Teaching Logic Programming

at the Budapest University of Technology

Péter Szeredi
`szeredi@cs.bme.hu`

Dept. of Computer Science and Information Theory,
Budapest University of Technology and Economics (BUTE)
H-1117 Budapest, Magyar tudósok körútja 2.

Overview of the talk

- The Budapest University of Technology and Economics
- The Declarative Programming course
- Elective LP courses
- Other educational activities
- Conclusions

The Budapest University of Technology and Economics

- BUTE — Budapest University of Technology and Economics:
  - Established in 1782
    - Initially: Institutum Geometrico-Hydrotechnicum
    - within the Faculty of Liberal Arts, University of Buda
  - Present:
    - Seven faculties, >110 departments and institutes
    - About 1000 full-time lecturers, several hundred research staff
    - Approx. 10,000 students, of which 10% are foreigners
    - BUTE issues about 70% of Hungary's engineering diplomas
- Faculty of Electrical Engineering and Informatics — the home of computer-related education
  - About 200 full time teaching staff
  - Approx. 3,000 students, majoring in Electric Engineering and Technical Informatics (roughly equivalent to Computer Science)
- Five year (10 semester) education leading to an MSc/MEng degree
- More info at: `http://www.bute.hu/en/`

Logic Programming education at BUTE

- Students exposed to LP-related topics (and the relevant subjects):
  - Majors in Technical Informatics at Faculty of Electrical Eng. and Informatics (approx. 400/year)
    - Mathematical Logic — semester 3, compulsory course
    - Declarative Programming (DP) — semester 4, compulsory course
    - Highly Efficient Logic Programming (HELP) — elective course
    - Selected Topics from Logic Programming (STLP) — elective student seminar
    - AI laboratory — semester 7 (specialisation Intelligent Systems)
  - Majors in Mathematics, at Faculty of Natural Sciences, with specialisation in Algebra and applications (approx. 10/year)
    - Logic Programming — LP (semester 7, compulsory course, same as the logic programming part of DP)
  - Students at some other faculties (e.g. Mechanical Engineering)
    - various courses on AI and Programming
- The author is involved in the DP (LP), HELP, STLP courses — this is the focus of the present talk

The Declarative Programming course

- Course topics
  - Functional programming, SML — Péter Hanák
  - Logic programming, Prolog — Péter Szeredi
- Course data
  - Base position: Semester 4 (spring), cross semester: 5 (autumn)
  - Students enrolled:
    - spring 2004: 473
    - autumn 2004: 162
  - Two 2*45 minute lectures/week, 14 weeks/semester
  - No laboratory exercises :-(
  - Presented — 20 times so far — since 1994
    (until 1999, under the title Programming Paradigms)
- Preceding courses
  - Programming languages: C, C++, Java
  - Mathematical Logic

Global course structure

- Major lecture blocks

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

- Links between the functional and logic programming parts
  - Common concepts and techniques, e.g. tail-recursion, accumulators, construction and decomposition using pattern-matching
  - Common major assignment, occasionally common minor assignments

