APPLICATION OF E-LEARNING IN PROGRAMMING LANGUAGES THEORY

Filip Goldefus[1)], Ota Jirák[2)]
Brno University of Technology, Faculty of Information Technology, Božetěchova 2, 612 66 Brno, Czech Republic
igoldefu@fit.vutbr.cz[1)], ijirak@fit.vutbr.cz[2)]
www.fit.vutbr.cz/~igoldefu/, www.fit.vutbr.cz/~ijirak/

Abstract. This paper summarizes the present state of learning of programming languages on Faculty of Information Technology, Brno University of Technology. The survey into the new methods of teaching programming languages, especially interactive support for teaching functional and logic programming, is given in this paper. Main goals of theoretical introduction lead to practical application of theoretical methods, implementation of software project. Proposed e-learning course contains theoretical part and practical parts with Prolog programming language. The generated set of examples and applications are employed in education. Finally, improvements for future development of the e-learning of programming languages are listed.

Keywords: programming languages, spaced repetition, lambda calculus, synchronous role, asynchronous role, Prolog, SLD resolution

1. INTRODUCTION

The knowledge of programming languages on Faculty of Information Technology, Brno University of Technology is a main part of profile of graduated students. This faculty deals with inconsistency of e-learning methodology, because there is no unified framework for creating and managing e-learning courses contrary of the Masaryk University of Brno, where the unified framework is provided as a part of web information system. The teacher of e-learning course is able to create and manage the courses. The uniform e-course framework leads to unified evaluation of students and preparation of courses with commonly known components and graphical user interface.

There are three types of communication in every e-course for building and sustaining communities, or types of information exchange respectively.

- Content-related exchange of information.
- Planning of tasks.
- Social support.

This paper deals only with the content-related exchange of information and partially by planning of e-learning task, which is a result of full specification of grant project. This project creates interactive support for teaching programming languages.

The principal part of solution of grant project called *Interactive support for teaching functional and logic programming* is creation of software application using Adobe Flash[1] and Action Script 3. Modeling of this application uses strict methods of software design, e. g. programming in pairs, design of entity relationship diagram, diagram of functions and dataflow diagram.

[1] http://www.adobe.com/

2. THEORETICAL INTRODUCTION

Aspects of modeling of an e-learning software application are given in this part of paper. The realized software application is oriented on asynchronous type of preparation for the teacher, which means, preparation of the course contents by building database with resources and programming examples. The student's role is synchronous, course is interactive and the evaluation of answers is immediate. There is a test verifying student's knowledge of subject area at the end of learning session.

There are two main areas of the whole system. The first one contains e-learning application called *flashcards* (see [2]). This application is based on principles used for memorize vocabulary words from classes of foreign languages. The second area is focused on logical programming and visualization of SLD resolution.

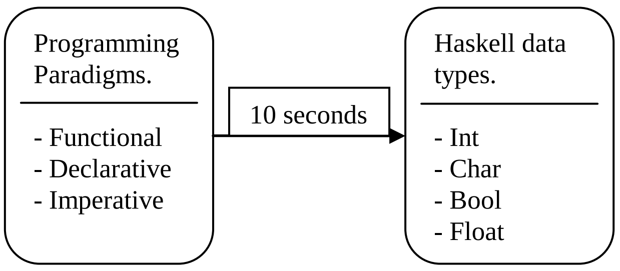

Fig. 1.Passive review of flashcards, where two consequent cards are shown in a learning session (in 10 seconds interval).

Flashcard type of e-learning

Flashcard type of e-learning is based on a random repetition of cards with some information. Course content is stored in data structure called *vocabulary* or *database of facts.* This information should be memorized by students. The vocabulary is divided into lessons by topics and by their difficulty. Every card is shown for a particular time, the easy memorable information is shown for a short time, and once during given learning session. Hard to remember information is shown for a long time, and two or more times during given learning session.

After several series of cards (see Fig. 1), there is a quiz (see Fig. 2), discovering student's knowledge of presented vocabulary. This method is sometimes called *spaced repetition*.

