The Diagnosing Behaviour of Intelligent Tutoring Systems

Renate van der Bent

August 2018

Supervisors:
Johan Jeuring
Chris Janssen

Master Thesis Artificial Intelligence

Universiteit Utrecht

Abstract

Intelligent Tutoring Systems (ITSs) need to determine the quality of students' responses to provide relevant feedback. This thesis presents a systematic literature review comparing the diagnostic processes of 40 ITSs of various domains. It investigates what kinds of diagnoses are made and how they are made. It also compares the processes across domains and across four tutoring approaches: model tracing, example tracing, constraint-based and intention-based approaches. The analysis identified eight aspects that ITSs diagnose: Correctness, Difference, Redundancy, Type of Error, Common Errors, Order, Preference and Time. All ITSs diagnose Correctness. Mathematics tutors diagnose Common Errors more often than programming tutors, and programming tutors diagnose Type of Error more often than mathematics tutors. There do not seem to be any differences between approaches. A general model was made that represents all diagnostic processes.

Contents

1 Introduction

Intelligent tutoring systems (ITS) are educational tools that are designed to aid students in their learning by providing personalized feedback. These interactive software systems are not meant to replace classroom teaching, but rather to provide supplemental individual tutoring to a large number of students. Human one-on-one tutoring is thought to be the most effective form of education [1], but it is often expensive. ITSs use techniques from Artificial Intelligence to capture effective human tutoring behaviours in computer programs, making individual tutoring available on a large scale and at a low cost.

Diagnosis is an important factor in tutoring. ITSs typically provide feedback on students' responses. A tutoring system needs to accurately diagnose a response to make sure that feedback is relevant and personalized. To date, there has not yet been any research that compares how different systems handle response diagnosis. In the proposed research I will investigate how different ITSs across various domains diagnose responses.

The aim of this research is to do a systematic literature review of available step-based tutoring systems, with the goal of classifying the diagnostic processes of these systems. Such research is important because a better understanding of the diagnosing behavior of ITSs could lead to better ITS design in the future. It provides insight into what kind of diagnostics are possible. The proposed research does not take into account the effectiveness of tutoring systems. However, the results of this research could be combined with results from effectiveness studies to get a better understanding of what kind of diagnostic processes are likely to be more effective. This knowledge could inform the design of tutoring systems that provide feedback effectively and efficiently.

Before explaining the current study, a common background needs to be established. Therefore, this thesis is structured as follows. In this chapter I will explain how ITSs work and what the different approaches to tutoring are. Then I will discuss the different kinds of feedback and why response diagnosis is an important factor in tutoring. I will end the chapter by introducing the study presented in this thesis. Chapter 2 discusses related work. Chapter 3 describes the method, the results are presented in chapter 4 and chapter 5 and 6 provide a discussion and conclusion, respectively.

A note on terminology: the term *tutor* can refer to either a human tutor or an ITS. Throughout this thesis, I will treat the terms *ITS* and *tutor* as synonyms and I will always refer to a human tutor as *human tutor*, to avoid confusion.

1.1 How Intelligent Tutoring Systems work

A tutor has a task domain, e.g. it might be designed to teach students algebra, programming in a specific language, or physics. In most ITSs, a student works on a sequence of tasks from the task domain. For example, a task in an algebra tutor could be to solve a quadratic equation. The sequence of tasks is managed by an outer loop [2], which decides what tasks to give to a student. How tasks are selected varies per system. ITSs often require students to complete tasks in multiple steps, where "a step is a user interface action that the student takes in order to achieve a task" [2]. Whereas the outer loop is concerned with tasks, the steps within a task are managed by the inner loop. Tasks

can differ in granularity. Granularity refers to the amount of steps required to complete a task, and reflects the amount of reasoning required per step. A larger grain size of the user interface requires more reasoning per user interaction [3]. It should be noted that not all tutoring systems have an inner loop. Those that do not are generally called Computer-Assisted Instruction [4]. The current study only concerns ITSs, i.e. tutoring systems with an inner loop.

The inner loop has several responsibilities. Most commonly it provides feedback on steps, hints on what step to take next, assesses the student's knowledge, and reviews solutions. Some systems only give mimimal feedback. Usually this is information about whether the step is correct or incorrect. Other tutors give error-specific feedback in addition to minimal feedback. Error-specific feedback is meant to help students understand why a step was incorrect and is aimed at preventing similar errors. Hints about next steps are meant to help students who get stuck and do not know how to proceed to solve a task. Assessment of a student's knowledge is an analysis of the student's competence.

There are several approaches to checking and diagnosing student's responses. These are explained in the following sections.

1.1.1 Model tracing

Students' errors in procedural skills are rarely random. In fact, they are often minor variations on a correct procedure that can be precisely defined [5]. These types of errors are known as bugs in the literature. ITSs that have bug catalogs to recognize errors are using the model tracing approach. Model tracing is a paradigm first developed by Anderson et al. [4], for providing individualized tutoring to students. This approach is based on the ACT* theory of cognition and involves constructing performance models [6]. A performance model consists of a set of correct rules rules for solving a specific problem, the "ideal model", and a bug catalog consisting of common misconceptions. The ideal model can thus generate a solution path for the problem. The collection of "buggy" rules allows the model to recognize incorrect solutions. The performance model thus allows the tutoring system to trace the student's cognitive states by comparing the student's responses to production rules in the model.

1.1.2 Constraint-based

In the constraint-based approach to tutoring, knowledge is not represented in the form of production rules, but rather in the form of constraints [7]. Tutoring systems using a constraint-based model check students' responses against a set of constraints for a solution to a problem. As a result, solutions are deemed correct so long as they do not violate any constrains. This is in contrast to the model tracing approach, where responses are usually considered to be incorrect if they are not recognized by the model.

1.1.3 Example tracing

A third approach is example tracing [8]. Example tracing tutors were developed as a trade-of between development cost and instructional power. They can be almost as powerful as model tracing tutors, but require significantly less time to develop. In these tutors, solutions are demonstrated by an expert. These can be correct or incorrect solutions. Solutions can be generalized to problems that are similar. A student's solution is then compared to the expert solutions. Compared to the model tracing and constraint-based approach, this approach is easy to use and inexpensive for tutor authors. A drawback is that it is only "pseudo intelligent", and ITSs using example tracing sometimes cannot recognize all possible solutions to a problem.

1.1.4 Intention-based

Intention-based diagnosis was first implemented in PROUST, an automated bug detector for Pascal programming [9]. This approach aims to understand the intended structure and function of "designed artifacts", particularly program code. It can distinguish errors in intention from errors in the implementation. The intention-based approach is motivated by the empirical finding that beginner programmers often fix surface level implementations of bugs rather than the underlying bugs themselves. To properly diagnose these types of mistakes, a program needs to have knowledge of the intended function of the program, and knowledge of how a student intended to achieve that function. For example, students often confuse the meaning of if- and while-statements. If a student writes a buggy program using a while-statement that results in an infinite loop, when she meant to use the functionality of an if-statement, pointing out the infinite loop is not helpful. Rather, feedback should focus on the difference between if- and while-statements.

1.2 Providing feedback

Narciss [10] identifies five types of feedback. These types are also used in a review of programming tutors [11], [12]. Knowledge about task constraints (KTC) involves hints on task requirements and hints on task-processing rules, i.e. how to approach an exercise. Knowledge about concepts (KC) encompasses explanations about subject matter and examples illustrating concepts. The third type is knowledge about mistakes (KM). This type of feedback can involve the number of mistakes, a description of the mistake or the type of mistake. Keuning et al. [11], [12] distinguish five types of mistakes, although these are specific to the domain of programming (e.g. compiler errors). Knowledge about how to proceed (KH) involves bug-related hints for error correction, hints on task-processing steps and hints on how to improve a solution. Lastly, there is knowledge about meta-cognition, which involves guiding questions or explanations of strategies.

To give relevant KM and KH type feedback on a student's response, an ITS needs to analyze the response and diagnose any errors. For example, a system cannot generate a description of a mistake and give information about how to correct it, if it does not

diagnose the student's error. Therefore, an important responsibility of the inner loop is what VanLehn [2] calls step analysis and what Heeren and Jeuring [13] call the diagnose service. In line with Heeren and Jeuring I will refer to it as the diagnose service.

The diagnose service essentially performs formative assessment. Black and William define formative assessment as "encompassing all those activities undertaken by teachers, and/or by their students, which provides information to be used as feedback to modify the teaching and learning activities in which they are engaged" [14]. In other words, formative assessment is needed to generate feedback for remediation. While this definition does not mention ITSs, it is assumed that ITSs are also capable of providing formative assessment.

Black and William note that teachers rarely use formative assessment in practice. Instead, teachers generally tend to focus on superficial learning and low-level skills such as recall, and mostly provide summative feedback, i.e. grading. Similar results were found by Chi et al. [15]. They examined whether human tutors can accurately assess the level of understanding of students. Their analysis of human tutoring sessions found that human tutors often fail to recognize knowledge deficits, false beliefs and misconceptions that students have. This is unfortunate, because formative assessment has been found to be very effective at improving learning [14]. ITSs that use formative assessment could therefore lead to improved learning.

Formative assessment requires a detailed diagnosis of a student's problems. Nitko and Brookhart [16] distinguish six approaches to diagnosis of learning problems, namely (1) profiling content areas strengths and weaknesses, (2) prerequisite knowledge and skills deficits, (3) mastery of specific objectives, (4) identifying student's errors in performance, (5) knowledge structure analysis and (6) component competencies of problem solving. The fourth approach, identifying student's errors in performance, is the most relevant for ITSs. The goal of this approach is to identify and classify errors in student's performance, so that remedial instruction can be provided. To a lesser extent, the fifth approach is also used in ITSs.

1.3 Research proposal

Because different ITSs handle response diagnoses differently, in this thesis I will compare the diagnosing behaviour of several ITSs, across various domains. Section 3 describes the inclusion criteria for the selection of ITSs to be compared. Specifically I want to study what aspects the diagnose services distinguish and how these are composed. With aspects I mean for example correct rules, buggy rules or constraints. I will compare the diagnostic processes to each other and see if it is possible to make an abstraction or general scheme that they all use. To do this, I will develop a labeling system to categorize the diagnose services and their aspects. I believe that the analysis of diagnose services in ITSs contributes to a better understanding of the diagnosing behavior of ITSs in general. The results of this study could in future research be linked to the effectiveness of tutoring systems, thus leading to a better understanding of what kind of diagnose services are possible and most effective. This knowledge could be used to improve tutoring systems in the future.

My thesis aims to answer the question: How do intelligent tutoring systems determine the quality of student's responses? This will be divided into three main questions. Firstly, what aspects can be distinguished in the diagnosis of student's responses? Secondly, how are these aspects composed in the learning environment? Once these questions have been answered for all selected learning environments, I will compare them to each other and attempt to answer a third question: are there patterns or perhaps even a general scheme that can be identified in the diagnostic process of the different learning environments? I.e. are there diagnostic aspects that are used in all systems, and do they diagnose student's responses in a similar order or following a general scheme? When answering this last question, I will also make a distinction between systems using different approaches to calculating feedback, e.g model-tracing versus constraint-based approaches, and I will make a distinction between domains, e.g. mathematics tutors versus physics tutors.

2 Related work

To date, there is not yet any research on how ITSs across domains diagnose responses. However, there is some related work that is limited to certain domains. Chanier et al. [17] reviewed the ways errors are analyzed in several ITSs for second language learning. They distinguish two kinds of systems: "computational-error systems" and "early deeper Error Analysis systems". These differ in the variety of errors they can handle, and in the level of detail of the error diagnosis.

"Computational-error systems" generally use a computational grammar and a parser to recognize errors in natural language sentences. The ITSs using this approach that were analyzed are the French Grammar Analyser, XTRA-TE, Menzel's system and the ILTS for German. In the French Grammar Analyser, the computational grammar codes all errors explicitly. The authors note that incorrect sentences are occasionally accepted and sometimes wrong corrections are proposed by the system. XTRA-TE is a Chinese-English tutor, that uses a multi-pass parser to diagnose errors. It makes a distinction between syntactic and semantic errors, but cannot recognize word order problems. Menzel's system is an ITS that teaches German. This system uses constraints to diagnose errors. When an error is encountered, the systems selects the minimum set of constraints that have been violated. The ILTS for German has a similar approach as XTRA-TE, but can also handle some word order problems.

The "Early systems with deeper error analysis" that were reviewed are VP2 and ALICE. VP2 is an ITS for native Spanish speakers learning English. It also uses a computational grammar, but unlike the previously mentioned systems, it parses students' responses using both an English and Spanish grammar. This way errors can be explained in terms of the Spanish grammar. ALICE is an ITS for native Italian speakers learning English or French. It combines the use of a bug catalog with pattern matching to identify errors. Furthermore, NOBILE, IFAAR, the ET system and Zock's system were briefly reviewed.

Other related work evaluates diagnosis services in individual systems. For instance, El-Kecha et al. [18] evaluate the diagnosing behavior of PépiMep, a diagnosis system for algebra that is part of a web-based mathematics platform. The system can distinguish 13 different patterns in student's responses.

