THE ROLE OF INDUCTIVE INFERENCE IN THE DESIGN OF INTELLIGENT TUTORING SYSTEMS

Ladislav SAMUELIS, Ľubomír FAŠIANOK
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Slovak Republic, Letná 9, 042 00 Košice, tel. 055/602 4313,
E-mail: Ladislav.Samuelis@tuke.sk, Lubomir.Fasianok@tuke.sk

SUMMARY

The ultimate goal of recent Intelligent Tutoring Systems (ITSs) is to personalize user interaction with online course material. The aim of this contribution is to point out the role of the application of inductive inference based algorithms in the evolution of present ITS. In particular it focuses on the utilization of a special simple (inductive inference) incremental algorithm and discusses its time-complexity features. Finally, it introduces correspondence (analogy) between the inductive inference by example concepts and the concepts of the Shareable Content Object Reference Model (SCORM).

Keywords: *intelligent tutoring systems, inductive inference, programming by examples, SCORM*

1. INTRODUCTION

The history of the ICAI (Intelligent Computer Assisted Instruction) is about 25 years old [1]. The history of the programming by examples (it is considered as a branch of the machine learning [8]) has approximately the same age [2]. It is obvious that the research results obtained in both branches influenced each other in the history. The applicability of the artificial intelligence (especially machine learning) tools in software engineering played important role in the short history of both sciences. This influence continues in the Internet era further.

The aim of this contribution is twofold. On the one hand, we analyze the incremental algorithm introduced in the literature [4] and then evaluate its time complexity features. On the other hand, we try to raise the awareness of the applicability of already known techniques and concepts (of the programming by examples paradigm) in novel Internet based applications like ITSs.

In order to achieve these goals, we introduce the role of the examples in the program synthesis paradigm and then we focus to the description of the time complexity features of a particular algorithm.

At the end we sketch the analogy between the concepts of program synthesis by examples and the concepts of the SCORM standards specifications.

2. PRINCIPLES OF PROGRAMMING BY EXAMPLES

There are two main streams of automating the synthesis of programs:

1. Synthesis of correct programs by formal specifications. The main feature of this approach is to generate correct programs by implementing correct transformations [6].
2. Synthesis of programs by examples. This means that the user knows a part of the behavior of the algorithm and the task is to construct program. From artificial intelligence point of view these algorithms belong to the inductive inference mechanisms. An overview of programming by examples is described in [2].

There exist many areas when the demonstration is a suitable tool for automating tasks. E.g. paths of robots represent linear plan and the task is to construct program or the sequence of learning objects represent the progress of the student in the learning material and the task is to construct the navigation plan (learning by watching). The structures of the systems devoted to synthesis of programs by examples are similar to the structure of linguistic pattern recognition systems.

In the next sections we demonstrate the interactive process between the students and the ITS on an example.

- The student has at disposal a set of conditions and operations.
- The student defines a trace using the conditions and instructions. This trace is stored for later processing.
- The system synthesizes the minimal possible program (i.e. a set of transitions and states), which coincide with the given trace.
- If the synthesized program does not correspond to the intended trace, it is possible to correct the trace and return back to the synthesis process again.

The above-mentioned steps represent in fact rules and the task is to infer a grammar, which is a simplified method of machine learning. Question of correct interpretation of the system's input in order to design appropriate hypothesis (or next step) is also called incremental learning. The roots of the incremental learning lie in the programming by examples topic, which were elaborated in years 1970 and 1980 [2].

In the next paragraphs we describe formally the incremental algorithm, demonstrate its mechanism

on an example and then analyze its time complexity for the worst input.

