Dustin Jones

Ashok Goel

CS-7637

2016-10-02

Intelligent Tutoring

While some animals have demonstrated the capability to teach others of their kind new abilities and behaviors, humans are unique in having turned learning and teaching into a science. Of the many challenges associated with effective knowledge transfer, finite resources (e.g. time for both student and teacher) play an important role in the ability for humans to teach one another new skills. One way of addressing this limitation is by increasing the availability of teaching resources and bringing on-demand education to students, most notably by building digital tutors (DTs), computer programs designed to teach students a new skill at their own pace.

The number of topics available to learn is infinitely large, and building a DT that can effectively teach all of those topics is currently unfeasible. However, building a DT that can effectively teach a single topic with significant breadth and depth is far more achievable. At the core of all learning, teaching and tutoring is the ability to test if the topic of choice has been learned or not. The most common way for this to be tested is for a question to be posed, an answer given, and the response analyzed for correctness. For a DT, the ability to quantifiably determine if an answer is correct, and if an answer was incorrect how close was it to being correct is crucial. If DT is unable to accurately determine if an answer is correct or not, the tutor is unable to assess if the topic has been learned successfully or if more instruction on the topic is still necessary.

When designing a DT, there are some topics that are better suited to being taught over others. Topics such as psychological therapy are poorly suited for being taught by a DT because the outcomes of such therapy is largely nondeterministic, that is to say there are few conclusive ways of determining if a person being counseled has truly been ‘fixed’; outcomes are generally not binary in nature. The topic of computer programming however, is particularly well suited to be taught by a DT. In its simplest sense, a program takes an input and returns an output; the output is either correct or it is incorrect given the particular input. Programming languages have well defined syntax and semantics, a program in a given language will either compile successfully or it will not. Both of these features make the process of asking a question and analyzing an answer for correctness highly deterministic.

Tutoring in Programming

Teaching a student how to properly construct a program such that it is able to be either compiled or run successfully by an interpreter is a fundamental component of a class seeking to teach computer programming. In order to effectively teach computer programming, the tutor should break the instruction down into many incremental steps, each building upon the last. By breaking down the instruction into many steps, a student’s progress can be evaluated along the way, not only to determine if they’ve learned enough to sufficiently advance, but also to track how well a topic has been learned.

The main process of the DT will be the evaluation of a student at the end of each topic. The DT will present a series of questions regarding the topic, evaluate the answers for correctness, and either assist the student with incorrect answers or advance the student to the next topic. When evaluating responses from the student, the tutor must be able to identify if an answer is incorrect, and in what way it is incorrect. Initially the tutor may be limited to a set of predefined cases determining correctness. As students progress through the course, both on an individual level and across all students, incorrect answers can be recorded. Recording these cases can be a simple step in identifying where a concept might have been missed, forgotten or misinterpreted along a student's learning process.

If we assert that the responses to questions must be runnable content, that is to say that student's answers must be a runnable program, either in whole or in part, then we can establish a basis upon which to evaluate the response. The process for evaluating if an answer is incorrect, and exactly how it is incorrect can be broken into three parts: syntax, content, and output.

Syntax, the verbiage and structure of a response, can largely be enforced and identified by the compiler or interpreter associated with the given language. Identifying syntactic errors is relatively trivial with most modern languages, but identifying why the syntax was incorrect in the first place is much more challenging.

Judging the content of a response, whether or not the response uses the content taught by a particular lesson can be accomplished using lexical analysis, static code analysis tools, and abstract syntax trees. Content of responses generally has a high chance of variation, especially as topics get to more advanced subject matter like design patterns, therefore being able to classify and store multiple distinct solutions to a particular problem (across students) will greatly enhance the ability of the tutor to determine if a response sufficiently exercises the subject of the lesson.

Evaluation of the output is important, but on its own doesn't provide much context to overall issues and learning.

Output needs to be used to ultimately test if a response is correct, but also used to determine where a response went wrong. For example, a common problem among programs is the 'off-by-1' error, where the algorithm by which an output is determined stops just one step short, or goes one step too far of the correct answer, such as in printing all of the elements in an array.

Learning by Recording Cases and Case-Based Reasoning

While each of these steps for judging if an answer is either correct or incorrect can provide some insight to a student on their own, identifying more specifically where the student's response went wrong, what they can do to improve it, and what concepts they may want to revisit are crucial to providing useful feedback from which to learn.

Recording errors of syntax may prove useful in teaching someone already familiar with one programming language a different programming language. This may help personalize lessons to help individuals of different backgrounds learn more effectively and more quickly by providing tie-ins with the language they are more familiar with. They may also help to provide insight as to whether a syntax error was an error in understanding or merely a typographical mistake.

Both recording cases and case-based reasoning are intimately connected when addressing the correctness of content and output. Because of the extreme variation in responses to advanced lesson content questions, recording responses and their output can be used to build a library upon which to compare student responses, both for correctness, but also for things like commonality in

approach, performance, and readability. Taking this approach allows for students to submit answers that while still correct in the conceptual knowledge, adapt it with their own expertise to bring unique and potentially novel solutions to otherwise common questions. The tutor is able to use the collective responses to develop hints on problem areas based on analysis of output and performance metrics when similar pieces of responses are identified in other correct solutions.

Works Cited

Koedinger, Ken, and Michael Tanner. "7 Things You Should Know About Intelligent Tutoring Systems." *ELI* (2013): n. pag. *Educause*. Web. 1 Oct. 2016.