https://dtai.cs.kuleuven.be/projects/ALP/newsletter/nov04/nav/articles/szeredi/szeredi.html

The Early Days of Prolog in Hungary - a personal account

Peter Szeredi
Budapest University of Technology and Economics
http://www.cs.bme.hu/~szeredi/english

Editor: Enrico Pontelli

Below is a somewhat lengthy letter written around 1992 to Peter Van Roy, when he asked me for material for his JLP paper on Prolog implementations [PVR1994].

Background

In early seventies there was no specialised computer science education at the Hungarian universities. The demand for computer specialists was mostly covered by employing people who graduated in mathematics or engineering.

I graduated in 1972 as a mathematician, my thesis was in mathematical logic. I had also been programming computers for six years by that time.

My first encounter with computers and programming was in 1966 when I was 17. I was quite lucky to find a place, the Computer Center of the Ministry of Heavy Industries (NIM IGÜSZI), where I got a friendly reception and given access to computers. I worked there part-time during my studies and had done quite a bit of hacking in systems programming. I also was active in the Algol 68 group: we first translated the Algol 68 report to Hungarian and then started to work on an Algol 68 implementation (which has never been finished). [A sidetrack: during implementation I managed to discover an inconsistency in the type (mode) system of Algol 68 which was later named "incestuous unions".]

I joined (full time) the software department at NIM IGÜSZI in 1972. I worked in a group producing various system programs for a new Hungarian computer. For our development work we used a relatively unknown language CDL (Compiler Definition Language) designed by C. H. A. Koster of Nijmegen University. We got acquainted with Koster through our work on Algol 68 as he was one of the authors of the Algol 68 report. Koster invented a variant of van Wijngaarden's two level grammars called affix grammars, and designed a language based on this type of grammar, which was CDL.

Here is a piece of code in CDL:

```
factorial+n+f-n1:
   equal+n+1, make+f+1;
   subtr+n+1+n1, factorial+n1+f, times+f+n+f.
```

If you replace the affixes (+id) with parenthesised arguments, delete the local variable declaration (-n1) and replace the ':' with ':-' you get exactly Edinburgh Prolog notation (all this in 1970/71, some two years before Prolog and six years before the Edinburgh notation was invented). Of course, CDL is an imperative language, with destructive assignment, and also there is no backtracking, but otherwise the notation and the control is surprisingly similar to that of Prolog.

CDL is actually an open ended language, i.e. it lacks any primitive operations. These (such as the equal, make, subtr, ... operations above) have to be supplied in the target language (most of the time the assembly language of the target computer, or sometimes a higher level language, such as C). CDL is thus more like a recursive macro language.

In early seventies computerisation was one of the buzzwords in Hungary, and there was quite a bit of government money for R&D work in computer software. From this money another group at NIM IGÜSZI (the "Logic in Computer Science" group), led by István Németi, got a grant for developing an interactive, semi-automatic verification system for a simple algorithmic language. The verification system contained a structure sharing Boyer-Moore type theorem prover. We managed to convince Németi's group to use CDL as the implementation language. Because of this, and also because of my earlier interest in logic I myself got interested in this work. I remember looking at the theorem prover working on small theorems in algebra, and realising that it wasn't able to go much beyond toy problems.

Besides program verification, Németi's group was interested in all kinds of interactions between logic and computer science. A particular goal of theirs was to study and develop logic based declarative programming languages such as Prolog. They had seminars on declarative programming languages (e.g. on logic programming) during 1973 and they corresponded with the department of Computational Logic in Edinburgh which led to Németi's visiting Edinburgh in 1974.

The first Hungarian Prolog

Németi had research contacts in Edinburgh and visited Scotland several times. From there he brought the news and some papers about a theorem prover (?) called Prolog. I was quite taken aback by the fact that there was a theorem prover that could do something useful: calculate the factorial of a number! Németi brought the Marseille interpreter as a deck of cards from Edinburgh together with a brief description of built-ins (on a line printer listing) entitled Epilog[400,400] and copies of slides of David Warren's talk entitled "What is Prolog?".