The aim of the synthesis is to construct a minimal final deterministic automaton with branches and loops, which are expressed as:

$$I_1 \xrightarrow{c_1} I_2$$
$$I_2 \xrightarrow{c_2} I_3$$

where I_1, I_2 and I_3 are the instructions of the trace and c_1 and c_2 represent the conditions for the execution of the respective instructions. In this model the program equals to the regular grammar:

$$(V_n, V_t, D, I_0)$$

where

V_n - is the set of program instructions (non terminal symbols)

V_t - is set of conditions, which belong to the appropriate transitions between the program instructions (set of terminal symbols)

D - is set of rules, which does not contain 2 or more rules with the same left side

I_0 - is the start non terminal symbol

The algorithm for building the model is summarized as follows. Let the symbol P be a set of available instructions, which are necessary for constructing the example.

$$P = \{I_1, I_2, ..., I_K\}$$

We introduce notation $[I_j]$, for the set of equal instructions $I_j \mid 1 \le j \le K$. It is valid that

$$[I_j] = \{1I_j, 2I_j, ..., X_j I_j\}$$

where the integers in front of I_j s are called labels.

Let the overall number of I_j s in the model equal to $\left\| \left[I_j \right] \right\|$ and the $\left[I_j \right]^*$ is the actual number of $\left[I_j \right]$. The number of the total instructions c_2 in the model is:

$$L = \sum_{j=1}^{K} |[\mathrm{I_j}]|$$

where the number of various types of instructions is K. Because the value of the L is varying during the synthesis, we introduce the L^* for the actual value of L. Then

$$L^* = \sum_{j=1}^{K} |[\mathrm{I_j}]^*|$$

"Step" in the example is defined as a pair of (c_p, I_q). Different steps (l) may contain the same pairs, i.e. the same condition c_p and same instruction I_q. That is why we introduce the notion of $N(l)$ for the condition and the $O(l)$ for the instruction in certain step l. The $u(l)$ will denote the label of the instruction $O(l)$.

The principle of the program synthesis is in searching the value of $u(l)$, which will fulfill the following conditions:

1. The number of instructions L in the program is minimal and it is true that $K \le L \le M$, where M is the maximum number of instructions of the example.
2. If the M is the maximum number of instructions of the example, then during the synthesis it is necessary to assign a label $u(l)$ to every instruction $O(l)$ of the example and at the same time to achieve deterministic flow of control. I.e. for every step i where $i \le l$, and $O(i-1) = O(l-1)$, $u(i-1) = u(l-1)$ and $N(i) = N(l)$, then either
   a) $O(l) = O(i)$ and in this case it is possible to provide merging, i.e. $u(l) = u(i)$ or
   b) the above-mentioned conditions are not true and $O(l) = O(i)$, then new node has to be created, i.e. $u(l) \neq u(i)$ for the respective instructions in $O(i)$ and $O(l)$. This creation of the new node has to be done in order to secure the deterministic control of flow.