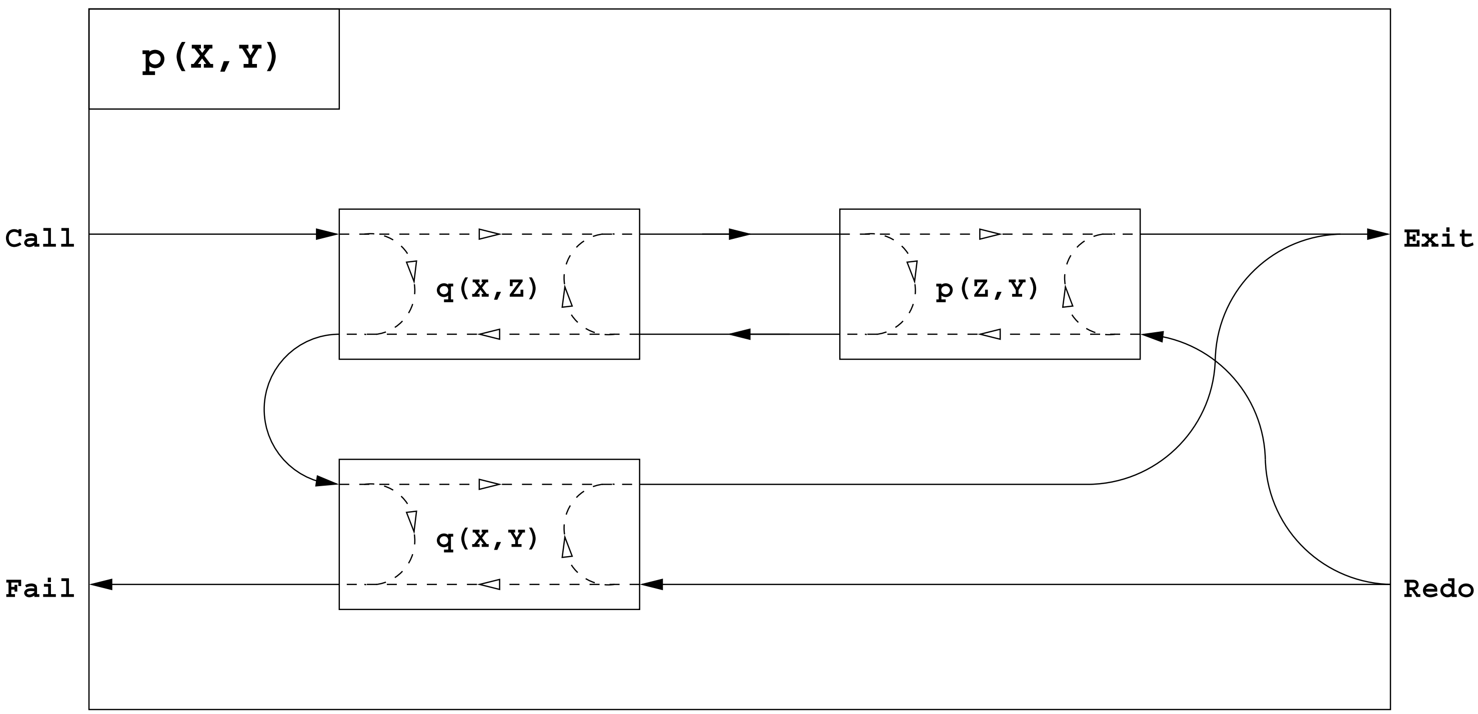
Topics in LP — part 1

Lecture 1.	Introducing the declarative programming paradigm. A very simple declarative subset of the C language (Cékla = beet-root in Hungarian :-).
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.

SAMPLE SLIDE: Procedure-box model example

```
p(X,Y) :- q(X,Z), p(Z,Y).
p(X,Y) :- q(X,Y).

q(1,2). q(2,3), q(2,4).
```



SAMPLE SLIDE: Box model — OO view, the „next solution” method of `p/2`

```
bool p::next()
{ switch(clno) {
  case 0:                    // entry point for the Call port
    clno = 1;                // enter clause 1:                          p(X,Y) :- q(X,Z), p(Z,Y).
    qaptr = new q(x, &z);    // create a new instance of subgoal q(X,Z)
  redo11:
    if(!qaptr->next()) {     // if q(X,Z) fails
      delete qaptr;          // destroy it,
      goto cl2;              // and continue with clause 2 of p/2
    }
    pptr = new p(z, py);     // otherwise, create a new instance of subgoal p(Z,Y)
  case 1:                    // (enter here for Redo port if clno==1)
    /* redo12: */
    if(!pptr->next()) {      // if p(Z,Y) fails
      delete pptr;           // destroy it,
      goto redo11;           // and continue at redo port of q(X,Z)
    }
    return true;             // otherwise, exit via the Exit port
  cl2:
    clno = 2;                // enter clause 2:                          p(X,Y) :- q(X,Y).
    qbptr = new q(x, py);    // create a new instance of subgoal q(X,Y)
  case 2:                    // (enter here for Redo port if clno==1)
    /* redo21: */
    if(!qbptr->next()) {     // if q(X,Y) fails
      delete qbptr;          // destroy it,
      return false;          // and exit via the Fail port
    }
    return true;             // otherwise, exit via the Exit port
  } }
```

Topics in LP — part 2

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).