Porting of the Marseille Fortran interpreter to the ICL 1903A computer of NIM IGÜSZI was included into our research contract for 1975. A member of Németi's group, Péter Tóth, who was given this task, encountered unexpected problems due to the different word and character size (ICL 1900 had 24 bit words and 6 bit characters). While he was struggling with the porting, I decided to try to write my own Prolog interpreter in CDL. I actually had never had a look at the Marseille Prolog itself, I based the implementation on the last three slides of David Warren's above mentioned talk, entitled 'Example to illustrate how Prolog is implemented'. This showed three snapshots in the execution of the (naive) rev(a.b.X, a.b.X) goal. The structure of the (single) stack was quite apparent from these slides, and this, together with my earlier encounter with a structure sharing unification algorithm was enough to get me started. CDL proved to be a very convenient implementation tool, especially suited for parsing, so the tokeniser and parser were written in CDL, rather than in Prolog (as was the case for the Marseille interpreter).

I succeeded in producing a working Prolog interpreter in a few weeks, just in time for the usual yearly out of town gathering of NIM IGÜSZI software developers where Prolog was the main topic that year. The availability of a working interpreter gave a big boost to that meeting. Several people got rather excited about Prolog, including Ferenc Darvas who worked on projects for drug-industry, Iván Futó, who worked in simulation, Zsuzsa Márkusz, an architect, Miklós Szõts, a civil engineer turned computer scientist. (Both Zsuzsa Márkusz and Miklós Szõts were students of Németi.) In a few weeks after the meeting Miklós Szõts produced the very first

application of the Hungarian Prolog: a program for planning of a one level workshop building using prefabricated panels. Ferenc Darvas also came back with a number of application proposals, where he believed Prolog to be of good use, such as predicting drug interactions and various other programs in drug design.

Very active months and years followed, with various applications projects going on, which demanded various language extensions, ports to other computers, debugging facilities, etc. Through this fruitful interaction between implementors and application developers the Prolog interpreter grew from a toy to a usable system.

This system (which is often confused with MProlog, the second Hungarian Prolog, see later) had no name in itself. It used the Marseille syntax, but the built-ins had English names (although some, which could pass as English, at least for me, remained the same as in the Marseille Prolog, e.g. univ, egalf, etc.). Here is an example:

```
+FACT(0,1).
+FACT(*N,*F) -LESS(0,*N) -MINUS(*N,1,*N1) -FACT(*N1,*F1) -
TIMES(*F1,*N,*F).
```

It was a batch-oriented system, as most of the computers in Hungary at that time were mainframes without interactive access. It provided a selective exhaustive trace and backtrace services. It had a number of new built-ins over the Marseille functionality, such as undoable input and undoable database predicates.

Undoable input was implemented using a a circular buffer in the tokeniser to store the tokens read in. Token and term input predicates trailed the value of the current token pointer of this circular input buffer before the actual input was made. If such an input predicate was backtracked over, its side effect was undone by moving the current token pointer back to its original position (of course, this backtracking capability was limited by the size of the buffer). Subsequent input predicates read the tokens from the buffer, and switched back to ``real'' input only when the sequence of unread tokens was exhausted.

The second half of seventies there was thus a boom for Prolog in Hungary. Several pilot application projects were started, mostly with the help of government grants, though. The Prolog was ported to the Siemens BS2000 computer, the first widely used interactive implementation.

[You asked for anecdotes, so here is one. One of the first, very popular, interactive demo programs (written by my younger brother János) was a Hungarian natural language question-answer system, which could parse and remember simple facts and rules about people and itself (the program) and answer questions about these. It rejected statements from which one could deduce some negative fact about the program itself. E.g. after the rule 'If one is nice, he is stupid' had been entered, it refused to accept the statement 'You are nice', and actually gave an explanation why. One night a big shout was heard at the terminal room, where two students were playing with this program: I managed to tell the computer that he is stupid - shouted one of them - simply by spelling it stoopid (it is even easier to misspell the Hungarian equivalent of stupid).]

The Hungarian Prolog arouse the interest of LP researchers fairly soon. I've dug up a letter from David Warren dated February 1976, and from Bob Kowalski dated July 1976 in which they inquire about our Prolog. Bob Kowalski actually visited us for a few days on his way back from vacation in Poland in the summer of 1976. David Warren came in June 1977 for a two-week

visit. [It turned out at that time that David was influenced in his design of the Edinburgh syntax by his earlier work in Algol 68, thus explaining similarities with CDL.] In 1978 there was symposium on Logic in Programming in Salgótarján, with several logic programmers present. In 1980 the Logic Programming Workshop was held in Debrecen.

