# Teyjus: A λProlog Implementation

The name *Teyjus* stands ambiguously for a project and a family of systems that are about providing an efficient and robust implementation of the language *λProlog*. As a project, *Teyjus* has spanned a period of over fifteen years, involving people from Duke University, SUNY Buffalo, the University of Chicago and the University of Minnesota. As a system, *Teyjus* has seen two incarnations. Its first version, released in mid-1999, is characterized by a complete support for the operation of higher-order unification. Its second version, released in April 2008, is oriented around a special form of higher-order unification known as pattern unification. The new version also includes modularity notions coupled with a complete form of separate compilation.

The work on *Teyjus* fits into a larger research theme for the group at the University of Minnesota: developing logics and systems for specifying, prototyping and reasoning about computations. In addition to *Teyjus*, the group has recently played a role in developing systems such as *Bedwyr* and *Abella* that can be used to reason about specifications written in a *λProlog*-like language. Some of this work represents an ongoing collaboration with the *Parsifal* group at INRIA, Saclay under the aegis of the SLIMMER project. We limit our attention here to the *Teyjus* system that concerns only the specification and prototyping aspects. A recent article in the newsletter on the *Abella* system has discussed the reasoning focus and the group members can also be contacted directly for more details about this direction.

**The Language λProlog.** Dale Miller and Gopalan Nadathur led an effort in the late 1980s to provide a broad proof theoretic foundation for logic programming languages. This effort resulted in the identification of the intuitionistic theory of higher-order hereditary Harrop formulas upon which λProlog is based. This class of formulas enriches that of Horn clauses—the logical basis of *Prolog*—with the possibilities of quantifying over function and (certain occurrences of) predicate variables, of explicitly representing binding in terms and of using a fuller complement of connectives and quantifiers. By systematically exploiting the new features in the underlying logic, *λProlog* provides support at the programming level for capabilities such as higher-order programming, scoping over names and procedures, modular programming, abstract data types and the use of λ-terms as data structures.

A central contribution of *λProlog* is that it has revealed a dimension to higher-order capabilities in programming that was not known before and that is not easily replicated in contexts different from logic programming. These capabilities derive from the availability of λ-terms for representing objects and of a suitable set of primitives for manipulating such representations. Using these features, *λProlog* pioneered support for the notion of *higher-order abstract syntax*, a profitable way to view the syntax of objects whose structures involve binding. Several detailed studies have, in fact, indicated the utility of this language in building symbolic systems that manipulate objects such as formulas, programs, proofs and types. Space limitations prevent us from discussing these metalanguage capabilities in any detail here. However, the *Teyjus* distribution contains several example programs illustrating such possibilities and also provides pointers to papers that discuss the conceptual aspects

of the code that is presented.

**The λProlog Abstract Machine.** The *Teyjus* implementation is built around a virtual machine that is capable of realizing the operations that arise in typical *λProlog* programs efficiently. The well-known Warren Abstract Machine (WAM) provides a structure for treating the unification and backtracking operations that are central to the evaluation machinery of this language. However, there are additional features that must also be accommodated:

- The language allows the name and code space to be dynamically altered and this has to be reflected into unification and procedure invocation.
- Representations have to be devised for λ-terms that allow them to be used efficiently in encoding objects.
- A richer form of unification, known as higher-order unification, has to be supported.
- Types impact on the dynamic behaviour of programs and a method has to be found to reduce their run-time footprint while still satisfying this role.

The first version of *Teyjus* was based on an extension to the WAM designed in the 1990s that addressed all these issues. Much was learned from experimenting with that system that has led to a substantial redesign of the virtual machine. The most significant change was a decision to orient computation around a deterministic and terminating fragment of higher-order unification known as pattern unification. This choice considerably simplifies the internal structure of the machine and its instruction set, and also obviates almost all runtime computations over types. These aspects are explored in detail in the (forthcoming) doctoral thesis of Xiaochu Qi. The machine designed by Xiaochu also incorporates the insights gained from recent studies by the group into the practical consequences of choices in λ-term representation.

**The Structure of the Teyjus System.** The actual implementation uses an emulator for the virtual machine. Thus, there are two main components to the system: a compiler and an emulator. The function of the compiler is to process a given module of *λProlog* code, to certify its internal consistency and to ensure that it satisfies a promise determined by an associated signature and, finally, to translate it into a byte-code form. This byte-code form consists of a "header" part containing constant and type names and other related data structures and a sequence of instructions that can be run on the abstract machine once it has understood the header information. One part of the emulator is a "loader" that can read in such byte-code files and put the emulator in a state where it is ready to respond to user queries. The other part of the emulator is, of course, a byte-code interpreter that steps through instructions in the manner called for by the user input.

The *λProlog* language contains also modularity features that allow large systems to be constructed by composing smaller modules. In contrast to its first version, the new *Teyjus* system provides complete support for separate compilation relative to this language. What this means is that the compiler must process each module separately from any other and

must generate code in such a way that it can be later combined with the compiled forms of other relevant modules to build the image that is eventually to be run. This extra "composition" information that is generated by the compiler is actually another part of the header information in byte-code files. The task of composing these files is taken up by a third component of the *Teyjus* system, the linker.

The new *Teyjus* system has been implemented via a mix of OCaml and C code. The desire was to benefit from the high-level and declarative aspects of a functional language wherever possible and to use C only where proximity to the underlying machine is needed for efficiency, an attribute that applies mainly to the emulator. Portability has also been a major design consideration. The system runs on varied 32 and 64 bit architectures and under all the major operating systems.

**Ongoing Research and Development.** The group is involved in several software projects related to the *Teyjus* system. One important, missing component is a garbage collection scheme. This is currently being developed. Another interesting direction of inquiry is the potential for the *Teyjus* virtual machine to serve as a general compilation target for higher-order logic based languages. A specific possibility that is being investigated in this connection is that of compiling logic programming specifications in the Twelf language to a form that can be run on this machine. Another project that is of interest is that of developing an interoperability interface between *λProlog* and C. At the very least, such an interface will allow for the flexible use of a large variety of C library functions in enhancing the capabilities of the *λProlog* system. In a longer term, we are interested in closely integrating the specification and prototyping capabilities that *Teyjus* provides with the reasoning capabilities of other systems like *Abella* that the group is now building. Part of the desire here is to allow for an easy transition between these capabilities that ultimately represent different uses of the same specifications. In a more ambitious direction there may be benefits to be derived, for example, from using computations directly as a means for reasoning.

**People Involved.** The new *Teyjus* system is the result of a collaboration between Andrew Gacek, Steven Holte, Gopalan Nadathur, Xiaochu Qi and Zach Snow. Colin DeVilbiss is currently working on the missing garbage collector. The present effort has benefited considerably from the earlier version of the system to which contributions were made by Bharat Jayaraman, Keehang Kwon, Chuck Liang and Dustin Mitchell. Our collaborators on the SLIMMER project from INRIA, Saclay have also provided valuable input in recent years; David Baelde and Dale Miller deserve special mention in this regard.

**Web Site.** The link to the Teyjus web site is `http://teyjus.cs.umn.edu/`. You may contact Gopalan Nadathur for any details not found through the web site.

**Acknowledgements.** Work on the new version of the *Teyjus* system has been supported by the National Science Foundation Grant CCF-0429572. This work has relied on ideas developed earlier based on support from a series of grants also from the National Science Foundation: Grants CCR-8905825, CCR-9208465, CCR-9596119 and CCR-9803849. Opinions, findings and conclusions or recommendations that are manifest in this material are those of the project participants and do not necessarily reflect the views of the NSF.