.uni-sb.de/~henz/oz/interpreter.html`.

[1]the good witch in [1]; we imagine *Glinda* to be beautiful, but somewhat slow.

Martin Henz and Michael Mehl are researchers in the German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D–66123 Saarbrücken, Germany, email:`{henz,mehl}@dfki.uni-sb.de`

## 3. Applications

Every language construct in Kernel Oz is interpreted by a particular procedure in *Glinda*. Manipulating this procedure thus allows to manipulate the semantics of the language construct.

As an example consider a profiler based on *Glinda*. The profiler should show how often every procedure in a given program is executed at run-time. We only have to manipulate the procedure of *Glinda* that is responsible for procedure application, such that applications are executed as usual but additionally notify a profiler object. To obtain a profiler, this profiler object only has to keep track of the number of applications of each procedures and provide this information to the user.

Another application is visualization and tracing of Oz programs. To this aim, we augment *Glinda* with code that incrementally builds a data structure while it interpretes code. This data structure can be visualized (e.g. by the Oz Browser) to obtain a run-time trace of the program. Furthermore, we can manipulate the interpretation of certain constructs so that a user interaction is necessary to continue the interpretation. This allows the user to interactively step through the program and watch the resulting computation.

Other applications include experimentation with alternative language semantics and with program transformations.

## 4. Limitations and Problems

*Glinda* is not very fast and it is not clear whether it is useful as a tool for debugging programs of realistic size.

The implementation of conditionals and disjunctions using deep guards imposes several limitations. One limitation is, that the interpreter cannot be implemented as an object and therefore the different extensions of the interpreter can not be implemented using inheritance. Another one is that it is not possible to communicate to toplevel objects, e.g. the profiler or the visualizer, from within guards.

The creation of new threads, due to suspension of conditionals or disjunctions is currently not observable and needs further investigation.

## References


[1] L. F. Baum. *The Wonderful Whizard of Oz.* G. M. Hill, 1900.

[2] E. Shapiro, editor. *Concurrent Prolog. Collected Papers. Volume 1 and 2.* The MIT Press, 1987.

[3] G. Smolka and R. T. (ed.). DFKI Oz documentation series. Available on paper or via WWW from `http://ps-www.dfki.uni-sb.de/oz/`, Deutsches Forschungszentrum für Künstliche Intelligenz, Stuhlsatzenhausweg 3, D–66123 Saarbrücken, Germany, 1994.