Teaching Logic Programming at the Budapest University of Technology

Péter Szeredi

szeredi@cs.bme.hu

Department of Computer Science and Information Theory,
Budapest University of Technology and Economics
H-1117 Budapest, Magyar tudósok körútja 2.

Abstract

The paper describes courses related to Logic Programming at the Budapest University of Technology and Economics. We present the layout and the contents of the main such course, entitled Declarative Programming, as well as the tools used in teaching this subject. We then give a brief outline of the elective courses and other educational activities in the subject area.

1 Introduction

Logic Programming and the Prolog language has been taught at several Hungarian universities since the mid 1970's. I have been involved in teaching LP from the very beginning, first at the Eötvös Loránd University, Budapest, and later at the Budapest University of Technology and Economics (BUTE).

The paper gives an overview of courses related to Logic Programming at the Faculty of Electrical Engineering and Informatics at BUTE. These courses are primarily intended for undergraduate students of informatics, although students of mathematics and electrical engineering sometimes also take these courses.[1]

In Section 2 the declarative programming course is discussed in detail. Section 3 describes the elective courses related to LP. Note that details of one of these courses, on constraint programming, have recently been published in [11]. Section 4 discusses educational activities other than courses, such as student projects, student conference papers, as well as theses related to LP. The paper is concluded with a brief summary discussion.

[1] Note that at present the undergraduate studies at BUTE span 5 years and lead to an MSc degree.

2 The Declarative Programming course

Declarative programming is part of the compulsory studies for students of informatics at BUTE. The course is scheduled in the fourth semester, i.e. the spring semester of the second year, although the students are allowed to take the course earlier or later. It is preceded by programming language courses on C, C++, and Java, as well as by a course on mathematical logic, which introduces, among others, the theory of logic programming.

The Declarative Programming (DP) course covers the two main declarative paradigms. Péter Hanák teaches functional programming, using the MOSML dialect of the SML language, while Péter Szeredi is responsible for the logic programming part, which focuses on Prolog and uses the SICStus Prolog implementation [9].

The course in this setup has been taught in every semester since 1994. It evolved from a wider subject entitled "Programming Paradigms", which involved imperative, object-oriented and functional programming styles.[2]

The semester normally consists of 14 weeks. The DP course has two lectures of 2*45 minutes each week. The structure of the course is the following:

Lecture 1:	Introducing the declarative programming paradigm
Lectures 2-8:	Logic Programming, part 1
Lectures 9-15:	Functional Programming, part 1
Lectures 16-21:	Logic Programming, part 2
Lectures 22-27:	Functional Programming, part 2
Lecture 28:	Summary, outlook

Very unfortunately, the DP course has no laboratory exercises. In order to encourage students to do some programming tasks we hand out during the course several minor assignments as well as a single major assignment (see 2.2). We have also developed a computer tool, called ETS (Electronic Teaching aSsistant, or Elektronikus TanárSegéd in Hungarian) [3], to support Web-based student exercising, assignment submission/evaluation, marking, etc.

Although there is a clear division of the LP and FP parts of the course, these build on each other. We reuse concepts and techniques introduced in the other part, e.g. those of tail recursion, accumulators, construction and decomposition of data structures using pattern matching (unification). The two parts are also linked by the common major assignment, and occasionally by common minor assignments and exam tasks, see 2.2.

[2] Actually, the course run under the name "Programming Paradigms" until autumn 1999.

We will now focus on the common and logic programming parts of the DP course.

2.1 The topics covered by the course

This section briefly describes the topics covered by the course, lecture by lecture, following the schedule of the 2004 spring semester.

Lecture 1. Introducing the declarative programming paradigm. A very simple declarative subset of the C language.[3]

Lecture 2. Introductory Prolog examples (family relations, summing numbers in binary trees), Prolog as a subset of logic, declarative semantics.

Lecture 3. Procedural semantics of Prolog, execution models (goal-reduction and procedure-box models).

Lecture 4. Data structures, unification, the logic variable.

Lecture 5. Operators, disjunction, negation, if-then-else.

