Automatic Assessment within Intelligent Tutoring Systems

By S.M. Brommer

Automatic Assessment within Intelligent Tutoring Systems

By

S.M. Brommer

Department of Philosophy

Faculty of Humanities

Utrecht University

A thesis in partial fulfilment of

the requirements for the degree of

Bachelor of Science

Cognitive Artifical Intelligence

7.5 ECTS

Supervisor and first assessor:

Prof. Dr. J.T. Jeuring

Second assessor:

Prof. Dr. J-J.Ch. Meyer

7th May 2014

Abstract

Abstract: Intelligent Tutoring Systems are an artificial-intelligence approach to computer-assisted learning. Automatic assessment is an important part of these systems. Keeping track of a student's knowledge by means of automatic assessment allows the Intelligent Tutoring System to grade this student, and find suitable exercises for the student to practice with. This thesis gives an overview of automatic assessment within Intelligent Tutoring Systems.

Keywords: automatic assessment, intelligent tutoring system, student model, domain reasoner, artificial teaching

“Computer presence will enable us to modify the learning environment outside the classroom so that much, if not all, of the knowledge schools presently try to teach with such pain, expense and limited success, will be learned as the child learns to talk, painlessly, successfully and without instruction. Schools as we know them today have no place in the future.”

Seymour Papert, 1980

Acknowledgements

First and foremost, I wish to thank my supervisor Johan Jeuring. He guided me through this project by providing lots of advice, suggestions, and insights. Johan has taught me a lot about both the subject of this thesis, and academic writing. I enjoyed our discussions about this project.

I would also like to thank Steven Langerwerf for all the discussions we had about the subject, and for reading my thesis several times. His remarks have significantly contributed to the quality of this thesis.

Furthermore, I would like to thank my parents for their support and interest during my bachelor. I particularly want to thank Martin Brommer, my father, whose critical remarks have lead to improvement of this thesis.

Lastly, I would like to thank my professors and fellow students for making me a more critical thinker and a better writer.

Contents

Introduction

Approaches to artificial intelligence can be broadly divided into four groups: thinking humanly, acting humanly, thinking rationally, and acting rationally [14]. This thesis will adopt the acting-humanly approach, which considers artificial intelligence to be “the art of creating machines that perform functions that require intelligence when performed by people” [7]. This thesis will focus on creating a machine that is able to teach students. Teaching is considered to require intelligence when performed by people [16], and a teaching machine may therefore be called an artificial intelligent teacher.

In the field of computer-assisted education, artificial intelligent teachers are called ‘Intelligent Tutoring Systems’ (ITSs). One task for which ITSs may be used is automatic assessment. The research question of this thesis is: how can automatic assessment be implemented in ITSs? This question will be answered using both literature research, and an abstract implementation of automatic assessment within an ITS. The focus will be on an ITS developed at Utrecht University and the Open Universiteit [3].

Computer-assisted education focusses on presenting educational material by means of a computer [5]. ITSs are based on an artifical-intelligence approach to computer-assisted education [10]. The idea behind this approach is that the computer does not only act as a tool to instruct the student, but acts as an artificial teacher as well. An ITS analyses interactions between a student and the system, gives feedback on the student actions, and helps the student make progress [3].

There are huge differences between the architectures of existing ITSs, and therefore it is nearly impossible to describe an architecture which fits all of these systems. However, most ITSs have a structure based on the following four components [10]. This structure is visualised in Figure 1.

The user interface module allows the student to interact with the system.

The tutoring module selects which material will be presented to the student.

The domain reasoner or expert knowledge module is a content database consisting of a collection of exercises within the domain of the ITS. The domain reasoner can reason about these exercises and knows how objects of the domain can be manipulated to solve these exercises [3].

The student model module keeps track of the student’s progress. This module infers the student’s knowledge from the interaction between a student and the system, and represents this knowledge properly.

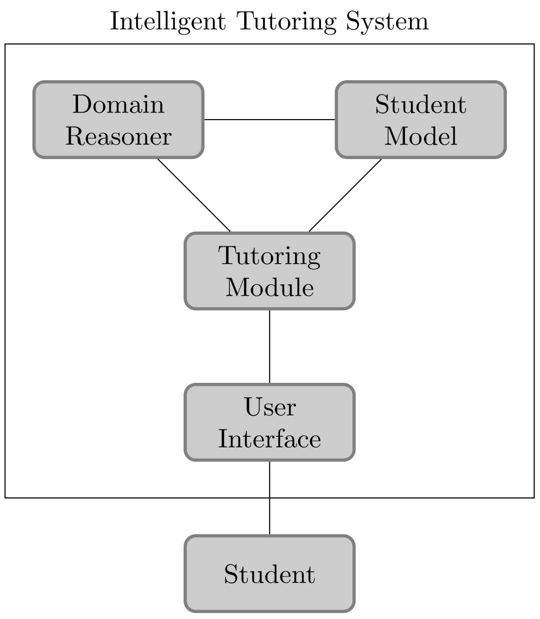

Figure 1: Implementation of knowledge

An ITS as described above acts like an artificial teacher. Among the tasks of a teacher are grading the student, and offering the student suitable exercises. For an ITS to do this, it needs to be able to keep track of the student's progress. This is done by automatic assessment.