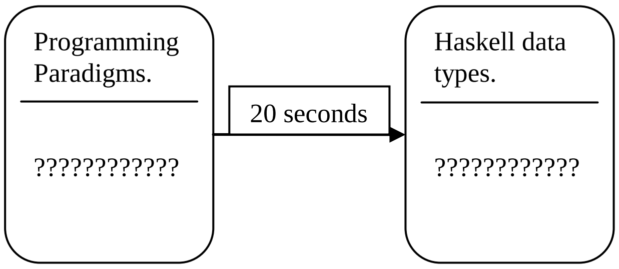

Fig. 2. Active recall of flashcards, where two consequent cards are shown in a testing session (in 20 seconds interval).

The time between repetitions of subsequent reviews is increased. There is a field for experimenting, how to set up the suitable periods of time for showing one flashcard, frequency of showing same card in one learning session, and the number of consequent cards for passive review followed by questions for active recall.

In [1], there are several hypotheses for determination of the frequency called:

- strength hypothesis,
- multiple-trace hypothesis,
- propositional encoding hypothesis.

The *Strength* hypothesis assumes that the increasing number of repetitions leads to strengthen memory. Every learning session enhances student's memory. Every repetition adds quantitative measurable amount of change in memory between two course lessons. This theory treats with cumulative frequency of repetition, depending on individual student cognitive abilities. The frequency is altered by obtaining successful or unsuccessful questions during the active recall phase.

The multiple-trace hypothesis assumes:

- Each presentation of information leaves its own trace in student's memory.
- Every trace of different presentations of same information coexists with another one in memory.
- Traces of different presentations of the same information can carry different attributes.

The *propositional encoding* hypothesis is that information per se is accumulated in propositional form during study of the list.

In the following text, only the strength hypothesis is used, because the multiple-trace and the propositional encoding hypotheses have no effect on design of e-course preparation and software application.

There is a problem, how to choose information for spaced repetition and massed repetition. The spaced repetition uses big stack of flashcards for every learning session. The massed repetition uses more stacks with fewer amounts of flashcards used in a learning session. There were many experiments exploring the dependency between type of information, and number of learning sessions (see [2]).

Example. The spaced condition corresponds to the strategy of using a big stack of 10 flashcards; the massed condition corresponded to the strategy of splitting flashcards into five smaller stacks of five cards each. There are two 100 seconds sessions, divided by 5 minute pause. They are followed by testing of all cards. After two sessions of passive review, where all flashcards for spaced repetition and all flashcards for massed repetition sequentially displayed (see Fig. 3).

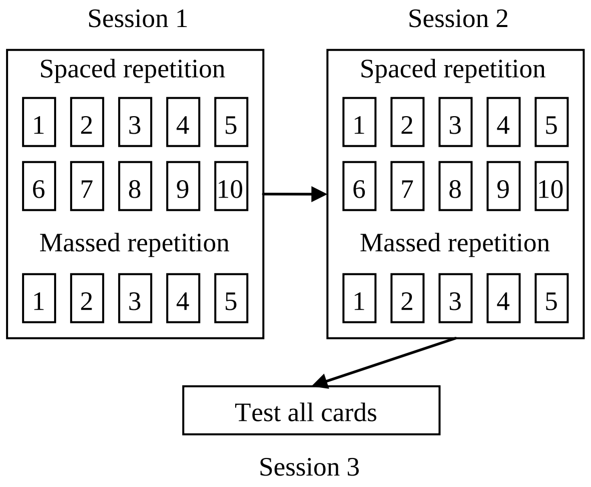

Fig. 3. Example of one learning course using 10 spaced time flashcards and 10 massed flashcards.

In [2], it was proved that spacing repetition is more effective than massed type of repetition. Experiments similar to previous example were used.

In experiments [3] with learning foreign languages was discovered dependence of questioning after learning a word. This experiment led to discovery of trace of probability of correct response, which declines dependence on time. Relation between time and successful response is illustrated on Fig. 4.

There are standalone software applications based on spaced repetition generally usable on any e-learning course, e. g.: Anki[2], Mnemosyne[3] and SuperMemo[4].

All implementations have limited possibilities of changing time intervals and other important attributes for spaced repetition.

It is important to divide e-learning course into several parts. As a result of previous theoretical premises we split the courses into: (1) memory part, and (2) creative part.

The first part contains "course vocabulary", information designated to be memorized.