3 Method

3.1 Selecting Intelligent Tutoring Systems

For a learning environment to be included in this review, it needs to meet three criteria. The documentation describes (1) an intelligent tutoring system, that (2) is capable of providing feedback at the level of individual steps and that (3) has been used in classrooms and not just in experimental settings, or, alternatively, tested on data from real students. The first and second criteria ensure that the learning environment has an inner loop with a diagnosis service. The third criterium ensures ecological validity, i.e. that the learning environment can make realistic diagnoses.

I will go through several selection rounds, to find relevant papers and to make sure that all the ITSs meet the three criteria. There are two ways in which I will find ITSs to include in this study. Firstly, I will consider systems from three reviews. Keuning et al. [12] (manuscript in preparation) classify the types of feedback given in programming tutors. Specifically, from this review I will include the systems that were labeled as providing feedback on task-processing steps, because these are assumed to meet the first two criteria. A review on the effectiveness of tutoring by VanLehn [3] classified systems as "answer-based", "step-based", or "substep-based". From this review I will include systems classified as "step-based" or "substep-based" to meet the inclusion criteria. I will also consider the papers reviewed in Cheung and Slavin [19], which reviews the effectiveness of educational software in mathematics.

Secondly, to find additional papers, I will conduct a literature search. A preliminary search in several academic search engines (Google Scholar, Scopus and ERIC) revealed that Scopus produced the most relevant search results. See Table 1 for the search terms and the resulting number of documents. Relevance of articles was judged by reading the abstract and when necessary by skimming through the article. The search term that produced the most relevant results was "intelligent AND (tutoring OR tutor) AND systems AND ((step AND based) OR stepwise)" in Scopus. This resulted in 195 documents. Using the same terms in ERIC always resulted in fewer documents, i.e. mostly a subset of documents found in Scopus. Searches in Google Scholar resulted in much more, but on average less relevant, documents. The papers found in Scopus were a subset of those found in Google Scholar. For this reason I will use the 195 documents found in Scopus for my analysis.

This initial selection of papers will be checked to see if the systems that are described meet the selection criteria. I will do this by first reading the abstracts, and discarding any articles that clearly do not meet all criteria. If it is unclear based on the abstract whether a system meets all criteria, I will read the full paper, again discarding any that do not meet the criteria. If they do, they will be included in the final selection. The goal is to have around 40 to 60 papers in the final selection.

Search Term	Google Scholar	Scopus	ERIC
intelligent AND tutoring AND systems AND (step-based OR stepwise)	4,270	27	9
tutor AND (step-based OR stepwise)	9,870	45	13
“learning environment” AND stepwise	13,100	52	19
“learning environments” AND (stepwise OR ”step based”)	8,280	55	25
“computer assisted” AND (learning OR instruction) AND (stepwise OR “step based”)	12,800	63	33
intelligent AND (tutoring OR tutor) AND systems AND ((step AND based) OR stepwise)	130,000	195	51

Table 1: The number of documents found in Google Scholar, Scopus and ERIC per search term

3.2 Analysis

I will categorize the ITSs described in the selected papers by approach (model tracing, example tracing, constraint-based, intention-based) and by domain. Then, starting with a small subset of papers (around 10), I will iteratively design a system for labeling and categorizing the diagnose services and which aspects are diagnosed. Using this labeling system I will then categorize the rest of the selected ITSs.

Once all the diagnose services have been labeled, I will check whether there are any differences between approaches or domains, by comparing the frequency at which aspects are diagnosed per approach and per domain. I will also try to capture the diagnostic processes in diagrams and try to abstract a general model or models from the labeling system. The labeling of the ITSs and the general model(s) are the end products of this thesis.

4 Results

The final selection of papers includes 47 papers covering 40 ITSs. A full overview of the ITSs, with corresponding references, domain and tutoring approach can be found in Appendix A. There are 26 model tracing tutors, 8 example tracing tutors, 11 constraint-based tutors and one Intention-based tutor. Note that an ITS can have multiple approaches. Some (e.g. Andes and Mathtutor) use constraints in combination with a model tracing or example tracing approach. It is unclear which approach the Technical Troubleshooting tutor uses.

The results are structured as follows. Section 4.1 describes the diagnostic aspects I have found based on a small sample of papers, and that are used to label the rest of the ITSs. Section 4.2 discusses the results of the diagnostic aspects per ITS and gives examples. Section 4.3 describes the frequency of aspects per approach and domain. The flow of information and the general models representing the diagnostic process are described in section 4.4.

4.1 Diagnostic aspects

Based on a small sample of papers (n=10), I determined that ITSs diagnose the following aspects: Correctness, Difference, Redundancy, Type of Error, Common Errors, Order, Preference and Time. These are explained below. To illustrate these aspects, unless otherwise specified, the running example will be the following algebra problem: "Solve for x: $5x + 6 = 7x$".

Correctness is a binary measure that indicates whether a submitted answer or step matches an ideal answer or step. Possible outcomes are *correct, incorrect* or *unknown*. For instance, if a student writes $5x+7x+6 = 0$ as the next step, this will be diagnosed as incorrect because it does not match the ideal next step $5x-7x+6 = 0$. If a student writes $5x + 6 - 7x = 0$, this will be considered correct because it is semantically equivalent to the ideal answer. In constraint-based approaches, a response is considered correct if it does not violate any constraints and incorrect if it does.

Difference is a measure similar to correctness, in that it indicates whether an answer or step matches an ideal answer or step. However, whereas Correctness is a binary measure, Difference refers to how far-off a submitted answer or step is to the ideal answer or step, on a scale. The outcome can be a number or a percentage. A difference of zero indicates a correct answer. For instance, suppose the difference is measured by edit distance. In the above incorrect response, the difference is one, because it takes one edit (i.e. replace "+" with "-") to change the incorrect response into a correct answer.

Redundancy refers to steps that are unnecessary or superfluous in an ideal solution. This also includes steps that are too small to be recognized as a meaningful step. Possible outcomes are *redundant, not redundant* or *unknown*. For instance, if a

student rewrites $5x - 7x + 6 = 0$ as $-7x + 5x + 6 = 0$, that step can be considered redundant.

Type of Error refers to a classification of errors. Possible outcomes differ per ITS. For example, if a student writes $5x - (7x + 6 = 0$, that can be classified as a syntax error.

Common Errors or Buggy Rules are misconceptions that students may have. Possible outcomes differ per ITS. A buggy rule for the running example could be "If an expression has the form $ax + b = cx$, the next step is $ax + cx + b = 0$". Following this rule leads to the incorrect answer of $5x + 7x + 6 = 0$.

Order refers to the order in which a student takes steps. Possible outcomes are *correct order, incorrect order* or *unknown* Note that this diagnosis has a bigger granularity than other diagnoses, i.e. it is a diagnosis over multiple steps.

Preference If a problem has multiple solutions, some solutions may be preferable over others. Possible outcomes are *preferred, not preferred* or *unknown*. For instance, in a programming tutor, one algorithm may produce the correct result but be less efficient than another algorithm. Preference can also be due to pedagogical reasons, when a teacher wants students to use a certain approach rather than another one, because it does not fit in the curriculum.

Time refers to the time it takes for a student to submit a step or solve a problem. The outcome is a measure of time in milliseconds or seconds. While many systems measure Time, only some use it for diagnostic purposes. This aspect was only labeled when Time was used for diagnostic purposes.

4.2 Diagnostic aspects per Intelligent Tutoring System

A full overview of the diagnosed aspects per ITS can be found in Appendix B. This section will describe the findings of each diagnostic aspect and give examples illustrating how they manifest in ITSs.

4.2.1 Correctness

All tutors diagnose whether a step is correct or incorrect. There are only two ITSs for which this is the only diagnosed aspect, namely Design-A-Plant [20], [21] and Technical Troubleshooting Tutor [22]. All other tutors also diagnose other aspects besides Correctness.

Design-A-Plant is a design-based learning environment that teaches botanical anatomy and physiology to middle school children. In a typical problem, a student's task is to design a plant that can survive in a given environment. A student chooses plant components from a library, and when she has put together a complete plant, the program tests whether the plant would survive in the environment. Whether a plant survives in an environment is checked using constraints. Every environment has constraints that a

plant needs to meet to survive. If (a part of) a plant violates a constraint, it is considered incorrect.

The Technical Troubleshooting Tutor is a learning environment that teaches troubleshooting in aircraft electrical systems. The student is presented with a realistic system fault simulation of an aircraft. The student's task is to inspect the system and perform tests to find out where and why the fault occurred. When a student selects an area where the fault could not have occurred, this is considered an mistake.

The Quantum Accounting ITS [23] tutors students to analyze business transactions. In a typical session, a student sees a list of transactions, and selects one to work on. The student then types in her analysis, i.e. the accounting effects of the transaction. An analysis is considered incorrect if it does not match the model answer. The tutor provides relevant feedback on the correctness of each effect. Students also have the opportunity to ask questions about the transaction. In addition to Correctness, the Quantum Accounting ITS also diagnoses Type of Error. The tutor is illustrated in Figure 1.

Ms. Lindquist [24] teaches symbolization, i.e. converting word problems into algebraic expressions. The ITS uses a natural language dialog to simulate a conversation between a student and a tutor, based on observations of tutoring sessions by experienced human tutors and findings from cognitive research. During a tutoring session, Ms. Lindquist presents an algebra word problem and asks a student to represent it in an algebraic expression. If the student has trouble doing this, the ITS will ask questions to help the student. For example, some questions simplify the problem for the student, break the problem down into steps, ask the student to explain their answer or prompt the student to translate an expression into English. The ITS compares a student's response to the solution produced by the production model to determine the correctness. In addition to Correctness, MS.Lindquist also diagnoses Type of Error and Common Errors.

4.2.2 Difference

There are five ITSs that diagnose the difference between a student's answer and the ideal answer. The Artificial Intelligence Tutoring System (AITS) [25] teaches various topics in AI, and in particular search algorithms. In a typical exercise, students are shown a graph and asked to give the nodes that result from applying a search algorithm to the graph. Thus, answers are represented as a sequence of nodes. The tutor uses edit distance to calculate the similarity between a student's answer and the correct answer, where the edit distance is calculated as the minimal cost of transforming the student's answer into the correct answer using node relabeling, node insertion and node deletion. In addition to Correctness and Difference, AITS also diagnoses Redundancy and Type of Error.

APROPOS2 [26] is an automated debugger for Prolog programs. A program in Prolog consists of a series of predicates and rules. A student is given a task and is asked to write a program to solve the task. APROPOS2 then checks the program to find bugs and suggests ways to fix them. It captures Difference in a penalty score. Programs

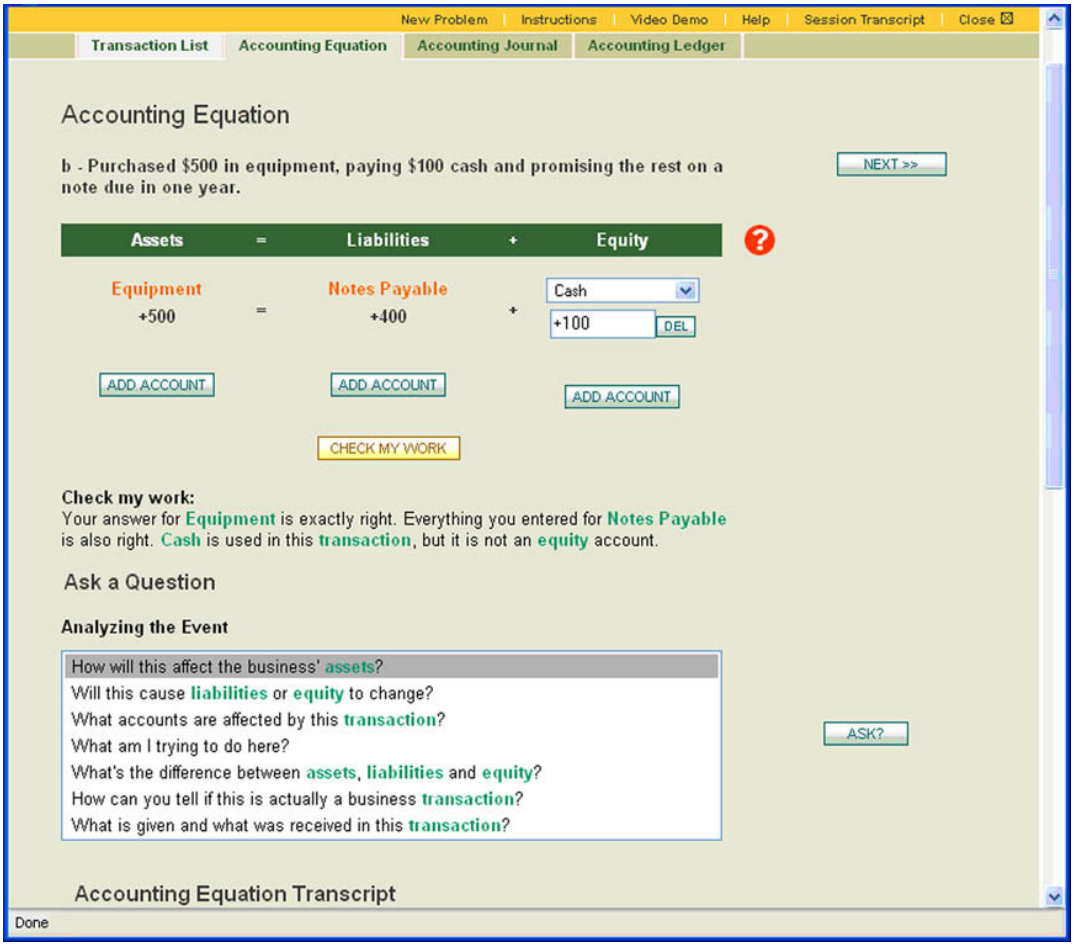

Figure 1: Screenshot of error entry and tutor's confirming and corrective feedback in Quantum Accounting [23].

are represented by templates called Prolog-frames (P-frames). The P-frame of a student's algorithm is matched against possible solution P-frames. The lower the penalty score is, the better the match. An algorithm that is computationally equivalent to a solution algorithm gets a penalty score of 0. In addition to Correctness and Difference, APROPOS2 also diagnoses Redundancy, Type of Error, Common Errors, Order and Preference.