SAMPLE SLIDE: Meta-logical predicates, implementing term-precedes

```
% T1 precedes T2-t in standard order. (Equivalent to T1 @< T2, except for
% variables, these are ordered by their first occurrence in T1-T2.)
precedes(T1, T2) :-
        \+ \+ (numbervars(T1-T2, 0, _), prec(T1, T2)).

% class(+T, -C): Term T belongs to term class C.
class(T, C) :-
        (   T='$VAR'(_) -> C=0          % variable
        ;   float(T)    -> C=1          % float
        ;   integer(T)  -> C=2          % integer
        ;   atom(T)     -> C=3          % atom
        ;   compound(T) -> C=4          % compound
        ).

% Numbervar'd term T1 precedes term T2-t.
prec(T1, T2) :-
        class(T1, C1), class(T2, C2),
        (   C1 =:= C2 ->
            (   C1 =:= 1 -> T1 < T2     % floating point numbers)
            ;   C1 =:= 2 -> T1 < T2     % integers
            ;   struct_prec(T1, T2)     % variables, atoms, compounds
            )
        ;   C1 < C2                     % different term classes
        ).
```

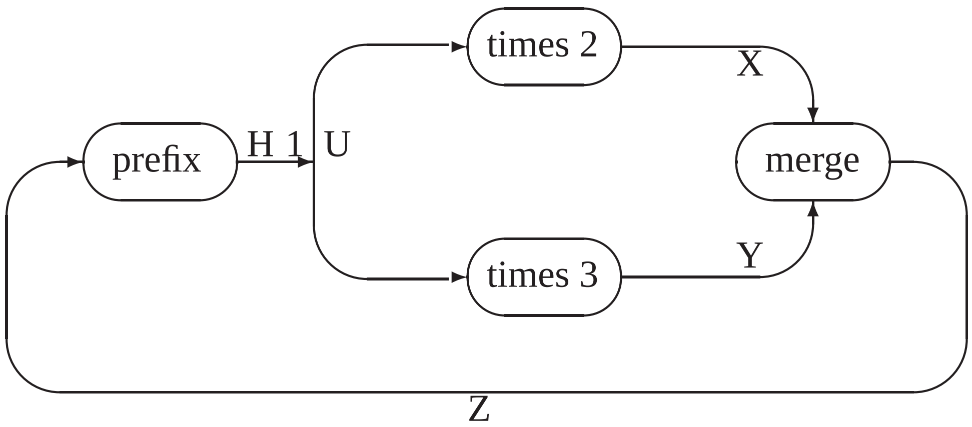
SAMPLE SLIDE: implementing term-precedes (contd.)

```
% S1 precedes S2  (S1 and S2 are callable terms, i.e. atoms or compounds).
struct_prec(S1, S2) :-
        functor(S1, F1, N1), functor(S2, F2, N2),
        (   N1 < N2 -> true
        ;   N1 =:= N2,
            (   F1 == F2 -> args_prec(1, N1, S1, S2)
            ;   atom_prec(F1, F2)
            )
        ).

% For the first argument position i, N0=<i=<N, for which
% S1[i] differs from S2[i], S1[i] precedes S2[i] (and such i exists).
args_prec(N0, N, S1, S2) :-
        N0 =< N,
        arg(N0, S1, A1), arg(N0, S2, A2),
        (   A1 = A2 -> N1 is N0+1, args_prec(N1, N, S1, S2)
        ;   prec(A1, A2)
        ).

% Atom  A1 precedes atom A2.
atom_prec(A1, A2) :-
    atom_codes(A1, C1), atom_codes(A2, C2), struct_prec(C1, C2).
```

