Computer Aided Exercising in Prolog and SML

Dávid Hanák, Tamás Benkő, Péter Hanák, Péter Szeredi

Budapest University of Technology and Economics, Hungary
{dhanak,benko,hanak,szeredi}@inf.bme.hu

FDPE Workshop, Pittsburgh, PA, USA

October 7, 2002

1. Introduction

The *Declarative Programming* course

- held at the *Budapest University of Technology and Economics* (BUTE);
- for 4th semester students in Computer Science;
- as an introduction to functional and logic programming;
- via (Moscow) SML and (SICStus) Prolog;
- emphasis is placed on the *declarative aspects*;

Constraints

- number of students increased from approx. 100 to more than 400 in eight years;
- the staff consists of two part-time lecturers and two PhD students;
- there are no laboratory exercises.

All these call for a system helping the work of lecturers and students.

ETS - An Environment for Teaching Students [1] is an integrated system of loosely connected components doing auxiliary tasks:

- evaluation of assignments;
- **exercising;**
- administration; etc.

2. The Exercise System

Functionality

- offers several types of simple exercise tasks;
- presents a problem, receives and checks the solution;
- reports any possible errors and requests correction;
- follows and registers the progress of students.

Design objectives

- user friendly interface;
- "fool proof" error handling with informative error messages;
- protection against malicious programs;
- easy management of exercises and exercise types;
- direct solution checking by invoking the interpreter;
- ability to organise tasks into categories;
- offering the exercises gradually following the lectures.

Two basic concepts

- **category:** a logical group of problems;
- **scheme:** the way how a task is presented and checked.

3. Some Prolog categories

Standard prefix notation. Specify the *canonical form* of an expression made up of

- operators:
  Q: `6*t-j` **A:** `-(*(6,t),j)`
- lists:
  Q: `[1,2|A]` **A:** `.(1,.(2,A))`
- arbitrary compounds:
  Q: `g(G/H, [2/3+u|J])` **A:** `g(/(G,H), .(+(/(2,3),u), J))`

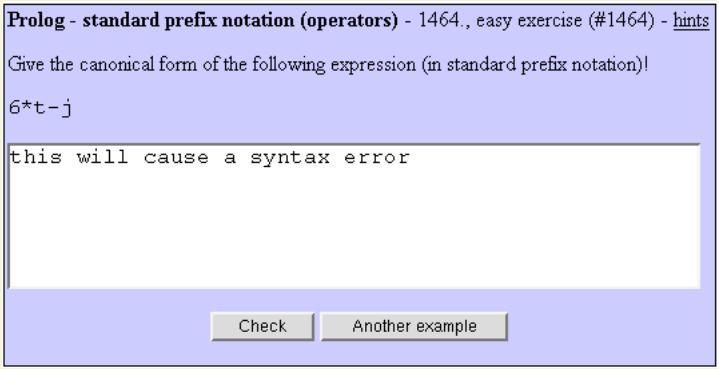
Unification. Specify the result (success/failure/error) of a Prolog unification, and in case of success, also specify the values of some or all variables.

- simple, only one variable:
  Q: `| ?- .(X,X) = [[]].` **A:** success, `X = []`
- advanced, several variables:
  Q: `.(U,[U,1]) = [E+2+3,F+G,E].` **A:** `E=1, F=1+2, G=3, U=1+2+3`

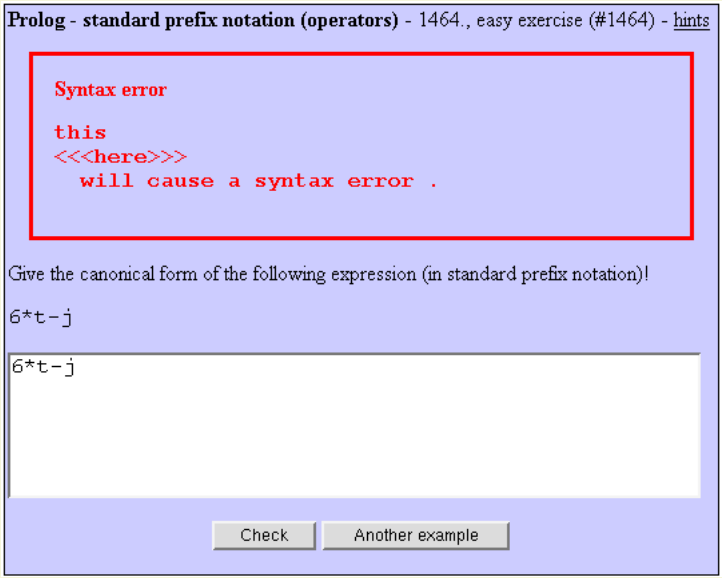
Programming. Write a simple Prolog predicate satisfying a given specification.

```
% longer(+L, ?S): the list L is longer than the list S
```