(Why2-)Autotutor [27] is a dialogue-based tutor for qualitative physics and computer literacy. During a typical session, the tutor asks a question that requires approximately a paragraph to answer sufficiently. To determine whether an answer is complete, Autotutor uses a Latent Semantic Analysis (LSA) score to represent the difference between the student's answer and the ideal answer. Every question has expectations associated with it. An expectation represents an aspect of a correct answer. An ideal correct answer is the set of all expectations of a question. For example, if a correct answer has five

expectations embedded in it, AutoTutor expects a complete answer to mention all five expectations. A student's responses to a question are matched to the expectations using an LSA score. If all responses match an expectation above a certain threshold, the expectation is considered covered. When a student gives an incomplete answer, the tutor asks follow-up questions until all expectations have been covered. In addition to Correctness and Difference, (WHhy2-)Autotutor also diagnoses Common Errors.

In AzAR 3.0 [28], a foreign language pronunciation ITS, Difference denotes how close the pronunciation of a phone[1] is to the reference pronunciation. The voice interactive tutor teaches pronunciation in second language acquisition and supports several languages, namely German, Polish, Russian, Czech and Slovak. It uses a Hidden Markov Model-based classifier and speech signal analysis to recognize a student's input, and compares it to a reference model of a native speaker. The difference is communicated to students using a color scale from red to green, where red indicates a mispronounced phone and green indicates perfect pronunciation. In addition to Correctness and Difference, AzAR 3.0 also diagnoses Type of Error and Common Errors.

The Research Methods Tutor (RMT) [29] teaches research methods in psychology using natural language dialogue. An animated pedagogical agent asks a student questions and analyses the student's response using Latent Semantic Analysis. It compares responses to expected responses and creates a vector representation of both. The similarity between the student's response and the expected response is represented by the vectors' cosine. Since this is a measure of similarity, where 1 means the response is equivalent to the expected response, the Difference is actually 1 minus the vectors' cosine.

4.2.3 Redundancy

Five ITSs can diagnose redundancy. These are Dragoon [30], APROPOS2, AITS, Keuning14 [31] and PHP ITS [32]. Some ITSs treat redundancy as an error, while others don't. Dragoon is an ITS that tutors students in modeling dynamic systems. In a typical problem, a student has to construct a directed graph representing a model of a dynamic system. In some of the problem descriptions, not all facts and numbers are relevant to the problem. When a student tries to define an unnecessary parameter, it is redundant, and Dragoon gives appropriate feedback. Dragoon treats redundancy as an error. In addition to Correctness and Redundancy, Dragoon also diagnoses Type of Error.

As mentioned earlier, in APROPOS2, programs are represented by P-frames. Students' programs are mapped to reference P-frames to find the best match. NOMATCH is a special P-frame which is used when no other match could be found. When a predicate definition or a subgoal is mapped to NOMATCH, it is considered redundant.

In AITS, an answer is redundant when the number of nodes in a node sequence is greater than the number of nodes in the ideal answer.

Keuning14 is an ITSs based on the IDEAS framework [13]. It teaches imperative programming. Keuning14 can diagnose another form of redundancy: similarity, although

[1] *Phone* is a linguistic term meaning a speech sound that is not specific to any language.

this function is currently not used. A program is diagnosed as similar when no significant changes have been made since the last submission. Since a step with no significant changes is unnecessary and superfluous, similarity is a form of redundancy. In addition to Correctness and Redundancy, Keuning14 also diagnoses Type of Error and Preference.

The PHP Intelligent Tutoring System teaches programming in PHP to beginner programmers. The ITS uses first order logic and AI concepts to represent states of programs and state changes. An exercise is specified by a set of facts about the Initial State and the Overall Goal. Redundant code in a student's program is found by keeping track of dependencies in a status diagram. If there are facts that do not contribute to any facts in the Overall Goal, those facts are considered unnecessary. Redundant code is considered incorrect. In addition to Correctness and Redundancy, PHP ITS also diagnoses Type of Error.

4.2.4 Type of Error

Type of Error is the second most commonly diagnosed aspects, with 28 ITSs making this diagnosis. Only a few of these are illustrated here. The error categories that are distinguished vary greatly per system and also depend on the domain. HBPS [33] teaches word/story problems in algebra. In a typical session, a student has to express relationships between quantities in algebraic expressions. For example, a problem description might be "Mike's father is three times as old as Mike. 4 years ago, he was four times older. How old is Mike?" The pogram distinguishes type I and type II errors. A type I error is when a student successfully identifies a relationship between quantities but expresses it incorrectly. Type II errors are when a student fails to identify a correct relationship between quantities. In addition to Correctness and Type of Error, HBPS also diagnoses Common Errors.

As mentioned earlier, an answer in AITS consists of a sequence of nodes. The number of nodes is checked against the correct answer to see if an answer is complete. The contents of the nodes is checked to see if they are accurate. Therefore, this ITS distinguishes three types of errors: complete but inaccurate, incomplete but accurate, and incomplete and inaccurate.

In the Invention Lab [34], a student learns scientific inquiry by inventing methods for calculating target properties of data. Students go through cycles of ranking the data, designing a method and evaluating it until the constructed method is valid. If the ranking is correct, the tutor checks whether any constraints are violated. If there are, in the next cycle, the dataset will be generated in such a way that the conceptual errors are exposed, i.e. the proposed method will not work on the new datasets. Thus, this system identifies the type of error and then generates datasets that specifically target the error. In addition to Correctness and Type of Error, the Invention Lab also diagnoses Common Errors.

Newton's Pen [35] uses a "pentop computer", i.e. a pen with an integrated digitizer and processor. The ITS teaches statics, which is a part of mechanical engineering concerning forces on bodies. Newton's Pen uses various image recognition techniques to recognize diagrams drawn on paper. The tutor can identify which properties of a

diagram are (in)correct. For example, it might give an error message about the location or orientation of an arrow.

With the exception of The LISP Tutor [36], [37] and the ACT Programming Tutor [38], all programming tutors also distinguish types of error. The most basic distinction in a programming tutor is between syntax errors and semantic errors, but several tutors also make more detailed distinctions.

4.2.5 Common Errors

There are 19 ITSs that can diagnose common errors. Only a few of them are illustrated here. All of the Cognitive Tutors (ACT Programming Tutor, Geometry Explanation Tutor [39], [40], Geometry Tutor [41], Mathtutor [42], Ms. Lindquist [24] and The LISP Tutor [36]) use buggy rules. They have an ideal model, i.e. a simulation of how an ideal student would solve the problem at hand. They have production rules, that the ideal model uses to construct programs. They also have a set of buggy rules. Buggy rules are production rules that represent common incorrect steps. At each step a student takes, a cognitive tutor tries to match the step to a production rule or buggy rule. If a step matches a buggy rule, it is considered incorrect and relevant feedback is given.

AskElle [43] teaches beginner programmers Haskell, a functional programming language. In this ITS, solutions are represented by strategies, which describe the process of solving a problem. It describes what steps need to be taken and how the steps relate to each other. A diagnose service tries to derive a submitted (possibly incomplete) program from the previously submitted program using any rules that are allowed by the strategy. If the student's program does not appear in the set of expected programs, the tutor checks if it can derive the program using known wrong approaches. In addition to Correctness and Common Errors, Ask-Elle also diagnoses Type of Error and Preference.

PLATO [44] is a framework for educational games based on Tutoring by Issue and Example. One of those games is an arithmetic game called "How the west was won". In the game, a student and the computer take turns creating arithmetic expressions from three randomly selected numbers and solving them. The goal of the game is to get students to create many different kinds of expressions. The value of the expression is also the number of steps the player can take on a board. Figure 2 illustrates the game board. The board has shortcuts and players can "bump" each other back. Because of this, it is beneficial to use different strategies depending on the situation. Since students often use a single strategy, the tutor's feedback focuses on getting the student to try different strategies. PLATO uses a paradigm of "Issues and Examples" to tutor students. An issue is activated when the student's behaviour indicates that they don't know a certain concept or skill. Issues can be considered common errors because they represent strategies that students often have trouble with. In addition to Correctness and Common Errors, PLATO also diagnoses Type of Error and Preference.

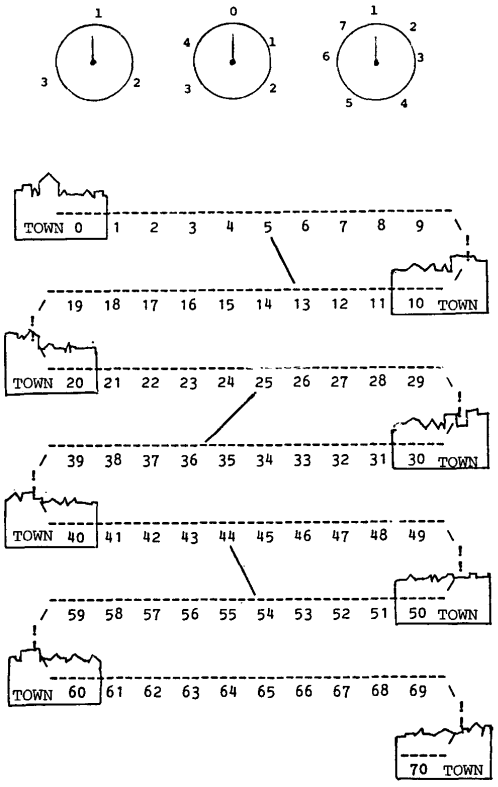

Figure 2: Screenshot of the game board and the randomly selected numbers in PLATO. [44]

4.2.6 Order

Four ITSs diagnose the order of steps. These are Mathesis [45], [46], APROPOS2 [26], Mathtutor and CIRCSIM-TUTOR [47], [48]. The Mathesis tutor teaches algebra to high school students. Mathesis has a feature called intelligent task recognition. Any algebraic expression can be entered into the tutor and is recognized by it. The tutor then generates model solutions that a student's solution can be compared to. Order is diagnosed by checking for operator precedence. To solve an algebraic expression, a student selects a part of it that she wants to work on. The student then selects an operation that she wants to perform on the selected part, from a drop-down menu. The tutor checks its internal representation of the expression to see if the selection has the highest operator precedence and whether the selected operation is the correct step to take, and provides feedback about it to the student. In addition to Correctness and Order, Mathesis also diagnoses Type of Error and Common Errors.

APROPOS2 checks the order of clauses and subgoals of a program, because in Prolog these can affect the outcome and efficiency of a program. It attempts to find wrong orders by heuristic code-matching.

Mathtutor is an example tracing tutor based on several Cognitive Tutors in the mathematics domain. Solutions and common errors are captured in behaviour graphs. In a behaviour graph, an author can specify whether steps can be taken in any order, or whether a strict order must be followed. Steps can be grouped together and nested, and the order can then be specified on a per-group basis. In addition to Correctness and Order, Mathtutor also diagnoses Common Errors.

CIRCSIM-Tutor is a dialogue-based ITS, which tutors students on circulatory physiology, specifically the way blood pressure is stabilized in the body. In a typical session, before the dialogue begins, a student fills in a table specifying which changes take place in seven components when something in the circulatory system is disturbed. Based on errors in this table, the ITS plans a tutoring dialogue. The tutor also takes into account the order in which values are added to the table, where some orders are considered incorrect. The reason for this is that the order of entry is thought to reflect the student's reasoning process. In addition to Correctness and Order, CIRCSIM-Tutor also diagnoses Type of Error.

4.2.7 Preference

There are five ITSs that use a preference diagnosis. These are APROPOS2, Andes [49], Ask-Elle, Keuning14 and PLATO. In both APROPOS2 and Ask-Elle, efficient algorithms are preferred over inefficient ones. In both of these tutors, when a student submits an inefficient program, the tutor will respond that, while correct, the algorithm is suboptimal.