SAMPLE SLIDE: Coroutining: Simplified Hamming problem (partial code)




```
% List H contains the first N numbers whose all prime factors are 2 or 3.
hamming(N, H) :-
     U = [1|H], times(U, 2, X), times(U, 3, Y),
     merge(X, Y, Z), prefix(N, Z, H).

% times(X, M, Z):  Elements of list Z are those of X multiplied by M.
:- block times(-, ?, ?).
times([A|X], M, Z) :- B is M*A, Z = [B|U], times(X, M, U).
times([], _, []).

(...)
```

Assignments

- How to teach a programming language without laboratory exercises?
  - Assignments: multiple minor and a single major assignment (common to LP and FP)
  - Interactive exercising tool (in ETS, see later)
  - All these are non-compulsory
- Minor assignments
  - 4-6 assigments alltogether in LP and FP
  - Fairly simple problems, < 20 lines of code needed
  - Submission within two weeks
- Major assignment
  - A single task to be solved in Prolog and/or SML
  - A scalable problem, most often a logic puzzle
  - Even a simple generate-and-test solution gets some credit
  - Best solutions compete in a „ladder contest” for extra credit
  - Submission in 4-6 weeks

SAMPLE: Minor assignment 1

- Write in Cékla a function `palindrome(n)`
  - `palindrome(n)` returns the smallest integer `b>1`
  - such that `n` written in base `b` is a palindrome.
- Main characteristics of the Cékla C subset:
  - integer type only
  - functions, `if`, and `return` statements
  - `+`, `-`, `*`, `/`, and `%` arithmetic operators, the six comparison operators

The solution of minor assignment 1

```
/*  The reverse of number ‘n’ in base ‘base’ appended to ‘revn’ */
int reverse(int n, int base, int revn) {
  if (n == 0)
    return revn;
  else {
    int last_digit = n % base;
    return reverse(n/base, base, revn*base+last_digit);
  }
}

/* The smallest integer ‘b’ >= ‘base’, so that ‘n’ written
   in base ‘b’ is a palindrome */
int palindrome1(int n, int base){
  if  (reverse(n, base, 0) == n)
    return base;
  else return palindrome1(n, base+1);
}

/* The smallest integer ‘b’, so that ‘n’ written in base ‘b’ is a palindrome */
int palindrome(int n) {
  return palindrome1(n,2);
}
```

Major assignment in spring 2002: the Magic Spiral puzzle

- The puzzle:
  - A square board of $n * n$ fields is given.
  - Place integer numbers, chosen from the range $[1..m]$, $m \leq n$, on the board, so that:
    1. in each row and each column all integers in $[1, m]$ occur exactly once (and so there are $n - m$ empty fields);
    2. along the spiral starting from the top left corner, the integers follow the pattern $1, 2, \ldots m, 1, 2, \ldots, m, \ldots$ (number m is called the period of the spiral).
  - Initially, some numbers are already placed on the board.
- An example puzzle and its solution:

			4			
1						

		1	2	3		4
2		3	4	1		
1	3	4			2	
4	2				3	1
3	1				4	2
	4		3	2	1	
		2	1	4		3

Major assignment in spring 2003: the Snake puzzle

- The puzzle:
  - A rectangular board of $n * m$ fields is given.
  - Mark certain fields on the board as belonging to the *snake*:
    - The snake consists of a sequence of neighbouring fields (i.e. fields sharing a side).
    - The position of the snake head (the first field) and of the tail (last field) is given.
    - The snake can touch itself only diagonally.
    - Certain fields of the table contain a number.
      - The snake cannot pass through fields containing a number.
      - If a field contains number c, then among the eight side and diagonal neighbours of this field there are exactly c which contain a snake piece. (cf. the minesweeper game).
- An example puzzle and its solution:

