**UNIVERSITY OF NIŠ**
The scientific journal **FACTA UNIVERSITATIS**
*Series:* **Philosophy and Sociology** Vol.2, N° 7, 2000 pp. 357 - 365
*Editor of series: Gligorije Zaječaranović*
Address: Univerzitetski trg 2, 18000 Niš, YU
Tel: +381 18 547-095, Fax: +381 18 547-950

# THE LISD APPROACH

*UDC 37.022+510.62*

**Djordje Kadijević**

Mathematical Institute, Serbian Academy of Sciences and Arts,
Kneza Mihaila 35, 11001 Belgrade, p.p. 367, Yugoslavia
E-mail: djkadij@mi.sanu.ac.yu; URL: http://www.mi.sanu.ac.yu/~djkadij

**Abstract**. *This study presents a didactic approach to learning mathematics through knowledge engineering. This approach is realized through the development of expert system knowledge bases. The development is based upon rapid prototyping that is utilized through programming in logic and PROLOG.*

Although a number of constructivist educators have suggested knowledge engineering and the use of expert system shells as cognitive tools (see, for example, Jonassen, 1996), a corresponding didactic approach has not been developed. In order to remedy this gap, my PhD study developed such an approach called LISD. This paper presents LISD design, its prerequisites and their didactic sequencing, LISD application, and difficulties that may follow it. For order LISD issues, see Kadijevich (1999).

## DESIGN

The LISD approach is based upon two able methodologies for mathematical problem solving and software development: the heuristic approach of Pólya and Schoenfeld, and rapid protopying via programming in logic and PROLOG[*]. As LISD uses a framework for developing logic programs (Galle & Kovács, 1992), it comprises six phases: identification, conceptualization, formalization, implementation, verification and elaboration.

– *Identification*. Selecting an area and its problems. Solving the selected problems by

Received March 23, 1999

[*] http://www.mi.sanu.ac.yu/~djkadij/methods.htm

using the heuristic approach of Pólya and Schoenfeld. Realizing the relevant knowledge regarding the solved problems that will be the subject of intelligent computer tutoring.
- *Conceptualization*. Expressing the relevant knowledge by means of suitable objects, their properties and relations among them.
- *Formalization*. Writing a program in logic expressing the selected objects, properties and relations.
- *Implementation*. Transforming the program in logic into a PROLOG program (an if-then knowledge base), by taking into account its declarative features. A text-editor is used.
- *Verification*. Testing the correctness of the PROLOG knowledge base by taking into account its procedural behaviour. Improving the knowledge base according to the test outcomes (if need be). An expert system shell is used. #1
- *Elaboration*. Improving the knowledge base in respect of its optimization, solving a wider class of related problems, etc.

Since rapid prototyping is applied, these phases do not follow each other in the given order, but they repeat and overlap several times, according to the number of the developed prototypes #2. Note that Schoenfeld's control of problem solving based upon various metacognitive questions — e.g., "What are you doing", "Why are you doing that?", and "How will this help you to solve the task?" — should be utilized in all phases through pair work where one student acts as the solver, whereas the other acts as the solver's external monitor (the students should exchange their roles regularly).

## PREREQUISITES AND THEIR DIDACTIC SEQUENCING

An effective LISD utilization is based upon an effective realization each of the following topics:
1. the heuristic approach of Pólya,
2. Schoenfeld's control of problem solving,
3. expert systems,
4. knowledge representation by an if-then formalism through programming in logic and PROLOG.

Our experience suggests that efficient LISD teaching/learning can be achieved by utilizing the following didactic steps:
1. Introduce the heuristic approach of Pólya within 3-4 hours. Present a number of examples regarding the use of specialization, generalization and analogy, and explain several ways of solving some of the considered problems.
2. Familiarize students with Schoenfeld's control of problem solving within 2-3 hours. Encourage them to play patiently their roles of the solver and the external monitor.
3. Introduce the features, architecture and usage of expert systems within 2 hours. Encourage students to consult an expert system regarding a suitable mathematical topic.