Ask-Elle and Keuning14 also diagnose another type of Preference, in the form of detours. A program has a detour when all its steps are valid, but it does not follow a strategy. The program is correct, but not preferred, because in these tutors it is preferable for a program to follow a strategy.

Andes is a tutor for college level physics. Andes diagnoses preference for pedagogical reasons. In a typical exercise, a student draws a coordinate system and vectors, defines variables and then writes equations. After every input action, the element in question is coloured green or red, indicating correct or incorrect, respectively. To promote conceptual understanding of physics, the system encourages students to write down major principles instead of immediately applying them to a problem. When students do not do this, the equation is coloured green because it is correct, but they lose points and are given a warning message because it is not the preferred approach. In addition to Correctness and Preference, Andes also diagnoses Type of Error.

In PLATO, students are free to choose a playing strategy, but some strategies are preferred by the tutor over others. When a student makes a suboptimal move, the tutor does not immediately give feedback. It only gives feedback when the student repeatedly makes suboptimal moves, which indicates that she does not know that other strategies can be used.

4.2.8 Time

Only one ITS uses time to diagnose students, namely Zatarain-Cabada13 [50]. This ITS is an affective tutor for arithmetic embedded in a social network. It uses response time to estimate the difficulty of a question. Students perform an initial diagnostic test to assess their knowledge level. Questions are ranked according to difficulty, and their weights correspond with their difficulties. The result of the test determines the learning level and teaching method of the exercises. The tutor tracks how long it takes to answer questions. A longer response time is interpreted as the student having difficulty answering the question.

4.3 Diagnostic aspects per Approach and Domain

In this section, the frequency at which aspects are diagnosed is compared between approaches and domains. This gives insight which aspects are most common in each approach and domain. Section 4.3.1 looks at the frequency per approach, and section 4.3.2 looks at the frequency per domain.

4.3.1 Per Approach

There are four approaches: model tracing tutors, example tracing tutors, constraint-based tutors and intention-based tutors. Note that the categories partially overlap. There are five ITSs that are in both the model tracing and constraint-based category, and one ITS (Mathtutor) is in both the example tracing and constraint-based category. Also note that, of the 40 ITSs included in this study, only one uses an intention-based approach. Table 2, 3, 4 and 5 show the frequency of aspects in model tracing tutors, example tracing tutors, constraint-based tutors and the intention-based tutor, respectively.

Model Tracing tutors	**26**
Correctness	26
Type of Error	16
Common Errors	14
Preference	3
Difference	2
Order	2
Redundancy	1
Time	1

Table 2: Frequency of aspects in model tracing tutors

There do not seem to be any differences between approaches.

Example Tracing tutors	**8**
Correctness	8
Type of Error	7
Common Errors	3
Difference	3
Redundancy	3
Order	2
Preference	1

Table 3: Frequency of aspects in Example Tracing tutors

Constraint-based tutors	**11**
Correctness	11
Type of Error	8
Common Errors	5
Preference	3
Difference	1
Redundancy	1
Order	1

Table 4: Frequency of aspects in Constraint-based tutors

4.3.2 Per Domain

While the domains of the ITSs vary greatly, they can be roughly grouped into four domains: Mathematics, Programming, Physics and Other domains. Mathematics includes domains such as algebra, arithmetic and geometry. Programming includes programming in specific languages, and more general domains such as object-oriented design and data structures. Physics includes qualitative physics and statics. Other domains include everything else, such as botany, foreign language pronunciation, database design and aircraft engineering. See Appendix A for all domains. Note that the categories overlap partially. (Why2-)Autotutor is in both the Physics and the Other Domains category because it teaches both physics and computer literacy. iList is in both the Programming and Other Domains category. This is because it teaches students about lists, which are important data structures in programming, but the tutor does not teach programming, per se. Rather, it teaches how lists behave, which is a more general topic of computer science. Table 6, 7, 8 and 9 show the frequency of aspects per domain, for Mathematics, Programming, Physics and Other domains, respectively.

An interesting finding is that ITSs in the domain of mathematics more often diagnose Common Errors than ITSs in the other domains. 90.9% of math tutors diagnose Common Errors, compared to only 33.3% of programming tutor, 50% of physics tutors and 33.3% of tutors in other domains. In the domain of programming, ITS diagnose Type of Error

Intention-based tutors	**1**
Correctness	1
Type of Error	1

Table 5: Frequency of aspects in Intention-based tutors

Mathematics	**11**
Correctness	11
Common Errors	10
Type of Error	5
Order	2
Preference	1
Time	1

Table 6: Frequency of aspects in Mathematics tutors

more often than in the other domains. 86.7% of programming tutors diagnose Type of Error, compared to 45.5% of mathematics tutors, 75% of physics tutors and 66.7% of tutors in other domains. Also, Redundancy is diagnosed in three Programming tutors and two Other Domain tutors, but not in any Mathematics and Physics tutors. That is 20% of Programming tutors and 17% of Other Domains, compared to none of the Mathematics and Physics tutors. Again, no statistical analysis was used to determine the significance of these results, due to the small sample size and the exploratory nature of this research. The rest of the aspects seem to be diagnosed at a similar frequency across domains.

Programming	**15**
Correctness	15
Type of Error	13
Common Errors	5
Redundancy	3
Preference	3
Difference	1
Order	1

Table 7: Frequency of aspects in Programming tutors

Physics	**4**
Correctness	4
Type of Error	3
Common Errors	2
Difference	1
Preference	1

Table 8: Frequency of aspects in Physics tutors

4.4 Diagnostic Processes

In this section, the flow of information in ITSs leading to a diagnosis is illustrated in diagrams. Not all ITSs are represented here, because the diagnostic process was not always described in enough detail to make a diagram. White nodes represent input, green nodes represent diagnostic ITS components and purple nodes represent a diagnosis.

Figure 3 shows the most basic diagnostic process in an ITS. This diagram applies to Assistment, Design-a-Plant and Quantum Accounting. In these tutors, a student's submitted response is checked against a single ideal solution. If it matches, the response is correct. If not, the response is incorrect. Although Assistment and Quantum Accounting have an additional diagnostic aspect, namely Type of Error, this is not shown in the diagram. This is because it is unclear where the Type of Error is determined. RMT's diagnostic process is very similar, except that it uses cosine similarity to check whether an answer matches the expected response.

Other domains	**12**
Correctness	12
Type of Error	8
Common Errors	4
Difference	4
Redundancy	2
Order	1

Table 9: Frequency of aspects in Other Domain tutors

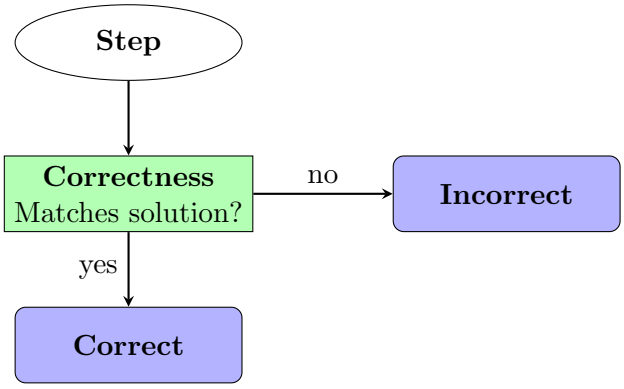

Figure 3: Assistment, Design-a-Plant, Quantum Accounting

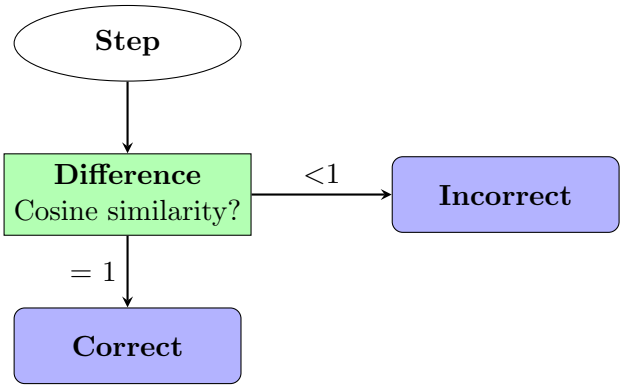

Figure 4: RMT

Figure 5 shows how Redundancy and Correctness are diagnosed in Dragoon. A node is redundant when it matches a distractor node, correct if it is equivalent to the ideal answer, and incorrect otherwise. Dragoon also diagnoses Type of Error, but it is unclear from the paper how this is determined.

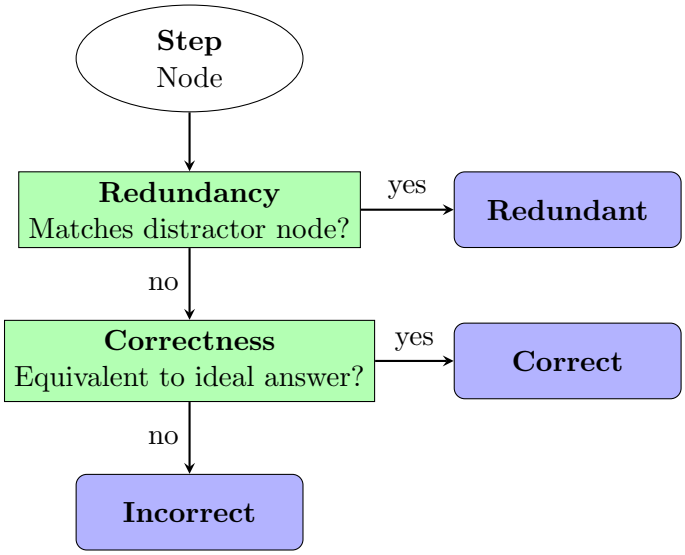

Figure 5: Dragoon

The information flow in AITS is illustrated in figure 6. The ITS calculates Difference in the form of edit distance. This information is used to infer Correctness. If the edit distance is zero, the node sequence is correct. Otherwise, AITS checks the number of nodes and the content of the nodes in the submitted answer, and uses this to determine Redundancy and the Type of Error. Redundancy is treated as a Type of Error. The *complete* and *accurate* diagnoses are labeled as types of errors. This might seem counter-intuitive, but this is because Type of Error in this ITS is a combination of completeness and accuracy, so a step can be *complete but inaccurate*, *incomplete but accurate* or *incomplete and inaccurate*. The diagnosis *complete and accurate* will never occur because in that case the edit distance is zero, and thus the step will be diagnosed as correct.

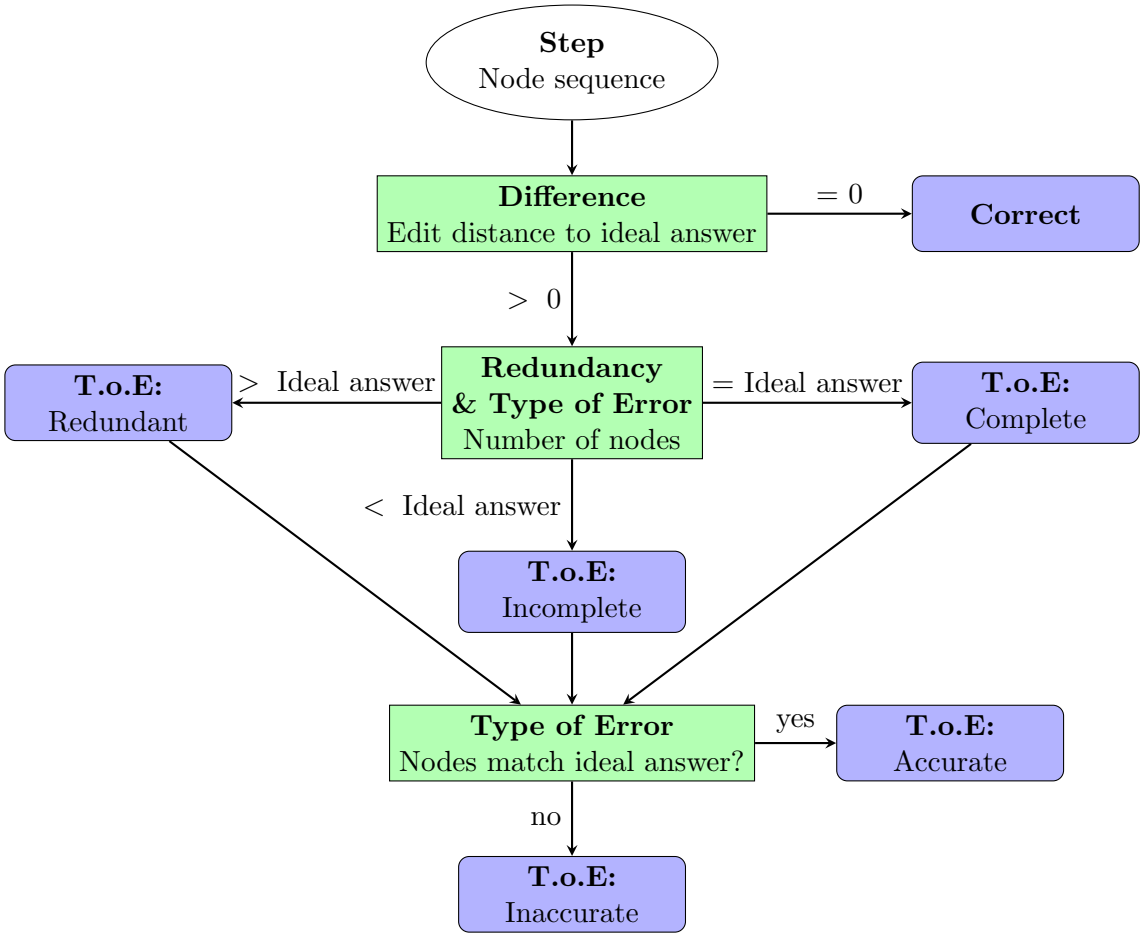

Figure 6: AITS

Figure 7 shows the flow of information in Mathesis. A student selects a part of an algebraic expression. Mathesis then checks whether the selected part has the highest operator precedence. If not, the order of steps is incorrect. If the selection does have operator precedence, the student can then select an operation to apply to the selection. Mathesis checks whether the operation matches the ideal answer. If it does not match, the operation is incorrect. If it does, the student types in the result of applying the operation to the selected part of the algebraic expression. Mathesis checks whether the result matches the ideal answer. If it does not, it checks whether it matches the result of any known buggy rule.