H		**T**		
		5		
4				
			3	

•		•	•	
•	•	**5**	•	•
4	•			•
	•	•	•	•
			3	

Major assignment in spring 2004: the Clouds puzzle

- The puzzle:
  - A board of $n * m$ fields is given. Mark certain fields on the board as belonging to a *cloud*:
    - Clouds occupy an area of rectangular shape, their width and height is at least two units.
    - No clouds touch each other, not even diagonally.
  - Supplied information:
    - the size of the board;
    - the number of cloudy fields in certain rows/columns of the board (`-1` means unknown);
    - the presence (`+`) or absence (`-`) of a cloud at certain fields of the board.
- An example puzzle and its solution:

```
+---+---+---+---+---+          +---+---+---+---+---+
|   |   |   |   |   | 4        | # | # | # | # | . | 4
+---+---+---+---+---+          +---+---+---+---+---+
|   |   |   |   | - | 4        | # | # | # | # | - | 4
+---+---+---+---+---+          +---+---+---+---+---+
|   |   |   |   |   | 0        | . | . | . | . | . | 0
+---+---+---+---+---+          +---+---+---+---+---+
|   | + |   |   |   | -1       | # | + | . | # | # | -1
+---+---+---+---+---+          +---+---+---+---+---+
|   |   |   |   |   | 4        | # | # | . | # | # | 4
+---+---+---+---+---+          +---+---+---+---+---+
  4   4  -1   4   2              4   4  -1   4   2
```

Tools used in teaching Declarative Programming

- The Electronic Teaching aSsistant (ETS) is a Web based facility providing
  - access to a database of students of the course, and their results,
  - assignment submission and automatic testing,
  - facilities for student exercising.
- The RDBG reduction debugger visualisation tool
  - step-by-step building of the Prolog search tree
  - skipping, unleashing, breakpoints
- The MATCH plagiarism-detection tool
  - detects similar assignment solutions
  - comparing the call graphs of the programs
  - with front-ends for both Prolog and SML
- Other tools
  - The Cékla interpreter
  - The `xdvipresent` package for slides

Student exercising with ETS

- Each exercise belongs to a scheme and a topic and has a difficulty:
  - The **scheme** determines the presentation and processing **format**, e.g. the student is expected to predict what happens when a given goal is run
  - The **topic** corresponds to the **part of the material** taught, e.g. list processing
  - The **difficulty** can be easy, medium, or hard.
- Prolog schemes
  - Execution: decide on the result of executing a given goal
    - Success/failure/error: deterministic goal, can fail or raise an exception. A single variable assignment is asked for.
    - Multiple variables: supply the substitutions of all variables of a deterministic goal (typically unification).
    - All solutions: enumerate all solutions of a non-deterministic goal, in proper order.
  - Canonical form: type in the canonical form of a Prolog term (cf. `write_canonical`).
  - Programming: write a Prolog program following a given specification.

Student exercising with ETS — examples

- Scheme: „Canonical form", topic: lists/operators

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

- Scheme: „Success/failure/error", topic: unification

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

- Scheme: „Multiple variables", topic: unification

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

- Scheme: „Success/failure/error", topic: list processing and control predicates

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

- Scheme: „Multiple solutions", topic: list processing:

Program:	`app([X|L1], L2, [X|L3]) :- app(L1, L2, L3).` `app([], L, L).`
Goal:	`| ?- app(L, [a|_], [a,b,a,b,a]).`

Elective courses — Highly Efficient Logic Programming

- 14 lectures, 2*45 minutes each, so far held 8 times since 1997
- Course topics
  - Mercury — efficiency through streamlining and cleaning the LP language (2 lectures)
  - CLP($\mathcal{X}$) — efficiency through additional reasoning capabilities (12 lectures)
    - the CLP(B), CLP(R/Q), CLP(FD) and CHR libraries of SICStus Prolog
    - major assignment usually shared with the preceding DP course
- Typical course layout (1 unit = 45 minutes)

	Topic	Time	Slides
1.	Prolog extensions relevant for CLP	3 units	15 slides
2.	The CLP($\mathcal{X}$) scheme and CLP(R/Q)	3 units	19 slides
3.	CLP(B)	2 units	8 slides
4.	CLP(FD)	14 units	109 slides
5.	Constraint Handling Rules	2 units	11 slides
6.	Mercury	4 units	23 slides
	$\sum$	28 units	185 slides