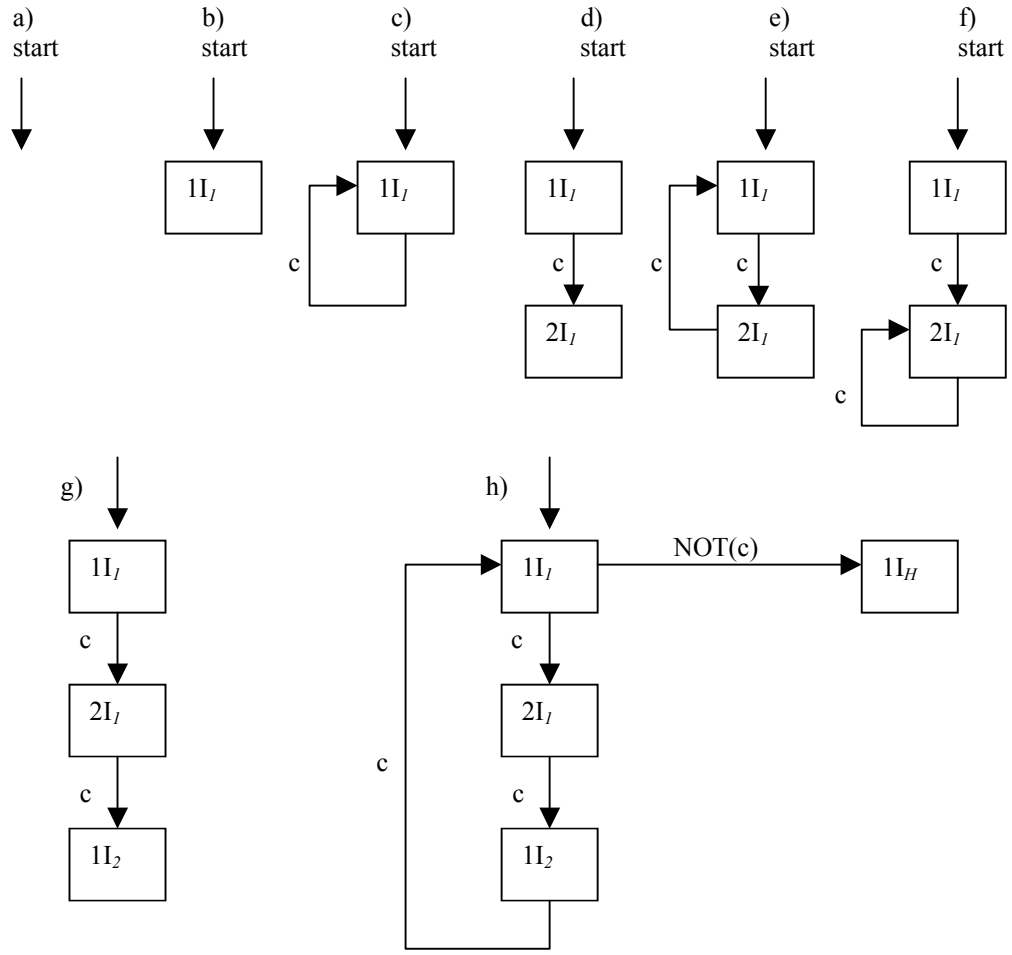
It is evident that when there does not exist a node in the model, which is merge able with the given instruction in the example, then new node has to be created.

3. EXAMPLE FOR THE INCREMENTAL ALGORITHM

We will illustrate the mechanism of the incremental algorithm on the following example. Let the trace T be defined by a sequence of instructions I_1, I_1, I_2 and I_H. The last instruction I_H is executed, when the condition $NOT(c)$ is fulfilled. Then we have

$$T = \{I_1, c, I_1, c, I_2, c, I_1, NOT(c), I_H\}$$

where I_1, I_2, and I_H are the instructions of the example and the c symbol represents the condition. The implementation assumes that the example is given in advance. The processing (synthesis) is illustrated on the following graph.



Explanation:

a) The starting point of the program.
b) The construction of the program begins with the $1I_1$ node because the first instruction of the example is I_1.
c) The synthesis process continues with the second I_1 instruction and it tries to merge it with the already existing $1I_1$ node. The already available $1I_1$ node is identical with the second instruction of the example and that is why the merge is successful. We have obtained a cycle. This partial model accepts infinite number of instructions of type I_1.
d) The example's third instruction is I_2, which is not represented in the partial model and that is why we have to modify the partial model. We modify it by adding a new node $2I_1$ (we increased the label of the second instruction by backtracking).
e) In the next step we try again to merge the I_2 example instruction with the existed (first) node in the model. This merge is unsuccessful.
f) We try to merge again the same instruction in the example (I_2) with the second node ($2I_1$) of the model but this is also unsuccessful (because I_2 is other type of instruction).
g) Due to the unsuccessful merge we have to add a new node $1I_2$. (This type of instruction was not included in the partial model as yet.)
h) The next instruction of the example (I_1) is merge able with the model $1I_1$ node. The last instruction of the model (I_H) is not yet in the model that is why it is necessary to create a new node $1I_H$.

It is evident, that during the construction of the final model, a new instruction of the example could completely modify the existing model.