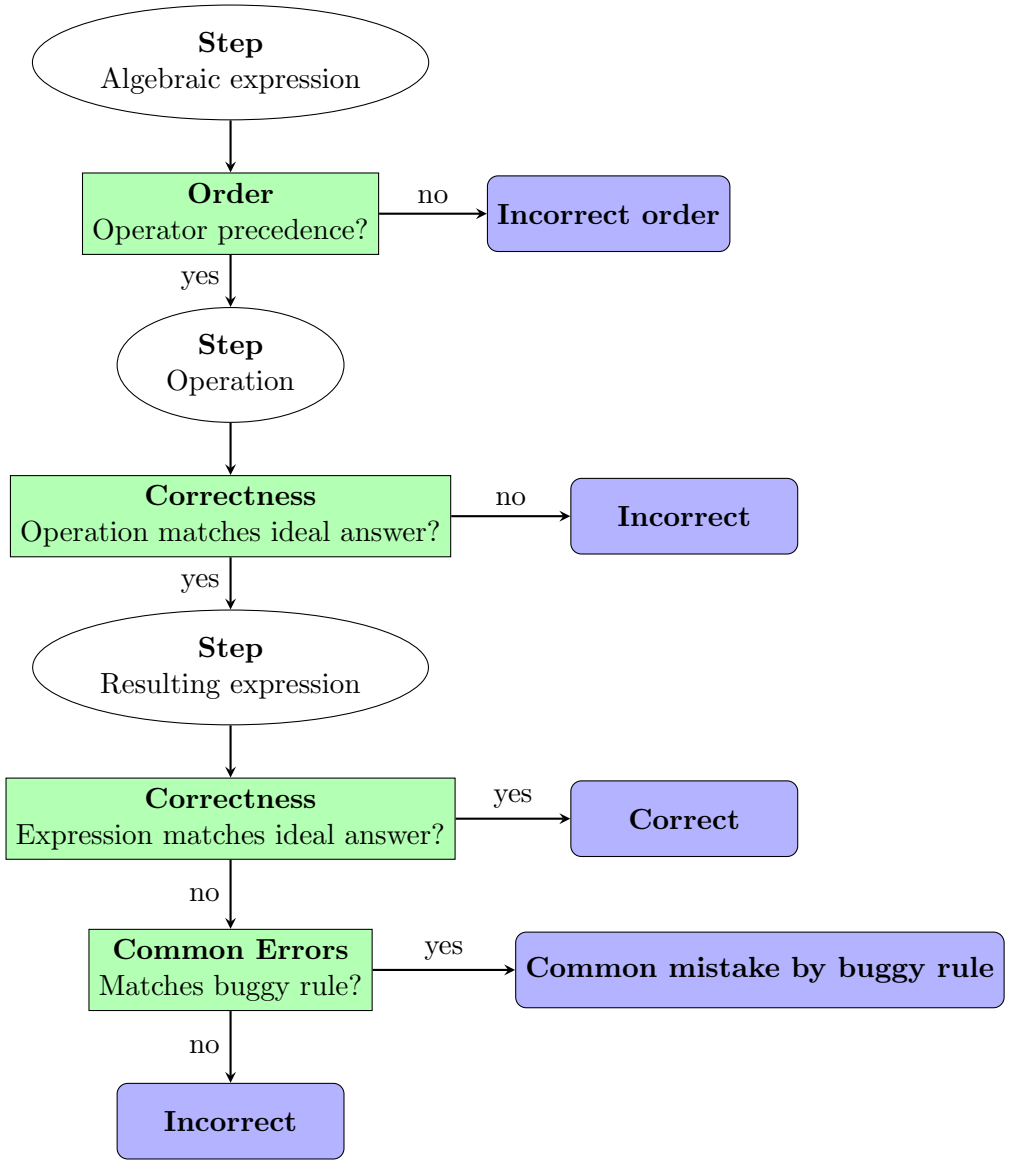

Figure 7: Mathesis

Figure 8 illustrates the information flow in HBPS. First, the ITS checks whether the input is a letter or an expression. A letter is always valid. If it is an expression, the ITS checks whether the expression matches an edge in the hypergraph that represents the problem. This is done by checking if all variables that occur in the expression also occur in an edge. If a match is found, this means the student correctly identified a relationship between the variables. However, it does not necessarily mean that the relationship was correctly expressed. Therefore, HBPS then checks whether the operations and the order of variables match those in the edge. If not, this is classified as a Type I Error. If no matching edge is found, this is classified as a Type II Error, which means a student

was unable to correctly identify a relationship between variables. The ITS then further checks whether the error can be explained by a buggy rule, or if it is an unknown mistake.

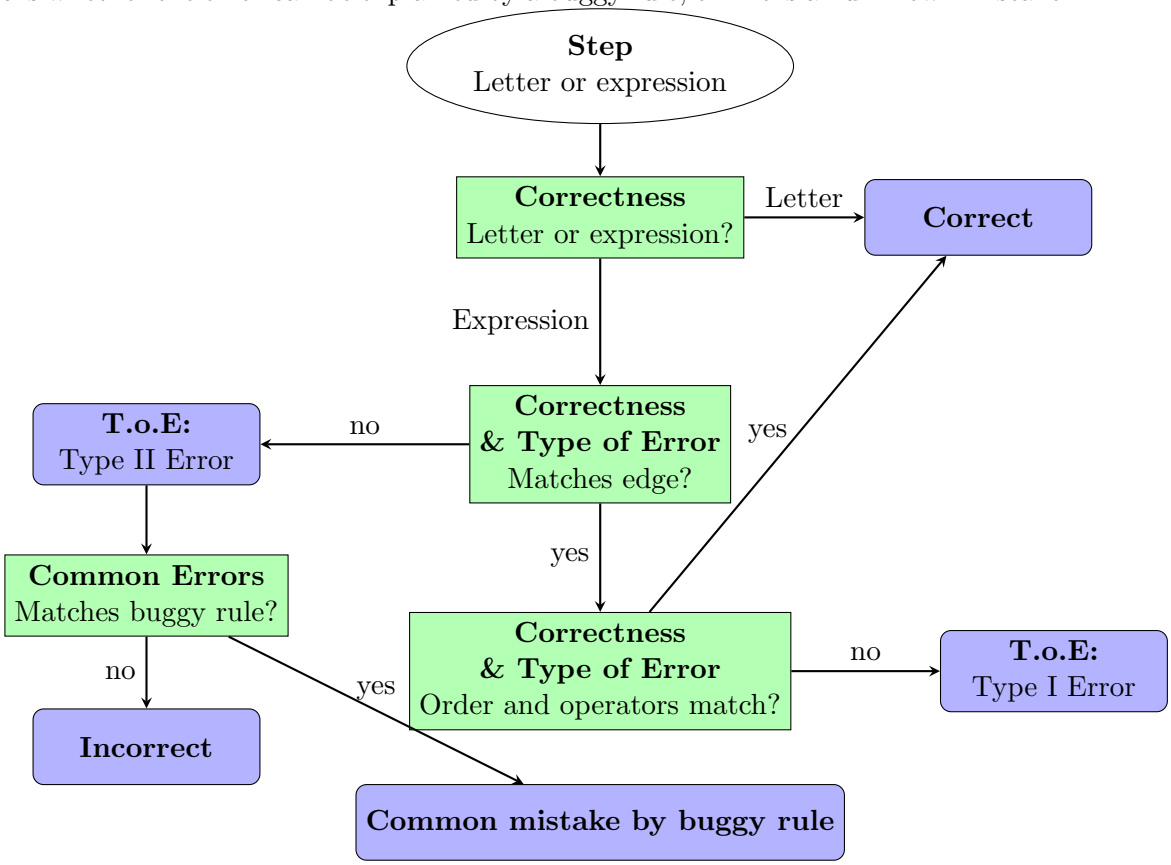

Figure 8: HBPS

Figure 9 shows the basic information flow in a cognitive tutor. This diagram applies to the ACT Programming Tutor, the LISP Tutor and the Geometry Tutor and also to PAT2Math. First, the ITS checks whether a submitted answer matches any production rule. If it does, the answer is correct. If it does not, the ITS checks whether it matches a known buggy rule. If it does, it is a common mistake by buggy rule. If not, it is an unknown mistake. The other cognitive tutors (Geometry Explanation Tutor, Mathtutor and Ms. Lindquist) have slightly different components. The diagnostic process in Mathtutor is very similar, as is shown in figure 10. Because Mathtutor uses an example tracing approach, unlike the other cognitive tutors, this tutor uses interpretations instead of production rules. An interpretation is a path in a behaviour graph that represents a (possibly buggy) solution. An interpretation includes a specification of the order of steps. Therefore, when an answer matches an interpretation, this automatically means that the Order is correct. The information flow of The Geometry Explanation Tutor and Ms. Lindquist could not be captured in a diagram because they are dialogue-based tutors. The diagnostic process of these tutors is more complicated, and is not fully

explained in detail in the respective papers.

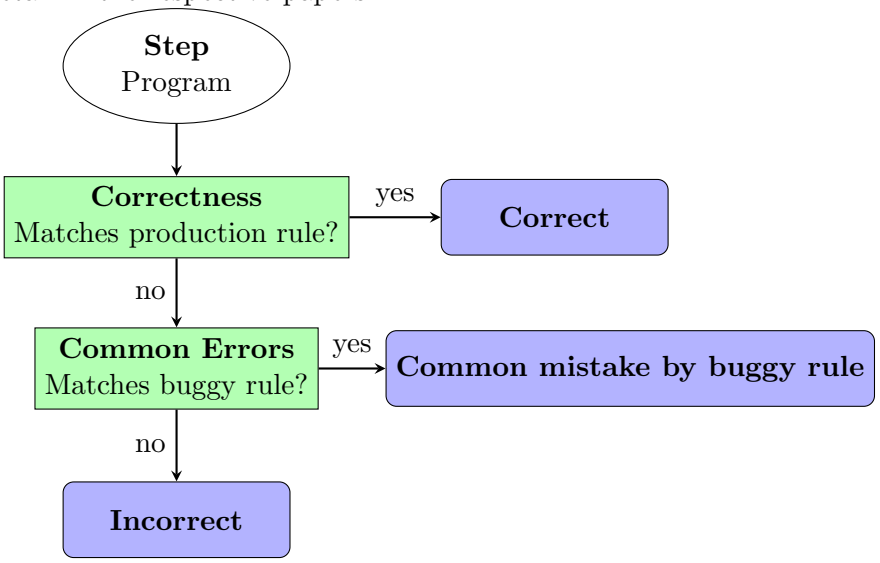

Figure 9: ACT Programming Tutor, LISP Tutor, Geometry Tutor, PAT2Math

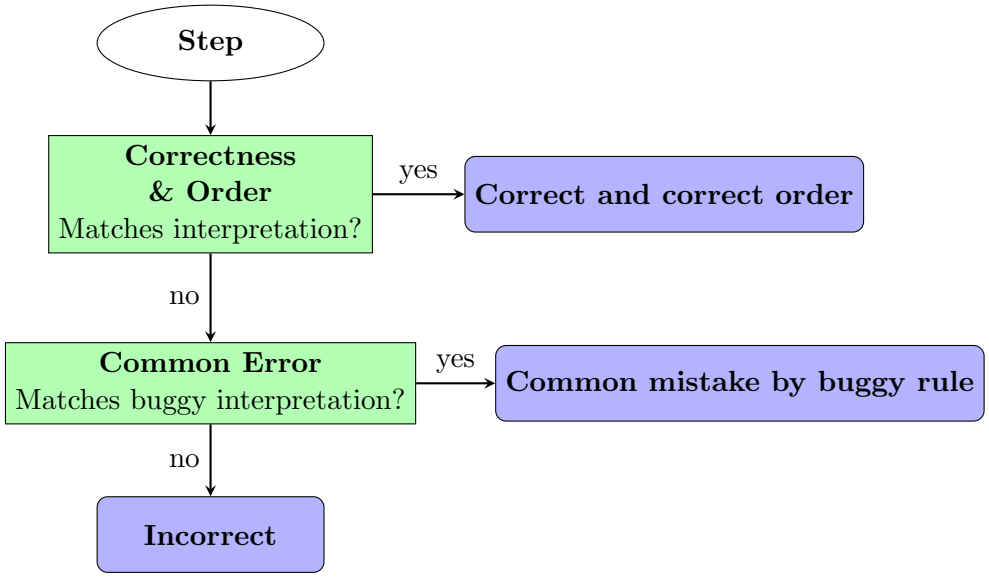

Figure 10: Mathtutor

The diagnostic process of the Invention Lab is shown in Figure 11. Students make a prediction about the ranking of two datasets. For instance, suppose the task is to invent a method to calculate the spread of a dataset. A student is presented with two datasets which she can use to test her method. Firstly, the student makes a prediction about the ranking of the datasets, i.e. which dataset has a larger spread. Then, the

student demonstrates a method to calculate the spread, by applying it to examples from the datasets. The ITS checks whether the same method is applied to both datasets. If not, this is classified as a method error. If it is, the ITS checks whether the method results in the predicted ranking. If the ranking is different, the method is incorrect. If the ranking is the same, the ITS checks whether the method violates any constraints. If no constraints are violated, the invented method is considered correct.