Lecture 6. Lists, basic list handling library predicates.

Lecture 7. Example: finding paths in a graph, using various representations.

Lecture 8. Prolog syntax summary.

Lecture 16. Pruning the search space. Control predicates.

Lecture 17. Determinism, indexing, tail-recursion, accumulators.

Lecture 18. Rewriting imperative programs to Prolog, collecting and enumerating solutions.

Lecture 19. Meta-logical built-in predicates.

Lecture 20. Modularity, meta-predicates, meta-programming, dynamic predicates.

Lecture 21. Definite clause grammars, "traditional" built-in predicates.

Lecture 28. Brief outlook on LP extensions (external interfaces, coroutining, constraints).

The material covered in the lectures is made available to the students in the form of a textbook manuscript [12], updated every 2–3 years.

[3]The subset supports only integer types, function declarations and calls, `if` and `return` statements, the `+`, `-`, `*`, `/`, and `%` arithmetic operators, and the six comparison operators.

2.2 Programming examples and assignments

This section describes some of the programming examples and assignments used in the course.

Examples. At the very first lecture we show a fairly limited natural language conversation system (in Hungarian), which can remember statements and reply to questions related to these. This example program is then discussed in detail towards the end of the course (lecture 21, DCGs).

In lecture 4 we illustrate the symbolic processing capabilities of Prolog via a simple example program for building arithmetic expressions using the four main arithmetic operators (`+`, `-`, `*`, and `/`), each of which can be used zero or more times. The task is to build such an expression, so that both its value and the numbers it contains are given in advance (each given number should be used exactly once). A specific instance of this task, building an expression valued 24 from numbers 1, 3, 4, and 6, seems to be quite a difficult puzzle for humans ...

In lecture 19 we present a two-slide program implementing the standard ordering relation of Prolog (`@<`) (with the exception of variable ordering). This sample program re-iterates the definition of the standard ordering using a wide range of meta-logical predicates (`functor`, `arg`, `atom_codes`).

Minor Assignments. The minor assignments are non-compulsory, and have to be submitted via the ETS system. The solution of the student is then automatically tested on a set of predefined test cases and the results are sent back to the submitter via email.[4]

The minor assignments can be re-submitted any number of times, up to the deadline (which normally is about two weeks after the announcement). After the deadline all submissions are re-tested on a new test-set.

The first minor assignment in the 2004 spring semester was related to the declarative C language subset introduced in the first lecture. The students were asked to write a fairly complex function in this C subset, `palindrome(a)`, which returns the smallest integer `b` such that `a` written in base `b` is a palindrome.

The next two minor assignments were about binary trees storing key-value pairs, described by the following Mercury-style type declaration:

[4] As suggested by one of the reviewers, it would be better to show the results in the browser, rather than by sending them in an email. The reason for the latter choice is that some assignments (most notably the major ones) have longer running times (up to several minutes), so it would be infeasible to have the student wait for the results. On the other hand, the exercising facility of the ETS, see 2.3, is fully interactive, and provides immediate feedback for the student.

```
:- type tree(T) ---> node(tree(T),tree(T),T)
                   | empty.
```

In assignment 2 we ask for a predicate `tree_pair(Tree, Key, Value)` to be written. This predicate should enumerate all `Key`s and corresponding `Value`s stored in a given `Tree` of type `tree(pair(KType,VType))`.

In assignment 3 a predicate `pairs_keys_values(Tree, TreeK, TreeV)` has to be submitted. All three arguments are isomorphic trees, each node of `Tree` contains a `Key-Value` pair, where `Key` and `Value` are in the corresponding nodes of `TreeK` and `TreeV`. The predicate (of course) should be capable of both splitting a tree of pairs, and building such a tree from separate key- and value-trees.

The last and most complex minor assignment addresses issues related to meta-logical predicates and meta-programming. A predicate is to be written, which is capable of transforming an arbitrary Prolog term by wrapping all "good" lists `GL` occurring in it (at any depth) into a `good(GL)` structure. A list is good if it is ground and all its elements satisfy a condition. The condition is given as a term, which should be called using the `call/2` non-standard meta-predicate.