- For more details see Szeredi's paper in LNAI 3010

Other elective courses

- Selected Topics from Logic Programming — student seminar
  - 14 lectures, 90 minutes each, so far held twice, in 2001 and 2003
  - I hold the first few lectures, normally on Prolog implementation
  - Subsequent (45 or 90 minute) lectures held by students, topics:
    - theory of LP
    - parallel logic programming,
    - object-oriented, graphical, and web-related extensions,
    - abstract interpretation,
    - tracing,
    - overviews of concrete Prolog implementations
- The Semantic Web and Ontologies (by Gergely Lukácsy and Péter Szeredi)
  - Web technologies, RDF, Description Logics, etc.
  - Link to LP: several students chose to write the assignment, a tableaux based reasoner for a Description Logic, in Prolog

Other LP-related activities — Directed Projects

- Semesters 8 and 9, weekly load equivalent to 6*45 minutes course
- Some Directed Projects I led:
  - 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, see the poster and the talk at WELP)
  - 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
- Typically Directed Projects serve as the basis for the MSc Thesis
  - All above projects, except for the Corba one, led to an MSc Thesis

Other LP-related activities — MSc Theses

- MSc Theses I supervised (further to Theses resulting from Directed Projects):
  - 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 — Lukács Tamás Berki, 2003

Other LP-related activities — Student Research Projects

- Student Research Projects — a long tradition at Hungarian Universities
  - Yearly, faculty-level Student Conference (TDK in Hungarian):
    - 40-60 page paper
    - 20 minute presentation
    - best papers get 1st, 2nd and 3rd prizes
  - Biennial National Student Conferences
- Student Conference in 2003 at the Faculty of Electrical Engineering and Informatics at BUTE:
  - 118 presentations,
  - involving approx. 170 students

Other LP-related activities — Student Research Projects (contd.)

- Student Conference Papers related to LP:
  - 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

Discussion

- Good points
  - Each year about 400 students of BUTE get acquainted with logic programming
  - Logic programming attracts the attention of talented students
    - programming style previously unknown to most students
    - the course shows some problems where the new paradigm can be used
    - the major assignment poses a challenge, as it would be quite difficult to solve using imperative languages
    - the ladder contest adds the thrill of peer-to-peer competition
  - Most talented students get further involved with LP:
    - learn constraint logic programming in the HELP course (top 10%)
    - explore further areas of logic programming in the STLP seminar (top 2–3%)
    - work on logic programming as part of his/her Directed Project, Student Research Project, or Thesis (top 1%)
  - Student evaluation of the LP courses is fairly positive

Discussion (contd.)

- Debatable points
  - Practice vs. theory of LP
    - the courses focus more on the practical programming side than on the theoretical side of LP
    - this seems to fit the students' interest better
    - theoretical foundations get discussed in the STLP seminar
  - Language(s) taught
    - a single FP+LP language (such as Oz, or Mercury) would avoid a lot of student confusion (mostly syntactic)
    - Prolog and SML have a much larger user/application base, and so will be more likely of use to the graduate
    - switching to a new language requires major investment on the lecturers' part
    - consequently, we stay with the present setup, and try to link the LP and FP topics as much as possible

Future Work

- Electronic Teaching aSsistant
  - stand-alone exercising tool need for students without Internet access
  - set of exercising should be made larger
  - extension of the administrative side to support multiple courses and multiple semesters
- The RDBG execution visualisation tool — several extensions underway
- The Match plagiarism-detection tool
  - front-ends for new languages, such as Cékla
- . . .

Acknowledgements

- Péter Hanák, exactly 10 years ago, invited me to teach declarative programming at BUTE.
- Enthusiastic student helpers (we had about 50 of these!) did a lot to make the task of teaching LP and FP possible
- Several students carried the major burden of development of various utilities and tools:
  - András György Békés
  - Tamás Benkő
  - Lukács Tamás Berki
  - Dávid Hanák
  - Gergely Lukácsy
  - Tamás Nepusz
  - Tamás Rozmán