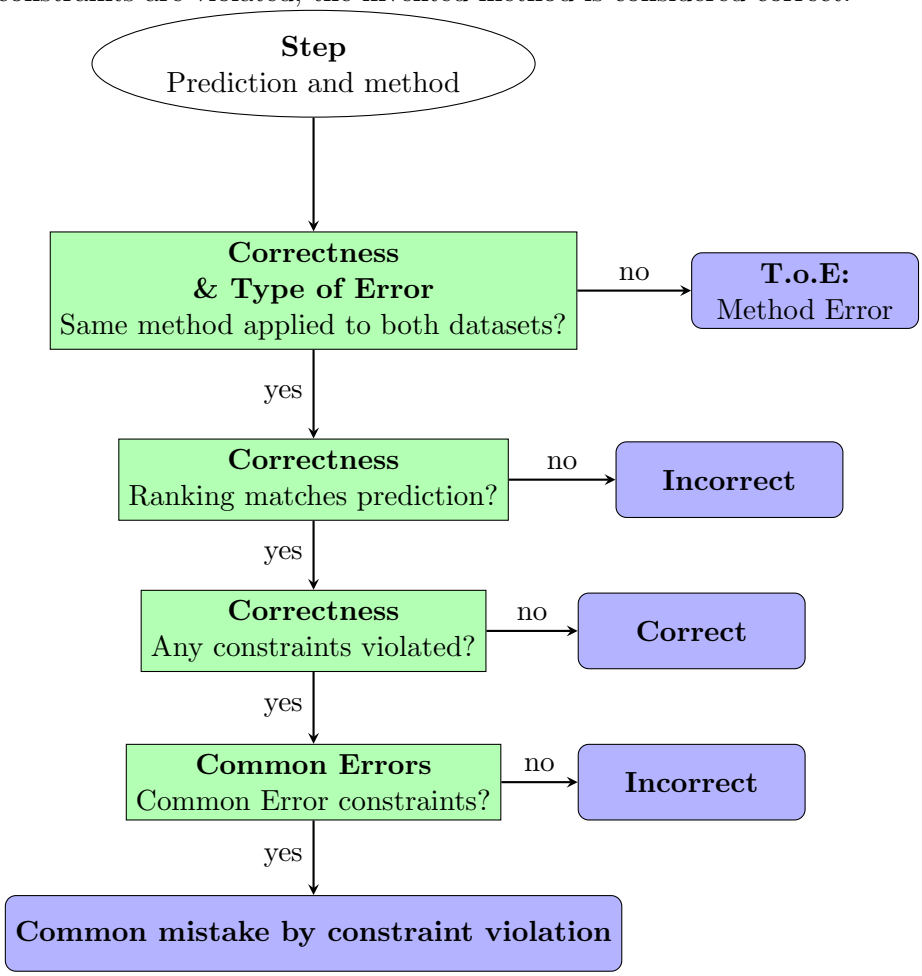

Figure 11: The Invention Lab

The diagnostic process in Keuning14 is illustrated in figure 12. A submitted piece of code is compared to known strategies. If the code follows a strategy, it is correct and expected. If it does not follow a known strategy, the tutor checks whether any significant changes were made to the code since the last submitted step. If not, the code is considered similar to the last step and thus redundant. If not, the ITS checks whether the code is recognized as a valid step since the last submitted step. If it is, the code is correct but a detour, and thus it is not preferred. If no valid step is recognized, the tutor tests the program to check whether the output is as expected. If the output is different,

the program is incorrect. If the output is correct, the program is correct but unknown.

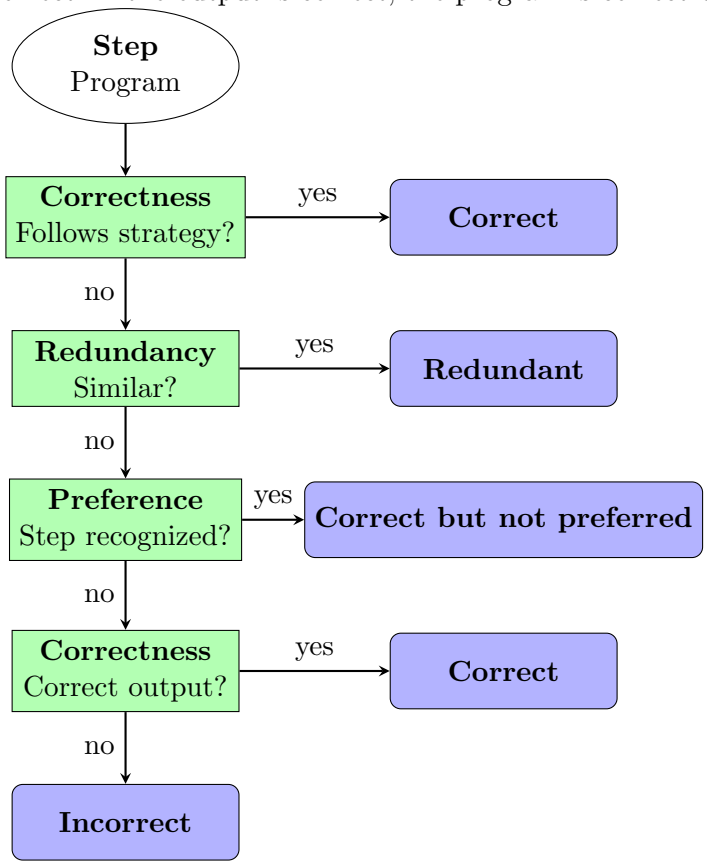

Figure 12: Keuning14

The diagnostic process in Ask-Elle is similar to that of Keuning14, with the addition of buggy rules and preferred strategies. The process is illustrated in figure 13. A submitted piece of code is compared to known strategies. If the code follows a strategy, the tutor checks whether the used strategy is optimal or if there are other, more preferable, strategies. If there aren't, the program is correct and expected. If the strategy is not the preferred strategy, the program is correct but suboptimal. If it does not follow a known strategy, the ITS checks whether it recognizes any known wrong approaches that represent common mistakes. If no common mistakes are found, the ITS checks whether the code is recognized as a valid step since the last submitted step. If it is, the code is correct but a detour, and thus it is redundant. If no valid step is recognized, the program is incorrect and unknown.

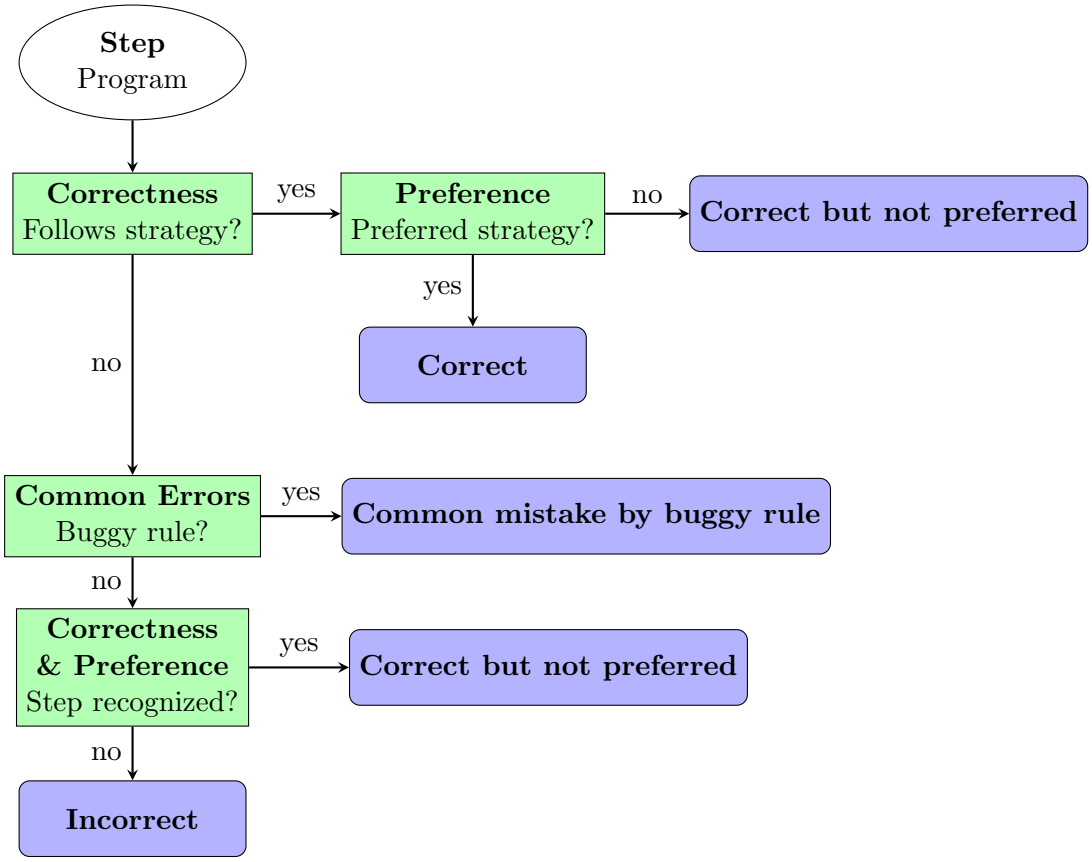

Figure 13: Ask-Elle

Figure 14 illustrates the information flow in PHP ITS. As was explained before, in this ITS program states and state changes are represented using first order logic and an exercise is specified by a set of facts about the Initial State and the Overall Goal. Firstly, the tutor checks whether all necessary facts are present in the code. If there are facts missing, the code is incorrect. Which facts are missing determines the Type of Error. If all necessary facts are present, the tutor checks whether all of them contribute to the Overall Goal. If there are facts that do not contribute to the Overall Goal, those facts are considered Redundant.

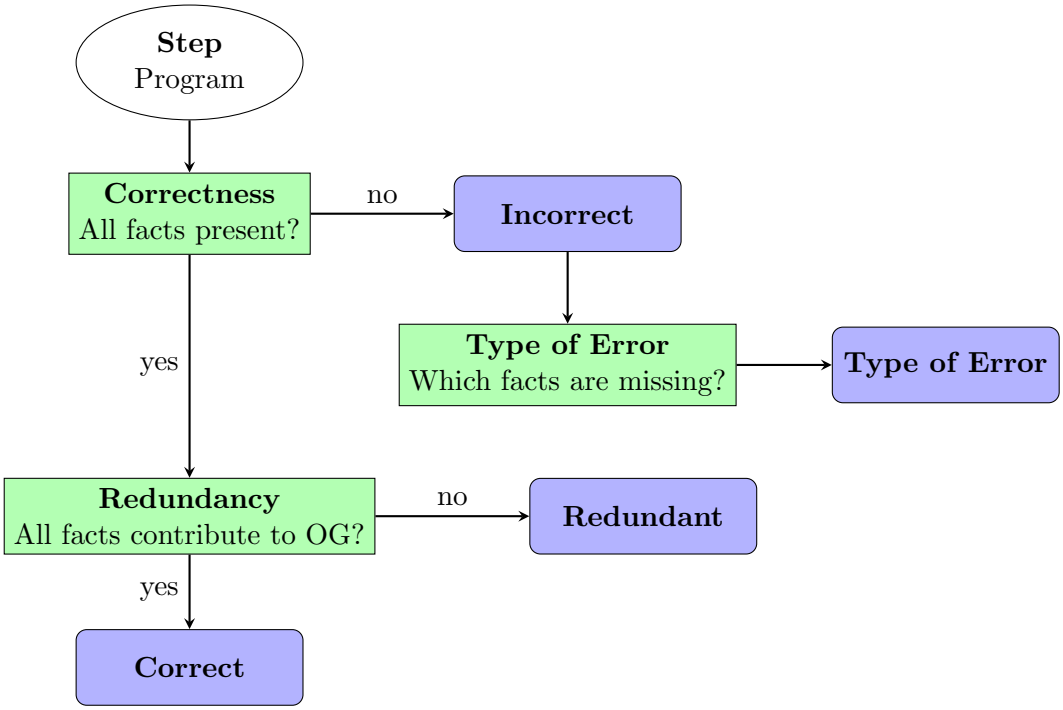

Figure 14: PHP ITS

5 Discussion

The aspects that ITSs diagnose are Correctness, Difference, Redundancy, Type of Error, Common Errors, Order, Preference and Time. Of these, Correctness is the most common. It is diagnosed in all systems. Correctness can be considered the most basic diagnostic aspect because other aspects depend on it. For instance, Type of Error relies on Correctness, because you must first establish that there is an error to determine what kind of error it is. Preference also depends on Correctness, because it must first be established that an answer is correct to determine whether it is the preferred correct answer.