Major Assignment. There is a single, non-compulsory major assignment issued in a given semester, which can be be submitted in Prolog, or SML, or both languages. Solving the major assignment requires much more effort than the solution of the minor ones.

The major assignment is very often a logic puzzle. Several such assignments from earlier years have been published in the paper on the BUTE constraint course [11] (as the DP major assignment is normally re-used in the constraint course of the subsequent semester).

The major assignment issued in the 2004 spring semester was the Clouds puzzle:

> A rectangular board of size $n * m$ units is given. The task is to mark certain fields (unit squares) of the board as belonging to a cloud. The following conditions are known to always hold:
>
> 1. Clouds occupy an area of rectangular shape and their width and height is at least two units.
> 2. No clouds touch each other, not even diagonally.
>
> To solve the puzzle one is given the following pieces of information:
>
> 1. the size of the board;
> 2. the number of cloudy fields in certain rows/columns of the board;

3. the presence or absence of a cloud at certain fields of the board.

Figure 1 shows an example puzzle and its (unique) solution.[5] A number is given by each row and column. If this number is non-negative, it is equal to the count of the cloudy fields, otherwise the count is not known. Cloudy fields known in advance are represented by the '+' character, while fields known to be non-cloudy (clear) by '-'. In the solution (right hand side) '#' represent cloudy fields and '.' clear sky.

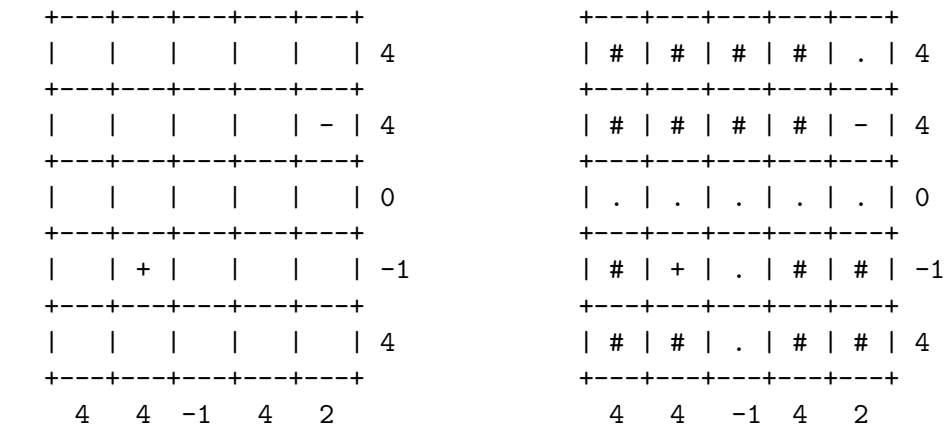

Figure 1: A Clouds puzzle and its solution

We use at least three distinct sets of test cases for assessing the students' programs. The first set is distributed with the announcement of the assignment, so that students can use it in the process of program development. The second set is used when the students hand in their solutions. The results of this test are sent back to the students via email. As for the minor assignments, students can hand in as many versions of their programs as they wish, but only the latest version will be used for the final test. This test is run on the third set of test cases, which has a similar difficulty level as the earlier ones. Students also have to submit a documentation via the ETS system.

The best solutions for the major assignment, which solve all the test cases of the final test, participate in the so-called "ladder contest". These programs are tested against bigger and more difficult test cases. The authors of the programs which achieve highest places in the ladder contest, get extra points.

In spring 2004 the largest test-case in the ladder test-set was of size 25*18, with 92 given fields. Only a single student submission was capable of solving this

[5]Naturally, a puzzle can have multiple solutions.

test case within the prescribed 120 second timeout.

2.3 Teaching tools

We now briefly introduce the tools and utilities used by the students and/or the lecturers during the DP course.

The ETS electronic teaching assistant. The web-based ETS system provides the following services:

- access to a database of students of the course, together with their results,
- assignment submission and automatic testing,
- facilities for student exercising.

