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

Prolog - standard prefix notation (operators) - 1464, easy exercise (#1464) - hints

Give the canonical form of the following expression (in standard prefix notation):

6*t-j

this will cause a syntax error
Standard prefix notation - figure 2

Syntax error

this

<<here>>>

will cause a syntax error.

Give the canonical form of the following expression (in standard prefix notation):

6*t-j

6*t-j
This isn't a canonical form!

Give the canonical form of the following expression (in standard prefix notation):

6*t - j

* (6, 
   - (t, 
      j
      )
   )

Check  Another example
Standard prefix notation - figure 4

That's in canonical form but it's not the same!

Give the canonical form of the following expression (in standard prefix notation):

6*t-j

\[-(\ast(6, t), j)\]
4. Some SML categories

Basic types and values. Given a tuple containing basic expressions, specify
- its value in its simplest form:
  \[\text{Q: } 8 :: 6+4 :: 9 \text{ div } 3 :: \text{nil} \quad \text{A: } [8,10,3]\]
- its type:
  \[\text{Q: } ("o"\text{"r"}, \text{op-(3,4)}, [[[\text{true}]]]) \quad \text{A: } \text{string} * \text{int} * \text{bool list list}\]

Polymorphism. Handling data structures without knowing the type of their constituents in particular. Specify
- a possible body with given type and head:
  \[\text{Q: } \text{val x : } 'a \rightarrow ('a \rightarrow 'b) \rightarrow 'b; \text{fun x y z = ?} \quad \text{A: } \text{fun x y z = z y}\]
- a function definition given its specification:
  \[\begin{align*}
  &\text{(* lgr (l,ls) = ‘ls’ is longer than ‘l’ *)} \\
  &\text{(* lgr : int * ’a list \rightarrow bool *)}
  \end{align*}\]

Higher order functions. Specify the type of an expression containing such functions.
\[\text{Q: } \text{foldr op=} \quad \text{A: } \text{bool \rightarrow bool list \rightarrow bool}\]
Basic types and values - figure 1

SML - basic types and values - 19., intermediate exercise (#3017)

What will be the type of x after evaluating the following SML command?

```sml
val x = ("o"^"r", op=(3, 4), [[true]])
```

> val x: cause syntax error

Let the next example be arbitrary
Basic types and values - figure 2

SML - basic types and values - 19., intermediate exercise (#3017)

Syntax error:
! Top level input:
! signature expr = sig val x : cause syntax error end
! Unbound type constructor: cause

What will be the type of \( x \) after evaluating the following SML command?

\[
\text{val } x = ("\circ" ^ "r", \text{op-}(3,4), [[\text{true}]])
\]

Let the next example be arbitrary
What will be the type of x after evaluating the following SML command?

```sml
val x = ("o"^"r", op-(3, 4), [[true]])
> val x : 'a
```

Select from:
- Check
- Another example

Let the next example be:
- arbitrary
- basic
- function
- operator
Basic types and values - figure 4

The specified type suits the value but it is too generic!

What will be the type of \( x \) after evaluating the following SML command?

\[
\text{val } x = ("o","r", \text{ op~}(3, 4), [[true]])
\]

> val x : string * int * bool list list

Let the next example be arbitrary
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