Figure 15 illustrates the general diagnostic process. A dashed border indicates that the components are optional. All tutors check whether a step is correct. This is done using Correctness or Difference. Before this is done, however, some tutors check the Order of steps or check how much Time it took to submit the step. After it has been determined that a step is correct, some tutors check whether a step is Preferred. Some tutors also check whether a correct step is Redundant. For incorrect steps, some tutors check whether the step contains Common Errors, and what Type of Error was made. Lastly, some tutors check whether an incorrect step is redundant. Note that, as was mentioned before, some tutors consider Redundancy as an error, while others treat it as correct.

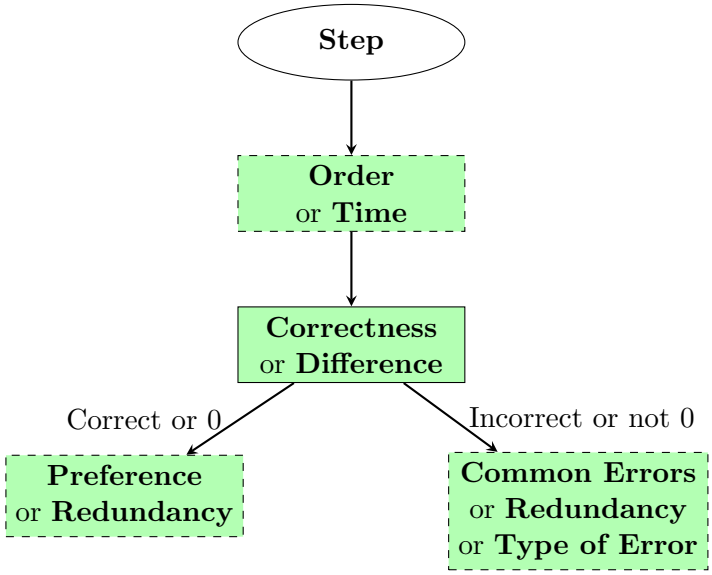

Figure 15: General diagnostic process

After Correctness, the most commonly diagnosed aspects are Type of Error and Common Errors. The frequency of these aspects differs per domain. Common Errors is the second most frequently diagnosed aspect in mathematics tutors, while that is Type of Error in programming tutors. This is perhaps due to the solution space in each domain.

In mathematics, problems typically have only a single correct solution, and there are only a few ways to get to the solution. This makes buggy rules a very suitable approach to finding errors. Since the solution space is small, the kinds of errors that students make can be anticipated relatively easily. In the domain of programming, the solution space is usually very large, which makes Common Errors less feasible. Type of Error is perhaps easier to determine in programming than in mathematics. In programming, the most basic categorization of errors is to make a distinction between syntactic and semantic errors. A syntactic error is easy to detect because a program will not run when it has a syntactic error. In mathematics, syntactic errors can also occur but are much less common than in programming errors. More research is needed to further investigate this difference between domains.

Only one ITS, Zatarain-Cabada13, diagnoses Time. It does this because the time it takes to answer a question is thought to reflect the difficulty of the question. Since there is only one ITS that does this, it seems to be an outlier. Why don't more ITSs diagnose time? It can be argued that Time is not a good measure of difficulty. Most ITSs can be accessed at home, without supervision. This makes it difficult to monitor how much time is actually spent on answering a question. A student might take a long time to answer simply because she is taking a break away from the computer, or because she is doing something else at the same time. Perhaps this is why most ITSs do not use Time for diagnosis.

Some ITSs make more fine-grained diagnoses than the ones discussed in this study. Arends17 [51] builds on the IDEAS framework [13] but provides additional diagnose services. This is because it is an ITS for the domain of microcontroller I/O programming, in which expressions can be semantically equivalent even when an incorrect step has been submitted. To handle these situations, the ITS can diagnose expressions that are semantically equivalent while also following a buggy rule, or expressions that are expected by a strategy despite not being semantically equivalent. Since these types of diagnoses are very specific to a domain, they are too fine-grained for the purposes of this research, which compares diagnostic processes across domains.

A limitation of this study is that it bases the analysis of diagnostic processes on the papers written about the ITSs, rather than on the source code of the ITSs. Unfortunately, not all papers provide an in-depth description of how diagnosis is done. Because of this it was not possible to describe the diagnostic process of some systems, since it is unclear how some aspects were determined. This also makes the analysis partly a matter of interpretation. For example, does a detour fall under Redundancy, Common Errors or Preference? In cases where I was unsure about an ITS, I consulted my supervisor for a second opinion.

I believe that the analysis of diagnose services in ITSs contributes to a better understanding of the diagnosing behavior of ITSs. For future research, the results of this study could be combined with results from evaluations of the effectiveness of tutoring systems (e.g. [3]). This would give insight into which diagnostic processes are most effective at improving learning. This insight could then inform the design and development of tutoring systems in the future.

6 Conclusion

The goal of this thesis was to explore the literature about the different Intelligent Tutoring Systems that are available, and gain insight into the diagnostic process of these systems. The research questions that this study aims to answer were the following: Firstly, what aspects can be distinguished in the diagnosis of student's responses? Secondly, how are these diagnostic aspects determined in the learning environment? Lastly, are there patterns or a general scheme that can be identified in the diagnostic process of the different learning environments? I.e. are there aspects that are diagnosed in all systems, and do they diagnose student's responses in a similar manner across domains and approaches?

Concerning the first question, there are eight aspects of student's responses that ITSs diagnose: Correctness, Difference, Redundancy, Type of Error, Common Errors, Order, Preference and Time. The second question was answered in section 4.4. Unfortunately, it was not possible to represent every diagnostic process in a diagram. Although the diagrams vary greatly between systems, a general model describing all ITSs was proposed in section5. Most importanly, all ITSs diagnose Correctness. There do seem to be differences in domains and approaches. The main difference in domains is that Common Errors is the second most frequently diagnosed aspect in mathematics tutors, while that is Type of Error in programming tutors. The analysis found no difference between the four tutoring approaches.

7 References

[1] Douglas C Merrill, Brian J Reiser, Michael Ranney, and J Gregory Trafton. Effective tutoring techniques: A comparison of human tutors and intelligent tutoring systems. *The Journal of the Learning Sciences*, 2(3):277–305, 1992.

[2] Kurt VanLehn. The behavior of tutoring systems. *International journal of artificial intelligence in education*, 16(3):227–265, 2006.

[3] Kurt VanLehn. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46(4):197–221, 2011.

[4] John R Anderson, C Franklin Boyle, and Brian J Reiser. Intelligent tutoring systems. *Science(Washington)*, 228(4698):456–462, 1985.

[5] John Seely Brown and Kurt VanLehn. Repair theory: A generative theory of bugs in procedural skills. *Cognitive science*, 4(4):379–426, 1980.

[6] John R Anderson, C Franklin Boyle, Albert T Corbett, and Matthew W Lewis. Cognitive modeling and intelligent tutoring. *Artificial intelligence*, 42(1):7–49, 1990.

[7] Antonija Mitrovic, Pramuditha Suraweera, and Brent Martin. Intelligent tutors for all: The constraint-based approach. *IEEE Intelligent Systems*, 22(4):38–45, 2007.

[8] Kenneth R Koedinger, Vincent Aleven, Neil Heffernan, Bruce McLaren, and Matthew Hockenberry. Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. In *International Conference on Intelligent Tutoring Systems*, pages 162–174. Springer, 2004.

[9] W Lewis Johnson. *Intention-based diagnosis of novice programming errors*. Morgan Kaufmann, 1986.

[10] Susanne Narciss. Feedback strategies for interactive learning tasks. *Handbook of research on educational communications and technology*, 3:125–144, 2008.

[11] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. Towards a systematic review of automated feedback generation for programming exercises. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 41–46. ACM, 2016.

[12] Hienke Keuning, Johan Jeuring, and Bastiaan Heeren. A systematic review of automated feeback generation for programming exercises. *Manuscript in preparation*, 2017.

[13] Bastiaan Heeren and Johan Jeuring. Feedback services for stepwise exercises. *Science of Computer Programming*, 88:110–129, 2014.

[14] Paul Black and Dylan Wiliam. Assessment and classroom learning. *Assessment in Education: principles, policy & practice*, 5(1):7–74, 1998.

[15] Michelene TH Chi, Stephanie A Siler, and Heisawn Jeong. Can tutors monitor students' understanding accurately? *Cognition and instruction*, 22(3):363–387, 2004.

[16] Anthony J Nitko and Susan M Brookhart. *Educational Assesment of Students 6nd Edition*. Boston: Pearson Education, Inc, 2011.

[17] Thierry Chanier, Michael Pengelly, Michael Twidale, and John Self. Conceptual modelling in error analysis in computer-assisted language learning systems. In *Intelligent tutoring systems for foreign language learning*, pages 125–150. Springer, 1992.

[18] Naima El-Kechaï, Élisabeth Delozanne, Dominique Prévit, Brigitte Grugeon, and Françoise Chenevotot. Evaluating the performance of a diagnosis system in school algebra. In *International Conference on Web-Based Learning*, pages 263–272. Springer, 2011.

[19] Alan CK Cheung and Robert E Slavin. The effectiveness of educational technology applications for enhancing mathematics achievement in k-12 classrooms: A meta-analysis. *Educational research review*, 9:88–113, 2013.

[20] James C Lester, Brian A Stone, Michael A O'Leary, and Robert B Stevenson. Focusing problem solving in design-centered learning environments. In *International Conference on Intelligent Tutoring Systems*, pages 475–483. Springer, 1996.

[21] James C Lester, Brian A Stone, and Gary D Stelling. Lifelike pedagogical agents for mixed-initiative problem solving in constructivist learning environments. *User modeling and user-adapted interaction*, 9(1-2):1–44, 1999.

[22] Scott D Johnson et al. Application of cognitive theory to the design, development, and implementation of a computer-based troubleshooting tutor. 1992.

[23] Benny G Johnson, Fred Phillips, and Linda G Chase. An intelligent tutoring system for the accounting cycle: Enhancing textbook homework with artificial intelligence. *Journal of Accounting Education*, 27(1):30–39, 2009.

[24] Neil T Heffernan and Kenneth R Koedinger. An intelligent tutoring system incorporating a model of an experienced human tutor. In *International Conference on Intelligent Tutoring Systems*, pages 596–608. Springer, 2002.

[25] Foteini Grivokostopoulou, Isidoros Perikos, and Ioannis Hatzilygeroudis. An educational system for learning search algorithms and automatically assessing student performance. *International Journal of Artificial Intelligence in Education*, 27(1): 207–240, 2017.

[26] Chee-Kit Looi. Automatic debugging of prolog programs in a prolog intelligent tutoring system. *Instructional Science*, 20(2-3):215–263, 1991.

[27] Arthur C Graesser, Peter Wiemer-Hastings, Katja Wiemer-Hastings, Derek Harter, Tutoring Research Group Tutoring Research Group, and Natalie Person. Using latent semantic analysis to evaluate the contributions of students in autotutor. *Interactive learning environments*, 8(2):129–147, 2000.

[28] Grażyna Demenko, Agnieszka Wagner, and Natalia Cylwik. The use of speech technology in foreign language pronunciation training. *Archives of Acoustics*, 35(3): 309–329, 2010.

[29] Elizabeth Arnott, Peter Hastings, and David Allbritton. Research methods tutor: Evaluation of a dialogue-based tutoring system in the classroom. *Behavior Research Methods*, 40(3):694–698, 2008.

[30] Jon Wetzel, Kurt VanLehn, Dillan Butler, Pradeep Chaudhari, Avaneesh Desai, Jingxian Feng, Sachin Grover, Reid Joiner, Mackenzie Kong-Sivert, Vallabh Patade, et al. The design and development of the dragoon intelligent tutoring system for model construction: lessons learned. *Interactive Learning Environments*, 25(3): 361–381, 2017.