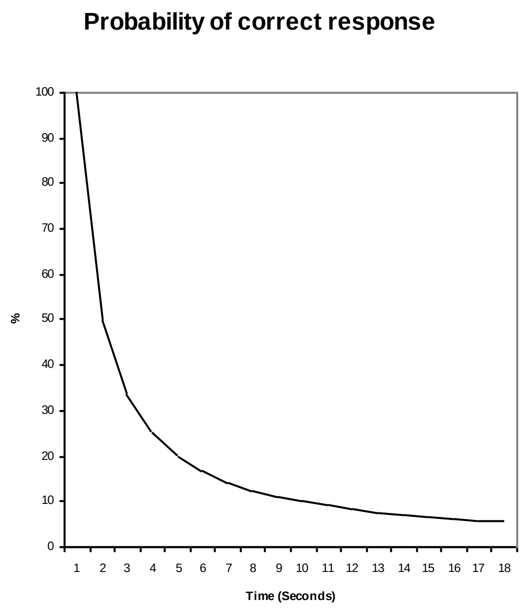

Fig. 4. Graph of probability of correct response depending on time after passive review of information [3].

Remaining part of e-learning course information is used in interactive part of software application. This learning section is more content dependable, and it is part of the following subsection.

<u>SLD resolution and visualization</u>
This section deals with preparation of content-dependent parts of e-learning courses focused on programming languages. Representation of an essential part of software application follows.

The visualization part of application consists of two segments:

- Visualization of Selective Linear Definite (SLD) clause resolution.
- Visual completion of programming code.

[2] http://ichi2.net/anki/
[3] http://www.mnemosyne-proj.org/
[4] http://www.supermemo.com/

The SLD resolution is the main principle used in logic programming. The advantage of the SLD resolution is systematic inference of rules. The SLD resolution is general process of mathematical logic used to inference conclusions.

The SLD resolution of a formula F is a sequention of tuples $<C_0, B_0>, \ldots, <C_n, B_n>$. Every C_{i+1} is resolvent of $<C_n, B_n>$ and $C_{n+1} = F$. For more details see [5].

The input for the resolution is a set (potentially an empty set) of sets of clauses – statements with variables. Clauses are defined as a disjunction (sequence of logical disjunctions) of literals, e.g. clause

$$F_1 = p \vee \neg q \vee r.$$

Literals are atomic formulas or their negations (p, $\neg q$, r). We can say that literals or clauses take the values true or false in logic programming. Logic programming language Prolog[5] uses special type of clauses, called *Horn clauses*. A Horn clause contains at most one positive literal, so the clause F_1 is not a Horn clause.

Every Prolog program contains: (1) database of facts, (2) rules.

Example of Prolog database of facts follows.

```
mother_child(trude, sally).
father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).
```

The same facts represented as Horn clauses:

```
{mother_child(trude, sally)},
{father_child(tom, sally)},
{father_child(tom, erica)},
{father_child(mike, tom)}.
```

The second part includes rules:

```
sibling(X, Y) :- parent_child(Z, X),
parent_child(Z, Y).
parent_child(X, Y) :- father_child(X,
Y).
parent_child(X, Y) :- mother_child(X,
Y).
```

Previous rules represent follows Horn clauses.

```
{sibling(X, Y), ¬parent_child(Z, X),
¬parent_child(Z, Y)},
{parent_child(X, Y), ¬father_child(X,
Y)},
{parent_child(X, Y), ¬mother_child(X,
Y)}.
```

[5] http://www.swi-prolog.org/

Practicing of the SLD resolution is in the style "fill the gaps." The questions contain parts of programs and student has to fill it in. He chooses the right literals and variables to complete the code to perform required results. Every resolution step will be animated. Student is asked for the right substitution of variables *(X, Y, Z)* and result of resolution step.

Example. The following set of clauses is given,

$$C = \{\{P(a,b)\}, P(X,X), \{P(Z,X), \neg P(Y,Z)\}\}.$$

SLD resolution has to find a substitution for variables *X, Y, Z* to satisfy the goal *{¬P(Y, a)}* using clauses from *C*.