The ETS system has been developed by the course lecturers together with several talented students of the course, who devoted their student research work [1] or Master's Thesis [2] to this subject.

We now focus on the student exercising facilities of ETS. The system supports various exercise schemes for both the Prolog and SML languages. As worked out in [2], a scheme describes how the exercise is presented to the student as well as how the student answer is processed. Correspondingly, a scheme is associated with a web-form, which includes the generic text of the exercise. This form is then specialised using a database of concrete exercise instances.

At present the following Prolog schemes are supported:

- Canonical form: the student is requested to type in the canonical form of a Prolog term (as if printed by `write_canonical`).
- Execution: The student should decide what will be the result of executing a given goal. There are three sub-schemes, depending on the determinism, number of variables, etc.:
  - Success/failure/error: The goal is deterministic and it can fail or raise an exception. In case of success, the value of a single variable is asked for.
  - Multiple variables: The goal is deterministic and it always succeeds. The student is asked to supply the substitutions of all variables. Typically used for unification exercises.

  - All solutions: The goal is possibly non-deterministic with a single variable. The student is asked to enumerate all solutions of the goal, in proper order.[6]

- Programming: The student has to write a Prolog program following a given specification.

Orthogonally to the schemes, the exercises are grouped into topics, which correspond to sections of the material taught. For example, the topic of lists may contain exercises in various schemes: canonical form (of lists), execution and programming (of list processing predicates).

For most of the topics the exercises come in several difficulty levels: easy, medium, and hard. Students can prescribe the difficulty level of exercises they want to practice.

We now give a few examples of various exercises in the ETS system.

- "Canonical form" scheme, lists and operators topic, hard exercises:

```
[[],[]]     - (1,2)     1 - - 1.
```

- "Success/failure/error" scheme, unification topic:

```
| ?- [X,1|X] = [_,_].
| ?- [X,a,X] = [1,2,a,1,2].
```

- "Multiple variables" scheme, unification topic:

```
| ?- g(1+2+3, [a,b]) = g(X+Y, [U|V]).
| ?- h([H, G], H*G) = h([Q/1|R], P/Q*3).
```

- "Success/failure/error" scheme, list processing and control predicates topics:

```
| ?- length(X, 1), member(a, X).
| ?- member(X, [1,2,3]), !, X < 3.
```

- "Multiple solutions" scheme, list processing topic:

[6] This sub-scheme could be eliminated in principle by encapsulating the goal into a `findall`. However this type of exercise is issued much earlier than `findall` is taught.

Program:	`app([X|L1], L2, [X|L3]) :-`
	`  app(L1, L2, L3).`
	`app([], L, L).`

Goal:	`| ?- app(L, [a|_], [a,b,a,b,a]).`
Task:	List the substitution(s) of `L` separated by `;`'s.

(The correct answer is: `[a,b,a,b];[a,b];[]`.)

Further details and examples regarding the capabilities of ETS can be found in [3].

Other tools. The declarative C language subset, used in the initial lecture has also been implemented and its interpreter has been made available to the students. This interactive implementation has been written in Prolog and it works by translating the C code to Prolog predicates.[7] Students thus can get acquainted with declarative programming and interactive function invocation in an already familiar environment of the C language.

Further tools serve for the visualisation of Prolog execution. First I developed a simple, non-graphical tool for drawing the search tree of a pure Prolog program. Subsequently, as a student research project, this was extended to general Prolog programs and a graphical interface was also provided [7].

Regarding tools used by the lecturers, we should mention that the slide presentations of the whole course (both Prolog and SML) have been developed using the `xdvipresent` tool [5]. Another, very important utility serves for detecting plagiarism in the student assignments. This tool has been developed in a student research project in 2000 [6] and since then it have excellently served the purpose of deterring students from copying each others work. The basic idea is to transform the programs into call-graphs and compare these graphs, rather than the program texts. Of course, this tool can not be used for simple assignments, where the code to be written consists of a few predicates or functions.