[31] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. Strategy-based feedback in a programming tutor. In *Proceedings of the Computer Science Education Research Conference*, pages 43–54. ACM, 2014.

[32] Dinesha Weragama and Jim Reye. Analysing student programs in the php intelligent tutoring system. *International Journal of Artificial Intelligence in Education*, 24 (2):162–188, 2014.

[33] Miguel Arevalillo-Herráez, David Arnau, and Luis Marco-Giménez. Domain-specific knowledge representation and inference engine for an intelligent tutoring system. *Knowledge-Based Systems*, 49:97–105, 2013.

[34] Ido Roll, Vincent Aleven, and Kenneth R Koedinger. The invention lab: Using a hybrid of model tracing and constraint-based modeling to offer intelligent support in inquiry environments. In *International Conference on Intelligent Tutoring Systems*, pages 115–124. Springer, 2010.

[35] WeeSan Lee, Ruwanee de Silva, Eric J Peterson, Robert C Calfee, and Thomas F Stahovich. Newton's pen: A pen-based tutoring system for statics. *Computers & Graphics*, 32(5):511–524, 2008.

[36] Albert T Corbett, John R Anderson, and Eric G Patterson. Student modeling and tutoring flexibility in the lisp intelligent tutoring system. *Intelligent tutoring systems: At the crossroads of artificial intelligence and education*, pages 83–106, 1990.

[37] John R Anderson and Brian J Reiser. The lisp tutor. *Byte*, 10:159–175, 1985.

[38] Albert T Corbett and John R Anderson. Locus of feedback control in computer-based tutoring: Impact on learning rate, achievement and attitudes. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 245–252. ACM, 2001.

[39] Vincent Aleven, Octav Popescu, and Kenneth R Koedinger. Towards tutorial dialog to support self-explanation: Adding natural language understanding to a cognitive tutor. In *Proceedings of Artificial Intelligence in Education*, pages 246–255. Citeseer, 2001.

[40] Vincent Aleven, Octav Popescu, and Kenneth Koedinger. Pilot-testing a tutorial dialogue system that supports self-explanation. In *International Conference on Intelligent Tutoring Systems*, pages 344–354. Springer, 2002.

[41] John R Anderson, C Franklin Boyle, and Gregg Yost. The geometry tutor. In *IJCAI*, pages 1–7, 1985.

[42] Vincent Aleven, Bruce M McLaren, and Jonathan Sewall. Scaling up programming by demonstration for intelligent tutoring systems development: An open-access web site for middle school mathematics learning. *IEEE Transactions on Learning Technologies*, 2(2):64–78, 2009.

[43] Johan Jeuring, Alex Gerdes, and Bastiaan Heeren. A programming tutor for haskell. In *Central European Functional Programming School*, pages 1–45. Springer, 2012.

[44] Richard R Burton and John Seely Brown. A tutoring and student modelling paradigm for gaming environments. *ACM SIGCUE Outlook*, 10(SI):236–246, 1976.

[45] Dimitrios Sklavakis and Ioannis Refanidis. An individualized web-based algebra tutor based on dynamic deep model tracing. In *Hellenic Conference on Artificial Intelligence*, pages 389–394. Springer, 2008.

[46] Dimitrios Sklavakis and Ioannis Refanidis. Mathesis: An intelligent web-based algebra tutoring school. *International Journal of Artificial Intelligence in Education*, 22(4):191–218, 2013.

[47] Nakhoon Kim, Martha Evens, Joel A Michael, and Allen A Rovick. Circsim-tutor: An intelligent tutoring system for circulatory physiology. In *International Conference on Computer Assisted Learning*, pages 254–266. Springer, 1989.

[48] Michael Glass. Some phenomena handled by the circsim-tutor version 3 input understander. In *Proceedings of the Tenth Florida Artificial Intelligence Research Symposium, Daytona Beach*, pages 21–25. Citeseer, 1997.

[49] Kurt Vanlehn, Collin Lynch, Kay Schulze, Joel A Shapiro, Robert Shelby, Linwood Taylor, Don Treacy, Anders Weinstein, and Mary Wintersgill. The andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education*, 15(3):147–204, 2005.

[50] Ramón Zatarain-Cabada, María Lucía Barrón-Estrada, Yasmín Hernández Pérez, and Carlos Alberto Reyes-García. Designing and implementing affective and intelligent tutoring systems in a learning social network. In *Mexican International Conference on Artificial Intelligence*, pages 444–455. Springer, 2012.

[51] Hugo Arends, Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. An intelligent tutor to learn the evaluation of microcontroller i/o programming expressions. In *Proceedings of the 17th Koli Calling Conference on Computing Education Research*, pages 2–9. ACM, 2017.

[52] Kurt VanLehn, Pamela W Jordan, Carolyn P Rose, Dumisizwe Bhembe, Michael Bottner, Andy Gaydos, Maxim Makatchev, Umarani Pappuswamy, Michael Ringenberg, Antonio Roque, et al. The architecture of why2-atlas: A coach for qualitative physics essay writing. In *International Conference on Intelligent Tutoring Systems*, pages 158–167. Springer, 2002.

[53] Arthur C Graesser, Shulan Lu, George Tanner Jackson, Heather Hite Mitchell, Mathew Ventura, Andrew Olney, and Max M Louwerse. Autotutor: A tutor with dialogue in natural language. *Behavior Research Methods, Instruments, and Computers*, 36(2):180–192, 2004.

[54] Kenneth R Koedinger and John R Anderson. Reifying implicit planning in geometry: Guidelines for model-based intelligent. *Computers as cognitive tools*, page 15, 2013.

[55] Leena M Razzaq, Mingyu Feng, Goss Nuzzo-Jones, Neil T Heffernan, Kenneth R Koedinger, Brian Junker, Steven Ritter, Andrea Knight, Edwin Mercado, Terrence E Turner, et al. Blending assessment and instructional assisting. In *AIED*, pages 555–562, 2005.

[56] Glenn Blank, Shahida Parvez, Fang Wei, and Sally Moritz. A web-based its for oo design. In *Proceedings of Workshop on Adaptive Systems for Web-based Education at 12th International Conference on Artificial Intelligence in Education (AIED'2005). Amsterdam, the Netherlands*, pages 59–64, 2005.

[57] JS Song, SH Hahn, KY Tak, and JH Kim. An intelligent tutoring system for introductory c language course. *Computers & Education*, 28(2):93–102, 1997.

[58] Gerhard Weber and Peter Brusilovsky. Elm-art: An adaptive versatile system for web-based instruction. *International Journal of Artificial Intelligence in Education (IJAIED)*, 12:351–384, 2001.

[59] David Arnau, Miguel Arevalillo-Herráez, Luis Puig, and José Antonio González-Calero. Fundamentals of the design and the operation of an intelligent tutoring system for the learning of the arithmetical and algebraic way of solving word problems. *Computers & Education*, 63:119–130, 2013.

[60] Jun Hong. Guided programming and automated error analysis in an intelligent prolog tutor. *International Journal of Human-Computer Studies*, 61(4):505–534, 2004.

[61] Davide Fossati, Barbara Di Eugenio, Stellan Ohlsson, Christopher Brown, and Lin Chen. Data driven automatic feedback generation in the ilist intelligent tutoring system. *Technology, Instruction, Cognition and Learning*, 10(1):5–26, 2015.

[62] Kelly Rivers and Kenneth R. Koedinger. Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *International Journal of Artificial Intelligence in Education*, 27(1):37–64, 2017.

[63] Wei Jin, Tiffany Barnes, John Stamper, Michael John Eagle, Matthew W Johnson, and Lorrie Lehmann. Program representation for automatic hint generation for a data-driven novice programming tutor. In *International Conference on Intelligent Tutoring Systems*, pages 304–309. Springer, 2012.

[64] Wei Jin, Albert Corbett, Will Lloyd, Lewis Baumstark, and Christine Rolka. Evaluation of guided-planning and assisted-coding with task relevant dynamic hinting. In *International Conference on Intelligent Tutoring Systems*, pages 318–328. Springer, 2014.

[65] Edward R Sykes. Design, development and evaluation of the java intelligent tutoring system. *Technology, Instruction, Cognition & Learning*, 8(1), 2010.

[66] Pramuditha Suraweera and Antonija Mitrovic. Kermit: A constraint-based tutor for database modeling. In *International Conference on Intelligent Tutoring Systems*, pages 377–387. Springer, 2002.

[67] Patricia A Jaques, Henrique Seffrin, Geiseane Rubi, Felipe de Morais, Cássio Ghilardi, Ig Ibert Bittencourt, and Seiji Isotani. Rule-based expert systems to support step-by-step guidance in algebraic problem solving: The case of the tutor pat2math. *Expert Systems with Applications*, 40(14):5456–5465, 2013.

8 Appendix A

ITS	Domain	M.	E.	C.	I.
(Why2-)Atlas [52]	Qualitative physics	•			
(Why2-)Autotutor [27], [53]	Physics & Computer literacy	•			
ACT Programming Tutor [38]	Programming	•			
AITS [25]	Search algorithms		•		
Andes [49]	Physics	•		•	
ANGLE [54]	Geometry	•			
APROPOS2 [26]	Prolog programming		•		
Ask-Elle [43]	Haskell programming	•		•	
Assistment [55]	Mathematics	•			
AzAR 3.0 [28]	Foreign language pronunciation		•		
CIMEL ITS [56]	OO design and programming	•			
CIRCSIM-TUTOR [47], [48]	Circulatory physiology	•			
C-Tutor [57]	C programming				•
Design-A-Plant [20], [21]	Botany			•	
Dragoon [30]	Dynamic systems		•		
ELM-ART [58]	LISP programming	•			
Geometry Explanation Tutor [39], [40]	Geometry	•			
Geomtery Tutor [41]	Geometry	•			
HBPS [33], [59]	Algebra word problems	•			
Hong04 [60]	Prolog programming	•			
iList [61]	Computer Science			•	
ITAP [62]	Python programming		•		
Jin12 [63]	Programming		•		
Jin14 [64]	Programming		•		
JITS [65]	Java programming	•			
KERMIT [66]	Database design			•	
Keuning14 [31]	Imperative programming	•			
Mathesis [45], [46]	Algebra	•			
Mathtutor [42]	Mathematics		•	•	
Ms. Lindquist [24]	Algebra word problems	•			
Newton's Pen [35]	Statics	•		•	
PAT2Math [67]	Algebra	•			
PHP ITS [32]	PHP programming			•	
PLATO [44]	Arithmetic			•	
Quantum Accounting [23]	Accounting	•			
RMT [29]	Psychology research methods	•			
Technical Troubleshooting Tutor [22]	Aircraft engineering	•?		•?	
The Invention Lab [34]	Scientific inquiry	•		•	
The LISP Tutor [36], [37]	LISP programming	•			
Zatarain-Cabada13 [50]	Arithmetic	•			

Table 10: M. = Model tracing, E. = Example tracing, C. = Constraint-based, I. = Intention-based

9 Appendix B

ITS	Corr.	D.	R.	T.o.E.	Comm.	O.	P.	T.
(Why2-)Atlas [52]	•			•	•			
(Why2-)Autotutor [27], [53]	•	•			•			
ACT Programming Tutor [38]	•				•			
AITS [25]	•	•	•	•				
Andes [49]	•			•			•	
ANGLE [54]	•				•			
APROPOS2 [26]	•	•	•	•	•	•	•	
Ask-Elle [43]	•			•	•		•	
Assitment [55]	•				•			
AzAR 3.0 [28]	•	•		•	•			
CIMEL ITS [56]	•			•				
CIRCSIM-TUTOR [47], [48]	•			•		•		
C-Tutor [57]	•			•				
Design-A-Plant [20], [21]	•							
Dragoon [30]	•		•	•				
ELM-ART [58]	•			•				
Geometry Explanation Tutor [39], [40]	•			•	•			
Geomtery Tutor [41]	•				•			
HBPS [33], [59]	•			•	•			
Hong04 [60]	•			•				
iList [61]	•			•	•			
ITAP [62]	•			•				
Jin12 [63]	•			•				
Jin14 [64]	•			•				
JITS [65]	•			•				
KERMIT [66]	•			•				
Keuning14 [31]	•		•	•			•	
Mathesis [45], [46]	•			•	•	•		
Mathtutor [42]	•				•	•		
Ms. Lindquist [24]	•			•	•			
Newton's Pen [35]	•			•				
PAT2Math [67]	•				•			
PHP ITS [32]	•		•	•				
PLATO [44]	•			•	•		•	
Quantum Accounting [23]	•			•				
RMT [29]	•	•						
Technical Troubleshooting Tutor [22]	•							
The Invention Lab [34]	•			•	•			
The LISP Tutor [36], [37]	•				•			
Zatarain-Cabada13 [50]	•							•

Table 11: Corr. = Correctness, D. = Difference, R. = Redundancy, T.o.E. = Type of Error, Comm. = Common Errors, O. = Order, P. = Preference, T. = Time