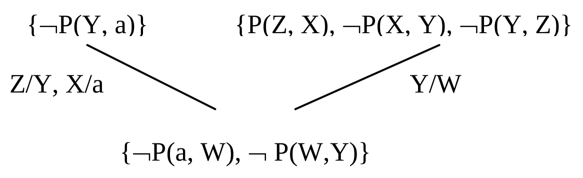

Fig. 5. One step of SLD resolution, creation of a resolvent from clauses.

Example of one step of SLD resolution for clause *{¬P(Y, a)}* and set of clauses *C* is on Fig. 5.

SLD resolution can be visualized as a tree where nodes are clauses and edges are labeled by substitutions for variables. E.g. *X/a* corresponds to substitution of variable *X* by value *a*.

4. IMPLEMENTATION

The implementation is based on Adobe Flash with Action Script. This technology can be used in most web browsers on common operation systems like Windows, Linux, etc.

The whole system is divided into two main sections:

- flashcards,
- SLD resolution visualization.

These parts have similar architecture depicted on Fig. 6.

The architecture is divided into several parts. The first, called *loader*, loads examples from:

- predefined files,
- uploaded files,
- textbox.

The second part transforms example/course lesson, stored in special format, into inner representation.

The third part returns semi code representing SLD resolution steps.

The last part handles visualization and GUI controls.

As we can see, this type of architecture enables us to change particular parts to add new type of examples.

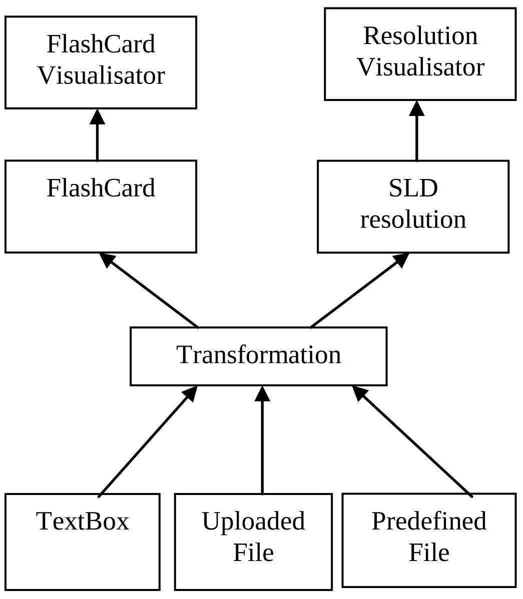

Fig. 6. System architecture.

5. CONCLUSION

The part of grant project related to logic programming is completed; the functional programming section remains, but will contain similar units (flashcards, animated resolutions) as a logic programming section.

The functional programming section similar to SLD resolution involves Lambda Calculus for fundamental principles of functional programming. Haskell[6] is functional language chosen for code examples.

Main interest of future research is aimed to experiments with flashcard application part, exploration of periods of time for each card, and selection of proper Haskell code examples.

Experiments include students activities with e-learning course and their classification.

6. REFERENCES

[1] Hintzman, D. L.: Repetition and Memory, In: Gordon H. Bower, Editor(s), Psychology of Learning and Motivation. Academic Press, 1976, Volume 10, pp. 47-91.

[2] Kornell, N.: Optimizing learning using flashcards: Spacing is more effective than cramming. Applied Cognitive Psychology, John Wiley & Sons, 2009, pp. 1297-1317.

[6] http://www.haskell.org/

[3] Pimsleur, P.: A Memory Schedule. The Modern Language Journal, Blackwell Publishing, 1967, pp. 73-75.

[4] Hankin, Ch.: An Introduction To Lambda Calculi For Computer Scientists. King's College Publications London, 2004, 166 pages, 0-9543006-5-3.

[5] Sterling, L., Saphiro, E.: The Art of Prolog. MIT Press, 1994, 549 pages.

THE AUTHORS

Goldefus Filip:
- Profession: 3rd grade of postgraduate studium BUT Brno, Faculty of Information Technology
- Scientific activity: theoretical computer science, formal languages, compilers

Jirák Ota:
- Profession: 3rd grade of postgraduate studium BUT Brno, Faculty of Information Technology
- Scientific activity: formal languages, automata, compilers, lambda calculus.

ACKNOWLEDGEMENT

This work was partially supported by the BUT grant FIT-10-S-2, the research plan MSM 00021630528, and the FRVŠ grant FR2392/2010/G1.