At present, the plagiarism detection tool contains front-ends for both SML and Prolog, but there are plans to extend it for further languages.

3 Elective courses

There are two elective courses closely related to logic programming:

[7] In the future we plan to hand out this compiler to students, and to use it in the second part of the course for illustrating the advantages of Logic Programming in compiler writing, cf. [13].

- "Highly efficient logic programming" held each year since 1997,
- "Selected topics from logic programming", student seminar, held in 2001 and 2003.

Both courses are single semester ones, with one lecture/seminar of 90 minutes per week.

The "Highly efficient logic programming" (HELP) course tries to show two facets of efficient logic programming:

- Making programs run faster by creating a streamlined and clean LP language, as exemplified by the Mercury project [10].
- Making programs run faster by adding more reasoning capabilities as shown by the constraint logic programming (CLP) approach.

However, the emphasis of the course shifted towards CLP, which takes about 11–12 lectures, while the basic outline of Mercury is presented in the remaining 2–3 lectures. The CLP part is based on SICStus Prolog, discussing all four constraint libraries present: CLP(R/Q), CLP(B), CLP(FD) and CHR.

There is an important link between the DP and the HELP course: the major assignment of the DP course is re-issued in the subsequent semester as the HELP assignment. It is not uncommon that the CLP program written as the HELP assignment is two orders of magnitude faster than the Prolog solution of the same student.

Further details on the HELP course can be found in [11].

In the seminar entitled "Selected topics from logic programming" students themselves present their account on a topic of their interest, be it a paper, some experience with an application or a programming language, etc. The lecture themes come from various areas of logic programming: theory, parallelism, object-oriented, graphical, and web-related extensions, abstract interpretation, tracing, as well as overviews of concrete Prolog implementations.

In the first few weeks I hold the seminars, so that the students have time for preparing their presentations. In both editions of the course so far, these lectures were about Prolog implementation techniques, primarily discussing the WAM approach. This is followed by student presentations of 45–90 minutes, depending on the topic. The students are asked to prepare their presentation at least one week in advance, and to discuss it with me; this ensures fairly good quality. The seminars are often followed by a lively discussion. Overall, the students were quite satisfied with the seminar.

In spring 2004 a new course on the Semantic Web was set up by myself and Gergely Lukácsy. Although this course is rather loosely related to LP, it is worth

mentioning that several students have prepared their assignment, a tableaux based reasoner for a Description Logic, in Prolog.

4 Other educational activities

There are several types of activities at BUTE, where enthusiastic students can explore some topics in greater depth.

Directed Projects. In semesters 8 and 9 all students of informatics have to do a Directed Project with a weekly load equivalent to six 45 minute lectures. I had the pleasure of leading about a dozen such student projects over the last seven years, in various areas of logic programming. Some of the more interesting ones are listed below:

- Applying CLP(FD) for scheduling plastic moulding machines — Tamás Benkő, 1997
- Interfacing Prolog to Corba — Gábor Gesztesi and Gábor Marosi, 1997–98
- Debugging CLP(FD) programs — Dávid Hanák and Tamás Szeredi, 2000 (this led to the development of a new SICStus library [4])
- Using CHR for reasoning in Description Logics — Bence Szász, 2002
- A Prolog based RDF reasoning system — Gergely Lukácsy, 2002
- A Prolog-Java interface using sockets — Péter Biener, 2003

Masters' Theses. It is quite often the case that students chose their work in the Directed Project as the basis for their MSc Thesis. In fact this happened to all but the Corba project in the above list. Some further Theses I supervised are listed below:

- Implementation of a constraint reasoning system — Tamás Rozmán, 1997
- Conversion of document description languages — Zsolt Lente, 2000
- Knowledge-based tools for information integration — Attila Fokt, 2000
- Computer Support for Declarative Programming Courses — Dávid Hanák, 2001

- Verification of object-oriented models using constraints — Péter Tarján, 2001
- Transforming object-constraints to logic — Károly Opor, 2002
- Logic-based methods for planning queries on heterogeneous data sources — Tamás Lukács Berki, 2003

Student Research Projects. There is a long tradition of Hungarian students doing a research project, in addition to their curricular activities. Such student research work is assessed at yearly Student Conferences, to which students have to submit a paper (usually 40–60 pages) and also deliver a 20 minute oral presentation.