4. THE TIME COMPLEXITY OF THE INCREMENTAL ALGORITHM

The time complexity is tightly coupled with the structure of the example. Let us investigate the worst case of the example from the time complexity of view. An example represents the "worst case", when it is necessary to execute maximum number of backtrackings. From the construction of the algorithm it is evident that the more uninterruptible

identical steps contains the example, the more backtracking it is necessary to execute.

The number of backtracking steps, which the above-mentioned incremental algorithm has to execute, is:

$$W = \sum_{i=1}^{n} \frac{i(i+1)}{2} + \frac{n(n+1)}{2}$$

where n denotes the longest sequence (in the example), which consists of identical steps (pair of condition, instruction).

From the time complexity of view, the i^2 is the most relevant element. Investigating the relation between the length of the uninterruptible sequence (trace) and the number of backtrackings, we may express the $\sum_{i=1}^{n} i^2$ element also with the following expression:

$$\sum_{i=1}^{n} i^2 = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$$

it means that time complexity of the worst-case example is polynomial $O(n^3)$.

This algorithm suffers from the flaw that the computing time is cubic for some inputs. This makes it impractical in any system, which is continuously receiving (identical) inputs over a long period of time. Reference [3] suggests a possible solution with the pre-processing of the trace. This could be subject for further investigation.

Programming by examples gains attention from language constructs point of view too. E.g. in [5] the author designs language constructs for programming by examples.

5. THE ROLE OF THE EXAMPLES IN INTELLIGENT TUTORING SYSTEMS

Early researchers on the subject believed that we were on the threshold of computers that could teach. The major restraint was the need for faster, bigger computers. That need is almost met today. Another restraint was the scope of artificial intelligence applications.

Increased access to the Internet and greater bandwidth are both expected to increase the number of individuals moving into online learning. The first LMSs (Learning Management Systems) passively presented hypermedia materials (hypertext multimedia). At the present time the need for the personification forced the implementation of artificial intelligence algorithms. In other words, we can say that the implementation of the already established algorithms (dealt in the field AI or SW engineering) in new environments, created the abstraction of "intelligent tutoring systems". Another stream of research is the standardization in the "provision of access to the highest quality education and training, tailored to individual needs, delivered cost effectively, anywhere and anytime" [7]. These efforts are aimed at the development of the "Shareable Content Object Reference Model" (SCORM), which defines a web-based learning "Content Aggregation model" and "Run-Time Environment" for learning objects. In the next paragraphs we sketch the analogy between the concepts incremental learning and the SCORM concepts.

The SCORM Content Model describes the components used to build a learning experience from reusable learning resources. This model identifies three types of components and defines how these lower-level reusable learning resources may be aggregated into higher-level units of instruction. The three classes of components within the SCORM Content Model are:

Assets: are texts, images, sound, media, web pages and assessments that can be delivered via a web client.

Sharable Content Objects (SCOs): it is a collection of one or more assets that can be launched by and communicate with a LMS.

Content Aggregation: is an organization of assets and SCOs into a cohesive unit of instruction.

We can relate it also to a classroom example; (according to the [7]). The Content Model might be thought of as the classroom itself, containing assets and SCOs (books, maps, displays, teaching kits) that may be aggregated into lessons (content aggregations) by a skilled teacher.

Sequencing and Navigation is information for the LMS telling it what to present to a learner, in what order and what navigation choices to offer. In this way SCORM supports the tracking of the learning activities. They can be, paced, assessed, or any of the other important things, which we can accomplish by using a SCORM conformant learning management system.

To sum up, the implementation of the adaptivity feature into the LMSs is a mainstream in the e-learning research. The question is: "What and where is the role of the examples (or inductive inference algorithms) in the ITSs architecture?" Considering the recent developments in the standardization and the role of the examples in this context, we should derive the following conclusions from the above-mentioned facts:

1. Conditions c of the input example could be replaced by the characteristics of the learning context. I.e. if a certain *aggregation (course)* consists of *shareable content objects* (*modules)*, which consists of *assets* (*chapters)*, then these 3 features (elements) may define the conditions c (or logical expression of conditions) of the input example.
2. Instructions I of the example could be replaced in fact by the *shareable content objects,* which are referred to by hyperlinks in the course.