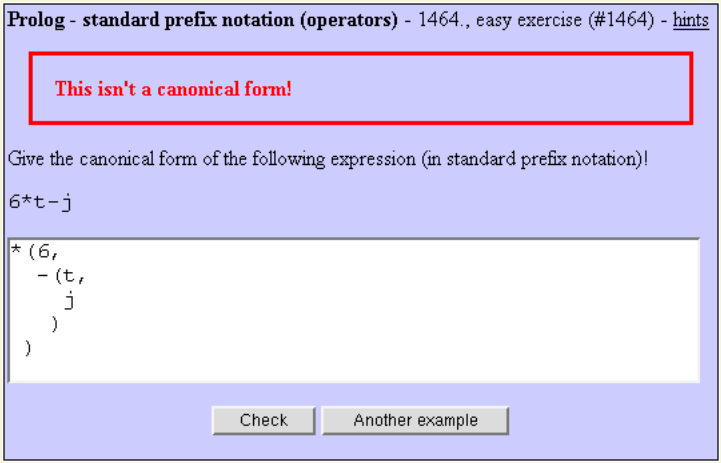
Standard prefix notation - figure 1



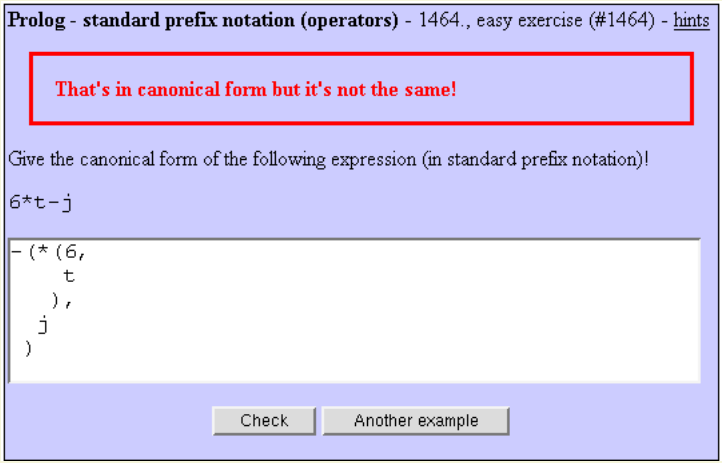
Standard prefix notation - figure 2



Standard prefix notation - figure 3



Standard prefix notation - figure 4



4. Some SML categories

Basic types and values. Given a tuple containing basic expressions, specify

- its value in its simplest form:
  Q: `8 :: 6+4 :: 9 div 3 :: nil` **A:** `[8,10,3]`
- its type:
  Q: `("o"^"r", op-(3,4), [[true]])` **A:** `string * int * bool list list`

Polymorphism. Handling data structures without knowing the type of their constituents in particular. Specify

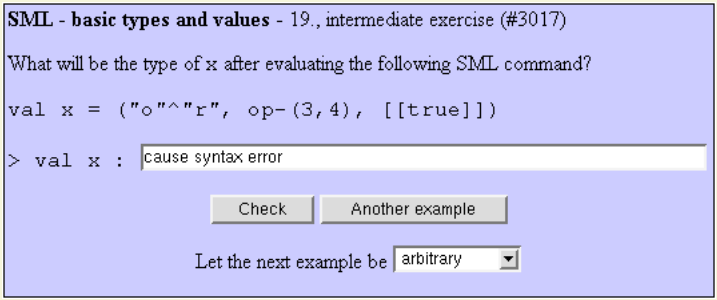
- a possible body with given type and head:
  Q: `val x : 'a -> ('a -> 'b) -> 'b; fun x y z = ?`
  A: `fun x y z = z y`
- a function definition given its specification:

```
(* lgr (l,ls) = 'ls' is longer than 'l' *
 * lgr : int * 'a list -> bool          *)
```