Separate Student Conferences are held every autumn within each Faculty of BUTE. In November 2003, there were 118 presentations involving about 170 students in 11 sections, in the Student Conference of the Faculty of Electrical Engineering and Informatics. Best papers get first, second and third prizes, these rewards are taken into account e.g. in the admission procedure of PhD studies. Paper which have received a first prize participate in the biennial National Students Conference.

The following Student Conference papers were presented in the area of (constraint) logic programming:

- Solving a stock exchange allocation problem (using CLP), Dániel Varró, 1998, I. Prize.
- Conversion of SGML languages, Zsolt Lente, 1999, II. Prize.
- Comparison of source program structures, Gergely Lukácsy, 2000, I. Prize, Rector's Special Prize; I. Prize at the National Student Conference, 2001
- Efficient access of an object-oriented database from logic programs, Ambrus Wagner, 2000
- A Web-based student exercising system for teaching programming languages, András György Békés, Lukács Tamás Berki, 2001
- Intelligent querying and reasoning on the Web, Gergely Lukácsy, 2002, I. Prize; Special Prize of the Hungarian W3C Office at the National Student Conference, 2003
- Visualisation of Prolog program execution, Tamás Nepusz, 2003, II. Prize

- Using abstract interpretation in SICStus Prolog, Balázs Leitem, 2003, III. Prize

5 Summary

I think it is a very good thing that each and every student of informatics at BUTE (which currently means about 400 students every year) has to get acquainted with the basics of logic and functional programming. I only hope that this remains so when the current five year curriculum (leading to an MSc degree) gets divided into two parts, following the Bologna principles.

It looks that the introductory DP course can attract the attention of talented students, who then get involved in further elective courses. A possible explanation of this is that, as opposed to other programming courses, such as C, Java, etc., the DP course introduces a programming style previously unknown to most students, and at the same time it shows some problems where this new paradigm can be successfully used. The major assignment, which is not compulsory, but is practically required for achieving the highest grade, gets the attention of the best students as a task which looks very difficult to solve using traditional, imperative languages. The ladder contest for the major assignment adds to this the thrill of a peer-to-peer competition. Consequently, quite a few students make an attempt to solve the major assignment, and so get some practice in declarative programming.

The major assignment of the DP course has almost always been a constraint problem. I often mention to students at the DP exam that their solution can be made hundred times faster using constraint techniques, which they can get acquainted with in the HELP course. This "advertisement" seems to work: for example, there are now 59 students enrolled for the 2004 autumn HELP course (as a reference, 480 students were enrolled in the 2004 spring DP course).

Both the logic programming and the CLP courses focus more on the practical programming aspects, and less on the theoretical ones. Although I agree that it would be beneficial to make the courses a bit more balanced in this respect, I think that the primary interest of the students, at least here in Hungary, lies in the practical applicability of LP, rather than in its theory. It may be interesting to note, however, that in the last edition of the LP seminar there was a student-initiated lecture on theoretical foundations of (C)LP, based on the first chapters of [8].

The student evaluations[8] of the courses described in this paper have almost always been positive. Nevertheless there are quite a few issues that need im-

[8]Students at BUTE are asked to fill in a questionnaire for each lecturer of each course every semester.

provement.

Most importantly, the better integration of the functional and logic programming parts of the DP course is always on the agenda. Teaching a single language encapsulating both functional and logic programming aspects (such as Oz, or Mercury) would help to avoid the (mostly syntactic) confusion of the present Prolog + SML setup. On the other hand, the novel languages mentioned have much less industrial acceptance at present. Also, switching to a new language would require a major investment on the lecturers' part, which we cannot afford presently. Therefore, we now remain with the two languages, and try to link the two topics in the lectures, pointing out the similarities and differences of Prolog and SML.