The inductive inference mechanism is able to archive the "history" of the progress in condensed form. The obtained program represents the path of

the learner's progress in the learning material and may be stored for further evaluation of the learning activities of the student.

The synthesized model could be helpful also for the instructor (or facilitator), because it demonstrates the learning path of the student. It enables the instructor to analyze learner's decisions. Of course the process may act vice-versa too. In that case the learner may study the thinking process of the instructor during the revision and marking processes of the students' work. The system AWS [9] could be helpful too. This system is able to generate information about visitor stereotypes – the user model, by the end of machine learning methods. The system was designed to suggest web pages to the user.

6. CONCLUDING REMARKS

This paper revisits the role of the examples in the design of intelligent tutoring systems. It provides formal description, analysis an example in detail and computes the time complexity of the worst example. Finally, it introduces the correspondence between the concepts of the incremental learning and the concepts used in the SCORM context.

Recent years have witnessed significant progress in intelligent user interfaces. It seems that the role of the examples is important in the human oriented subsystem of the ITSs. The issue remains: "How the learner can gain more insight into the knowledgebase interactively and help tailored to her/his personal needs?"

REFERENCES

[1] Venezky, R.L., Osin, L.: The intelligent design of computer-assisted instruction. (New York: Longman, 1991).

[2] Maulsby, D., Turransky, A.: Watch what I do Watch What I Do: Programming by Demonstration edited by Allen Cypher, co-edited by Daniel C. Halbert, David Kurlander, Henry Lieberman, David Maulsby, Brad A. Myers, and Alan Turransky, 1993, The MIT Press, Cambridge, Massachusetts, London, England

[3] Biermann, A.W., Baum, R.I, Petry, F.E.: Speeding up the synthesis of programs from traces. IEEE Trans. On Computers, Feb. 1975, pp.122-136

[4] Biermann, A.W., Krishnaswamy, R.: Constructing Programs from example computations IEEE Trans. on Software Eng., Vol.SE-2, No.3, Sept.1976, pp.141-153

[5] Rubin, V.R.: Language Constructs for Programming by Example, Proceedings of the third ACM-SIGOIS conference on Office automation systems, Providence, Rhode Island, United States, Pages: 92 – 103, Year of Publication: 1986, ISBN:0-897910210-1

[6] Bjorner, D., editor: Abstract Software Specifications, volume 86 of Lecture Notes in Computer Science. Springer-Verlag, 1980.

[7] ADL SCORM Model. http://www.adlnet.org/

[8] Machová, K.: Machine Learning. Principles and Algorithms. Elfa, s.r.o., Košice, 2002, 117, ISBN 80-89066-51-8

[9] Machová, K., Klimko, I.: Application of Machine Learning for Solving Internet cognitive load. Proc. of the 2nd Slovakia-Hungarian Symposium on Applied Machine Intelligence, Herľany, Slovakia, 2004, 237-242, ISBN 963-7154-23-X

BIOGRAPHY

Ladislav Samuelis, Assistant Prof.: Obtained MSc. in Electrical Engineering at Prague Technical University (1975), and PhD. in Informatics at Budapest University of Technology (1990). Has been engaged in research into the automatic program synthesis at the Institute of Computer Technology at the Technical University of Košice, Slovakia. Since 1998 affiliated with the Dept. of Computers and Informatics, Faculty of Electrical Engineering and Informatics, taught Operating systems, Database systems, Computer Networks and Java. Currently is now involved in research of intelligent tutoring systems.

Ľubomír Fašianok, PhD. student: Obtained MSc. in Informatics and Management at Technical University of Košice (2001). Since 2001 is PhD. student at the Department of Computers and Informatics at the Technical University of Košice. Currently is involved in research of tutoring systems and computer aided eLearning systems design with orientation to the special pattern and template based techniques.