Unfortunately by the end of seventies the steam started to run out for Prolog applications in Hungary. The biggest problem was the price of computer time. To run Prolog one needed to purchase computer time on one of the big western made mainframes, and this was much more expensive than employing people to do the same task by hand (the labour price being much more cheap than in the West). Thus in Hungary, in late seventies, natural intelligence proved to be much cheaper than the artificial one.

MProlog - the second Hungarian Prolog

Catching the last wave of big government grants for software, the development of a new Prolog implementation was started in 1978. This implementation, later called MProlog, was based on the three stack Warren model of 1977. It was written in CDL2, a successor to CDL.

The MProlog system offered two paths for program execution. During development, the Program Development SubSystem (PDSS) provided an interactive environment for writing and testing programs. For application delivery the Production System was offered, which used the traditional translate-link-execute model.

PDSS was a comprehensive environment for the development of MProlog programs providing support for program entry, editing, execution, tracing, saving, etc. PDSS was implemented in MProlog itself. For the programs entered, the system retained the source (without layout), including variable names. PDSS stored the program as a syntax-tree, which was always presented to the user in a standard, pretty-printed format.

In the Production System MProlog source modules were first converted into a compact internal format using the pretranslator. The translation process included syntactic and semantic analysis (such as variable classification), and produced a binary module containing Prolog code in a format suitable for interpretation or further compilation. Pretranslated modules could be linked---in MProlog terminology, consolidated---into an executable program by the consolidator program. The Consolidator also had the capability of consolidating several binary modules into a single new binary module. MProlog also has a native code compiler for certain architectures (IBM 370, M68000, VAX, Intel 386), as well as a byte-code compiler. The compiler transformed a pretranslated module to a compiled module, with the resulting module also amenable to consolidation. The compiler encompassed a number of important improvements over the original structure sharing compiler model, including techniques similar to those used in the WAM.

The binary program resulting from consolidation was executed by the interpreter. Unlike some other Prolog implementations, in MProlog the interpreter was the core of the system. It was written in CDL2 and in addition to interpretation proper, it served as the run-time system (i.e. memory management, etc.) for the compiled code.

MProlog extended the usual Edinburgh Prolog language in a number of respects. It had a (name based) module concept, user-controlled multi-level indexing, exception handling, a rich set of built-ins, including the undoable ones inherited from the first Hungarian Prolog. For more details on MProlog, see [Far1994].

Although the development of MProlog was started at NIM IGÜSZI, most of the people involved gradually moved to SZKI, a big computer company set up a few years before. I myself moved to the Theoretical Laboratory, the department responsible for advanced research within SZKI in 1979. This Lab was led by Bálint Dömölki, who helped a lot in keeping the logic programming R&D alive. In SZKI I met Péter Köves and Zsuzsa Farkas, who, in the following years were leading the development of various sub-projects for MProlog. All in all, SZKI provided an environment much more appreciative than NIM IGÜSZI for the results achieved.

MProlog was being developed on the Siemens machine of SZKI. It was demonstrated at the Debrecen workshop in June 1980, using a terminal connected to the mainframe in Budapest. The Hungarian-made terminal was very awkward to handle, one had to enter each message off-line, surroundedwith special control characters showing which part of the screen to transmit, and then press a special 'send' button to do the actual transmission. As I recall, only Lew Baxter from Canada was brave enough to tackle this terminal, successfully running his version of the who-owns-the-zebra (five houses) puzzle. Strangely enough, four years later he was recruited as the main Prolog expert for Logicware, the company set up in Toronto to market MProlog in North America.

The announcement of the Japanese Fifth Generation Project gave a big boost to the MProlog work. It was decided that MProlog will be made into a product marketed in the West. A user documentation was produced in English. One of the problems characteristic of the contemporary Hungarian environment was that it was very difficult to find a way of printing it out on a printer capable of producing lower case letters.

MProlog was ported to IBM CMS in 1981 and to VAX VMS in 1982. A problem with the latter port was that officially there were no VAXes in Hungary due to COCOM restrictions. In reality there were a few ones smuggled in, but they were very difficult to access. The initial, partly successful port to VAX was done at Nijmegen. Later we managed to get some limited access to a VAX in Hungary. Epsilon GmbH in West Berlin, formed from the developers of CDL2 also helped a lot by porting MProlog to computers not available in Hungary.