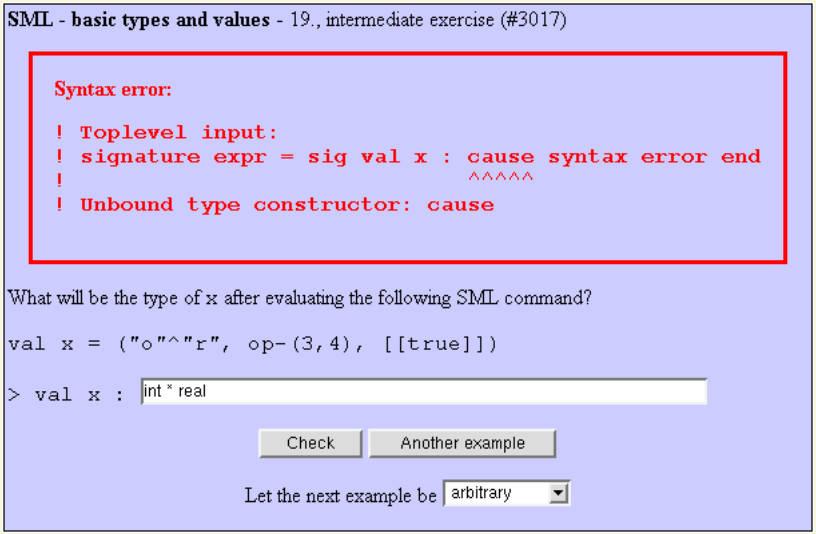
Higher order functions. Specify the type of an expression containing such functions.

Q: `foldr op=` **A:** `bool -> bool list -> bool`

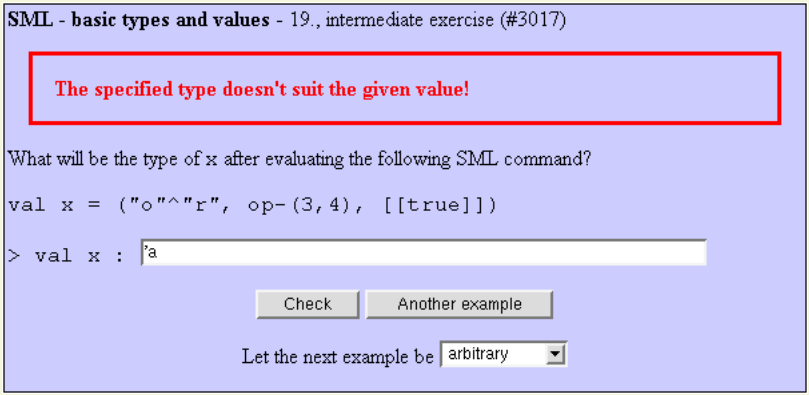
Basic types and values - figure 1



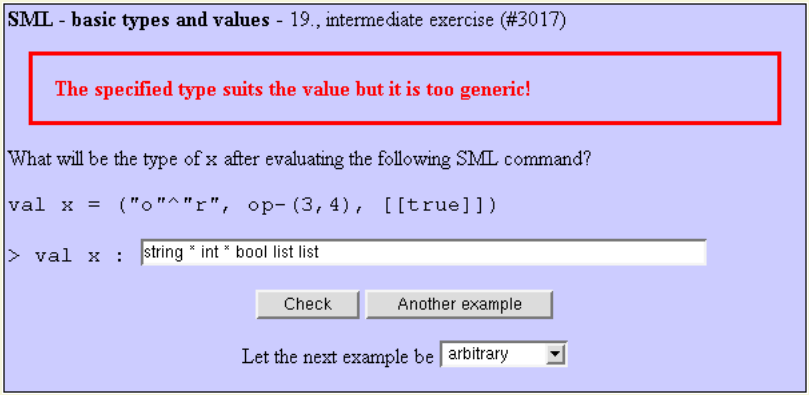
Basic types and values - figure 2



Basic types and values - figure 3



Basic types and values - figure 4



5. Schemes

Prolog schemes

Standard prefix notation: give the canonical form of an expression

Success/failure/error: give the result of a call, in case of success also determine the value of a specific variable

All solutions: enumerate (in proper order) all solutions of a goal, as returned in a specified variable

Programming: write a predicate satisfying a given specification

SML schemes

Type: determine the type of a declaration (value or function)

Value: determine the simplest form of the value of an expression

Function body: determine the body of a function if the head and the type is given

Type declaration: define a data type satisfying a specification

Programming: write a function conforming to a given specification

Acknowledgement

Thanks are due to

- all students who have helped us in the implementation of the ETS;
- especially to *Lukács Tamás Berki* and *András György Békés* for their work on the exercise system.

References

[1] Dávid Hanák: *Computer Support for Declarative Programming Courses* (in Hungarian), 2001, MSc Thesis, see also `http://dp.iit.bme.hu:4321/`

[2] András György Békés, Lukács Tamás Berki: *A Web-based Exercise System for Programming Languages* (in Hungarian), 2001, Students' Conference, Budapest, Hungary

[3] Péter Hanák, Péter Szeredi, Tamás Benkő, Dávid Hanák: *"Yourself, my lord, if no servants around" – A Web Based Intelligent Tutoring System* (in Hungarian), 2001, NETWORKSHOP01, Sopron, Hungary