4. Analyze the knowledge base of the consulted expert system whose shell supports an if-then formalism for representing knowledge. Encourage students to: (a) apply this formalism to simple knowledge items, (b) develop facts and rules (programming in logic), (c) create simple knowledge bases (programming in PROLOG) by using a text-editor, and (d) consult (test) these bases by using an expert-system shell supporting the utilized formalism. (3-4 hours)
5. Require students to develop small-scale knowledge bases for chosen problems according to the LISD design. Encourage them to use hierarchical rules. (Recursive rules should be avoided, at least in the beginning.)

It is important to underline that the LISD students do create knowledge bases through programming in PROLOG, but this programming need not be made explicit nor be widely applied. This is because the LISD utilization only requires knowledge regarding the shell formalism and the way it unifies terms and fires rules. This knowledge, which should be explained to the students when they encounter an inadequate procedural behaviour of the developed knowledge bases, is likely to cause only minor learning difficulties (Kadijevich, 1998).

## APPLICATION

In order to illustrate LISD application, let us examine a way of creating two simple prototypes regarding one object uniform motion, keeping in mind that the developed rules do not solve problems but express strategies whereby some classes of problems on motion can be solved. The emphasis is thus on developing and using algorithms to solve problems, which requires students "to give qualitative explanations of principles and make direct inferences from them without referring to the results of numerical calculations" (Lippert, 1990; p. 29). Note that although some expert systems allow numerical data input, expert systems in general usually do not perform numerical computation.

### prototype #1

- *Identification*. Some examples are ... They can be solved by ... Thus, in simple problems on one object uniform motion we deal with three variables - speed, travelled distance and travelling time - one of which is to be determinated by means of the others.
- *Conceptualization*. In all such tasks we have one objects (car, cyclist, pedestrian, etc.), which has three properties (speed, travelled distance and trevelling time) connected with a relation $s = d / t$.
- *Formalization*. If an object and its speed and travelling time are known, a corresponding program in logic may have the following form:

```
answer(object_travelled_distance = object_speed *
          object_travelling_time) if
known  object             and
known  object_speed       and
known  object_travelling_time.
```

Similar programs can easily be written if an object and its speed and travelled distance are known, or if an object and its travelled distance and travelling time are known.

– *Implementation*. Having in mind the chosen formalism for representing knowledge, our knowledge base may take the form (lines with comments begin with %):

```
% --------------------  rules  --------------------

1 rule
if     object(X)  and
       speed(X)   and
       travelling_time(X)
then   answer(travelled_distance(X) =  speed(X) *
                    travelling_time(X)).
% X stands for any objects (car, bicycle, train, etc.);
% speed(X) denotes that the speed of a specified object X
% is known

2 rule
if     object(X)  and
       speed(X)   and
       travelled_distance(X)
then   answer(travelling_time(X) =  travelled_distance(X)
                                  / speed(X)).

3 rule
if     object(X)              and
       travelled_distance(X)  and
       travelling_time(X)
then   answer(speed(X) =  travelled_distance(X) /
                    travelling_time(X)).

% --------------------  facts  --------------------
% how many objects are treated by the system
objects(1).

% which data may be asked by the system
askable(speed(X)).
askable(travelling_time(X)).
askable(travelled_distance(X)).

% explanations regarding the conceptualization/activation
% of the developed rules
help(1, [$ ... some text regarding rule 1 ...$]).
help(2, [$ ... some text regarding rule 2 ...$]).
help(3, [$ ... some text regarding rule 3 ...$]).
```

– *Verification*. It is easy to check that the listed program has a correct procedural

behaviour. If the student's logic program is correct, an incorrect procedural behaviour of its PROLOG program is usually caused by: (a) some typographical errors, (b) the use of some functors that are unrecognizable by the shell (use "askable" and "help" in the introduced format), (c) creating a rule containing askable(X) instead of X only, and (d) some syntax errors in respect of the shell's notation of facts and rules. These errors can be, in most cases, easily located and corrected.

– *Elaboration*. Extend the system's possibilities having in mind that an object's time coordinate is frequently given by two of the following variables: travelling time, starting time and finishing time.