There is also scope for improvement regarding the utilities and tools used in the courses. Students without Internet access would prefer a stand-alone exercising tool to the present Web-based one. The set of exercise schemes and exercises needs further expansion, especially regarding exercises involving program writing. The administrative part of ETS supports a single course in a single semester, it would be good to eliminate both limitations. Several improvements are underway regarding the tool for Prolog execution visualisation. The plagiarism detection tool has to be extended to support the declarative subset of the C language, introduced recently.

As the referees rightly pointed out, the paper would have benefited from the inclusion of more details on the mathematical logic course, as well as on the functional programming part of the DP course. Unfortunately, the strict time constraints did not allow for these to be included.

6 Conclusions

In the paper I tried to give an overview of the role logic programming plays in undergraduate education and research at the Budapest University of Technology and Economics. I hope that the experiences reported here can be of help in LP education at other universities.

Acknowledgement

I am indebted to Péter Hanák, who, exactly 10 years ago, invited me to teach declarative programming at BUTE, and with whom I could share the joy and troubles of this difficult task. I am also most grateful to all the enthusiastic student helpers (we had about 50 of these!), and especially to those who helped in the development of various utilities and tools: András György Békés, Tamás

Benkő, Lukács Tamás Berki, Dávid Hanák, Gergely Lukácsy, Tamás Nepusz, and Tamás Rozmán.

Thanks are also due to the anonymous referees for their helpful remarks.

References

[1] András György Békés and Lukács Tamás Berki. A Web-based student exercising system for teaching programming languages (in Hungarian), 2001. Students' Conference, BUTE, Budapest, Hungary.

[2] Dávid Hanák. Computer support for declarative programming courses (in Hungarian), 2001. MSc Thesis, BUTE, Budapest, Hungary.

[3] Dávid Hanák, Tamás Benkő, Péter Hanák, and Péter Szeredi. Computer aided exercising in Prolog and SML. In *Proceedings of the Workshop on Functional and Declarative Programming in Education, PLI 2002, Pittsburgh PA, USA*, October 2002.

[4] Dávid Hanák and Tamás Szeredi. FDBG, the CLP(FD) debugger library of SICStus Prolog. In Susana Muñoz Hernández and José Manuel Gómez-Pérez, editors, *Proceedings of the Fourteenth International Workshop on Logic Programming Environments (WLPE'04)*, Saint-Malo, France, September 2004.

[5] Manuel Hermenegildo. Slide presentations using latex/xdvi, 2003. CLIP Group, School of Computer Science, Technical University of Madrid, `http://clip.dia.fi.upm.es/Software/xdvipresent_html`.

[6] Gergely Lukácsy. Comparison of source program structures (in Hungarian), 2001. National Students' Conference, Eger, Hungary.

[7] Tamás Nepusz. Visualisation of Prolog program execution (in Hungarian), 2003. Students' Conference, BUTE, Budapest, Hungary.

[8] Ulf Nilsson and Jan Maluszynski. *Logic, Programming and Prolog (2nd ed).* John Wiley, 1995.

[9] SICS, Swedish Institute of Computer Science. *SICStus Prolog Manual, 3.11*, June 2004.

[10] Zoltan Somogyi, Fergus Henderson, and Thomas Conway. The execution algorithm of Mercury: an efficient purely declarative logic programming language. *Journal of Logic Programming*, 29(1-3):17–64, 1996.

[11] Péter Szeredi. Teaching constraints through logic puzzles. In Krzysztof R. Apt et al., editor, *Recent Advances in Constraints, Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2003, Selected Papers*, volume 3010 of *Lecture Notes in Computer Science*, pages 196–222. Springer, 2004.

[12] Péter Szeredi and Tamás Benkő. *Introduction to logic programming (in Hungarian)*. Budapest University of Technology and Economics, Faculty of Electrical Engineering and Informatics, 1998, 2001, 2004. Study-aid for the Declarative Programming course. Manuscript.

[13] David H. D. Warren. Logic programming and compiler writing. *Software Practice and Experience*, 10:97–125, 1980.