The first sale of MProlog was in September 1982, in France. In the subsequent years a number of European sales followed, in part through Epsilon. In 1983 a Japanese distributor was appointed and, after a version supporting Japanese characters was developed, several copies of the system were sold.

In early 1984 a Hungarian born Canadian entrepreneur established a company named Logicware to market MProlog in North America. The main goal of Logicware was to enter Prolog into the data processing departments of big companies. MProlog had the advantage of being available on mainframes as well as minicomputers, workstations and PCs. Logicware management decided that the biggest obstacle in widespread acceptance of (M)Prolog was the inappropriateness of the terminology and notation, as well as the lack of educational tools. Therefore big efforts went into redesigning the terminology and the language. For example the term 'clause' was replaced by the term 'statement', 'predicate' by 'definition', 'term' by 'expression', the new terms being thought to be more acceptable for data processing people. The built-in predicates were re-structured, and the names changed according to the new terminology. E.g. the predicate 'clause' became 'matching_statement' where the word 'matching' indicated that the built-in enumerated its multiple solutions on backtracking.

A group previously working on CDL2 itself, led by Tamás Langer joined SZKI late 1983. In its

height there were about 15 people working full time on MProlog.

We switched from using the CDL2 compiler to the CDL2 Lab, a sophisticated program development environment. The Lab, with its powerful inter-module code-generation scheme, helped a lot in efficient porting of MProlog to a wide range of architectures.

A new version of the compiler was designed and implemented. The first version of the compiler, which was a fairly straightforward adaptation of the Warren 1977 compiler was not really successful, as the code was only 5-7 times faster than the (fast) interpreted one (and speedups were even worse if lots of built-ins were invoked). The new compiler had various features taken over from the WAM as well as further extensions, some of which are described in a paper in the proceedings of the workshop on Implementations of Logic Programming Systems of ICLP93.

A special 16-bit version of MProlog was designed and implemented on IBM PCs. With the appearance of 386 processors a 32-bit version was also ported to the PC. Further ports included workstations (e.g. SUN), Macintosh, IBM MVS, etc.

Significant effort was spent on tuning MProlog. The most often used code in the interpreter was rewritten to assembly language of the target computer, resulting in a fairly fast interpreter. In mid 1980-s even a special piece of hardware was designed and prototyped, to support efficient execution of MProlog (but this experiment did not go any further than the prototype).

The PDSS went through several stages of development gradually achieving the functionality outlined above. Various extensions, user interface tools were also developed.

A tutorial supported by an appropriate educational software tool was also produced.

In addition to MProlog development work, efforts were made to produce libraries, tools, and concrete applications of MProlog.

Unfortunately the business plans of Logicware didn't materialise, and the company gradually went out of business. SZKI continued the maintenance and support of MProlog with the help of European distributors in Germany and Italy.

In 1990 a new company, named IQSOFT, has been formed from the Theoretical Lab of SZKI. IQSOFT maintained and used MProlog in early 1990-s, but mostly for internal purposes.

Other Hungarian Prolog implementations

Iván Futó and his group developed an extension of the first Hungarian Prolog, called T-Prolog, for discrete simulation purposes. One of the interesting features of T-Prolog was its coroutining control, which was initially implemented using (double) interpretation. In MProlog special built-ins for handling continuations were included, and a T-Prolog implementation avoiding double interpretation was constructed. In mid-eighties an independent Prolog implementation, called CS-Prolog, was developed by Futó's group.

The line of T-Prolog, CS-Prolog etc. languages is described in detail in Futó's invited talk at ICLP93 [Fut1993].

References

[Far1994] Zs. Farkas., P. Köves, P. Szeredi: MProlog: an Implementation Overview. In Evan Tick and Giancarlo Succi (editors): Implementations of Logic Programming Systems. Kluwer Academic Publishers, 1994, 103-117

[Fut1993] I. Futó: Prolog with Communicating Processes: From T-Prolog to CSR-Prolog (Invited paper), In: D. S. Warren (Ed.): Logic Programming, Proceedings of the Tenth International Conference on Logic Programming, MIT Press, 3-17.

[PVR1994] P. Van Roy: 1983-1993: The Wonder Years of Sequential Prolog Implementation. Journal of Logic Programming 19/20: 385-441 (1994)