**prototype #2**

– *Identification*. Some examples are ... They can be solved by ... Thus, as regards an object's time coordinate, we deal with three variables - travelling time, starting time and finishing time - one of which is to be determinated by means of the others.
– *Conceptualization*. An object's new properties are travelling time, starting time and finishing time, which are connected with a relation $t = t_f - t_s$.
– *Formalization*. If an object and its starting time and finishing time are known, a corresponding program in logic may have the following form:

```
answer(object_travelling_time = object_finishing_time -
            object_starting_time) if
known  object                 and
known  object_finishing_time  and
known  object_starting_time.
```

Similar programs can easily be written if an object and its starting time and travelling time are known, or if an object and its travelling time and finishing time are known.

– *Implementation*. Prototype #2 can be realized by adding to prototype #1 the following rules and facts:

```
% --------------------  rules  --------------------
% ... rules of prototype #1

4 rule
if     object(X)                    and
       starting_time(X)             and
       travel_duration(X) equals T  and
       T \== (finishing_time(X) - starting_time(X))
         % the last condition prevents the occurance
         % of the answer:
         % finishing_time(X) = starting_time(X) +
         % (finishing_time(X) - starting_time(X)).
then   answer(finishing_time(X) = starting_time(X) + T).

5 rule
if     object(X)                    and
       finishing_time(X)            and
```

```
        travel_duration(X) equals T  and
        T \== (finishing_time(X) - starting_time(X))
          % the last condition prevents the occurance
          % of the answer:
          % starting_time(X) = finishing_time(X) -
          % (finishing_time(X) - starting_time(X)).
then    answer(starting_time(X) = finishing_time(X) - T).

6 rule
if      object(X)           and
        travelling_time(X)
then    travel_duration(X) equals travelling_time(X).

7 rule
if      object(X)           and
        starting_time(X)   and
        finishing_time(X)
then    travel_duration(X) equals (finishing_time(X) -
                          starting_time(X)).

8 rule
if      object(X)  and
        speed(X)   and
        travelled_distance(X)
then    travel_duration(X) equals travelled_distance(X) /
                                  speed(X).

% --------------------  facts  --------------------
% ... facts of prototype #1

askable(starting_time(X)).
askable(finishing_time(X)).

help(4, [$ ... some text regarding rule 4 ...$]).
help(5, [$ ... some text regarding rule 5 ...$]).
help(6, [$ ... some text regarding rule 6 ...$]).
help(7, [$ ... some text regarding rule 7 ...$]).
help(8, [$ ... some text regarding rule 8 ...$]).
```

Having in mind that travelling time can be given in two ways, rules 1 and 3 of prototype #1 are to be modified. To achieve this end, the following code can be used:

```
1 rule
if      object(X)  and
        speed(X)   and
        travel_duration(X) equals T
then    answer(travelled_distance(X) = speed(X) * T).
```

```
3 rule
if     object(X)              and
       travelled_distance(X)  and
       travel_duration(X) equals T
then   answer(speed(X) = travelled_distance(X) / T).
```

– *Verification*. An interpretation of the developed knowledge base is not possible until we tell PROLOG system that "equals" is an user-defined operator. As "equals" corresponds to the built-in operator "=", it is natural to use the following definition:

```
:- op(700, xfx, equals).
```

(This user defined operator is not needed if we use "=" in a way that prevents unification, such as: solve(travel_duration(X) = T). Of course, instead of "solve", we can use another functor.) Even after this definition has been made, a correct interpretation of the knowledge bases is still not possible. This is because the shell, or PROLOG interpretator to be more precise, considers the developed rules in respect of their order (a rule's number is irrelevant from a point of its utilization). The rules therefore are to be rewritten. The following way is required: 4, 5, 1, 2, 3, 7, 8 and 6. As regards rules order in general, a rule of thumb is: in a set of related rules, more specific rules precede others; in a whole knowledge base, main rule precede subordinate rules.

– *Elaboration*. Extend prototype #2 in order to handle one object uniform motion in respect of a reference point.

## DIFFICULTIES

The considered prototypes clearly illustrate two groups of difficulties relevant to the phases of implementation and verification. The first group is caused by: (a) neglecting the shell's formalism - the chosen syntax for writing facts and rules, and (b) writing facts and rules carelessly. The second one is generated by an unsuitable order of the developed rules causing incorrect problem solving activities. Although both groups of difficulties may discourage students from experimenting with knowledge engineering, they still can be eliminated to a great extent with some teacher's help, especially when we use a shell that can display the way in which the rules are fired.

Various difficulties can also occur in other phases as they require students to: (a) generate and solve problems, (b) externalize, formalize and represent ways of solving these problems, and (c) optimize the content of the developed knowledge bases. Although they all can be carefully treated, our experience suggests that special attention should be given to writing rules using other rules. Of course, an introductory LISD utilization should not require the creation of recursive rules and optimizations based upon the compression of a number of rules into a few rules, because these requirements call for advanced cognitive and metacognitive activities.

Undoubtedly, an effective LISD utilization requires versatile cognitive and metacognitive activities. It therefore seems that the LISD approach is only suitable for those students whose intellectual and mathematical abilities are above average**#3**. Although the impact of the student's personal variables on his/her LISD outcome has not been studied so far, it is quite clear that only a few students, perhaps even in an able class,

can successfully act as knowledge engineers. What about other students? They can consult the developed knowledge bases in order to comprehend the ways whereby some related problems are solved by able students of the same age. This kind of monitoring may be valuable to many students, especially to those who can successfully do mathematics when some help is provided by the teacher. Although some empirical evidence regarding this issue is still needed, it was found that social interactions between students who were spontaneous experts and students who were novices were profitable for both kinds of student in respect of their learning outcomes (Marro Clement *et al*, 1998).

## NOTES

#1 The shell supporting an if-then formalism for representing knowledge should allow the use of the following commands: solve (solve the problem), how (show how the problem has been solved), why (show the underlying rule), help (show the procedural and/or conceptual background of the underlying rule), tron (show the way in which the rules are fired), troff (turn the tracing facility off), and list (list the content of the working storage - a temporary knowledge base created from the user's answers).

#2 These phases need not be introduced to the LISD students. If they have been introduced, the students should be given some freedom to utilize them as conceptualization, formalization and implementation may be utilized in parallel as a single phase.

#3 It is true that almost any method (including traditional lecturing) would work for able students, but LISD contrary to most methods does provide less able students with an intelligent assistant, which explains to them the strategies of solving problems within a particular area that have been realized by their able classmates.

## REFERENCES

1. Galle, P. & Kovács, L. B. (1992). *The logic of worms: a study in architectural knowledge representation*. Environment and Planning B: Planning and Design, **19**, 5-31.
2. Jonassen, D. H. (1996). *Computers in the classroom: Mindtools for critical thinking*. Englewood Cliffs, NJ: Prentice-Hall.
3. Kadijevich, Dj. (1998). *Can Mathematics Students Be Successful Knowledge Engineers?* Journal of Interactive Learning Research, **9**, 235-248.
4. Kadijevich, j. (1999). An approach to learning mathematics through knowledge engineering. Journal of Computer Assisted Learning, 15, 4, 291-301.
5. Lippert, R. (1990). *Teaching Problem Solving in Mathematics and Science with Expert Systems*. Journal of Artificial Intelligence in Education, **1**(3), 27-40.
6. Marro Clement, P., Perret-Clermont, A.-N., Grossen, M. & Trognon, A. (1998). *Le partenaire comme enseignant ou comme interlocuteur: une analyse expérimentale et interlocutoire*. Cahiers de Psychologie, no 34. Séminaire de Psychologie, Neuchâtel.

# UČENJE MATEMATIKE PUTEM REŠAVANJA PROBLEMA KROZ PROJEKTOVANJE INTELIGENTNIH TUTORSKIH PROGRAMA

**Djordje Kadijević**

*Imajući u vidu osobenosti Poljinog i Šenfeldovog načina rešavanja problema, pedagoške vrednosti inteligentnih tutorskih programa i način projektovanja ovih programa brzim prototipovanjem kroz programiranje u logici i PROLOG jeziku, rešavanje problema, koje je u skladu sa Poljinim i Šenfeldovim pristupom, moglo bi biti realizovano kroz projektovanje inteligentnih tutorskih programa u kome učestvuju učenici i njihovi nastavnici. Ovakav način realizacije problemske nastave matematike može se bazirati na metodičkom pristupu koga razmatramo u ovom poglavlju.*