Automatic assessing is measuring a student's knowledge about a certain learning domain by means of a computer [1]. The outcome of this assessment is a representation of the student's knowledge. Within an ITS, automatic assessment is based on the interactions between the student and the domain reasoner. The student model receives this information and uses it to update its representation of the student's knowledge.

Automatic assessment is highly useful for learning mathematical concepts, and it can lead to increased flexibility in teaching [12]. It encourages students to correctly solve problems, and provides teachers easy insight into the progress of their students [2]. Another useful consequence of automatic assessment is that it makes it possible to analyse student errors while they are still practicing. This analysis is for human teachers often more time-consuming than examining scores, but it can increase the quality of education [13]. Other studies have shown that the long term use of automatic assessment is relatively cheap [12].

The student's knowledge obtained by automatic assessment is represented for different domains. What the relevant knowledge domains are depends on the learning goal of a student. We assume that the main learning goal of a student is the ability to solve problems from a certain exercise class. This requires knowledge about strategies and the efficiency of these strategies. Knowledge about strategies requires knowledge about the rules used in this strategy, and the procedure or sequence of these rules. Knowledge about syntax is beyond the scope of this thesis.

This thesis is ordered as follows. Chapter 1 gives a brief overview of the domain reasoner and the information about interactions provided by the domain reasoner. Chapter 2 focusses on the student model and updating the knowledge representation. Chapter 3 provides a summary of our findings and suggestions for further research. The programming language used to implement our approach is Haskell [6], and the full implementation can be found in Appendix A.

Chapter 1

Diagnosing Using a Domain Reasoner

A domain reasoner is a component of an ITS which has knowledge about one or more exercise domains, and has the ability to reason about these domains. The domain reasoner is able to diagnose a student's action. This chapter gives an overview of this diagnosing by a domain reasoner. The focus will be on the domain reasoner from the ITS developed at Utrecht University and the Open Universiteit [3]. Section 1 will present the possible interactions between the student and the domain reasoner. In Section 2 the diagnoses from the domain reasoner and their compatibility with the student model will be discussed. Section 3 provides a proposal for new diagnoses.

1.1 Interaction

Representation

The representation of a student's knowledge in the student model is obtained from the interaction between the student and the domain reasoner, when solving a stepwise exercise. A step is the rewriting of a term to a new term. For example, a step may be the rewriting of the term $8x = 16$ to the term $x = 2$. Note that terms may take other forms, depending on the knowledge domain of the ITS. Terms may be a logical expression, or an expression from a programming language. This thesis will focus on mathematical expressions.

An interaction between the student and the domain reasoner consists of a request from the student and the response of the domain reasoner to that request. Bastiaan Heeren et al. states the following request types [3]:

A request for diagnosis of a step The student asks the domain reasoner to diagnose a step. The domain reasoner will evaluate this step, and provide information to the student. For example, it may diagnose a step as being 'correct' or 'not equivalent'.

A request to check whether the problem is solved The domain reasoner will take the last term provided by the student, and respond whether it is a correct solution.

A request for possible next steps The domain reasoner will take the last term provided by the student, and suggest possible next steps. Each suggested step will take the term closer to being solved. Only the name of the step will be suggested; it is up to the student to perform the step.

A request for a worked out next step The domain reasoner will take the last term provided by the student, determine the best next step and perform it. The response to the student will be both the name of the step, and the next term.

A request for the solution for the exercise The domain reasoner will take the last term provided by the student, and work out the term until it is solved. The response to the student will be all the taken steps, and all terms.

The domain reasoner receives extra input with the requests. For all requests, the domain reasoner receives the type of the exercise the student is trying to solve. For some requests, the domain reasoner receives the current and previous terms that are analysed as well. As will be shown in the next section, the student model only needs information about the student's request and the exercise type. Therefore, information about the current and previous terms of an exercise will be ignored in the automatic assessment.

The response of the domain reasoner is different for each kind of request and, as with the input, not all information from the output is relevant for the student model. Specifically, only the diagnosis of a step, and whether a problem is solved, are relevant information for the student model. Therefore, the other responses of the domain reasoner will be ignored in the automatic assessment.

Implementation

The request types and relevant feedback from the domain reasoner are modelled as values of the data type `Interaction`. This is shown in Figure 1.1. Each `Interaction` represents a requests. All `Interaction`s carry information about the type of the `Exercise` for which the request was made.

The `Interaction` representing the `CheckStep` request also holds information about the `Diagnosis`. How this `Diagnosis` is implemented will be discussed later in this chapter.

The `Interaction` representing the `CheckSolved` request holds extra information as well, namely whether or not the exercise is solved. This is represented as the Boolean `Solved`, where `True` indicates that the exercise is solved, and `False` indicates that it is not solved.

An `Exercise` type is represented by the `Problem` of which this exercise is an example, the `Strategy` by means of which this exercise should be solved, and the `Rule` the student used to rewrite the term [4]. These are all identified by a `String`.

```
data Exercise = Exercise Problem Strategy Rule

type Problem  = String
type Strategy = String
type Rule     = String

data Interaction =
    CheckStep   Diagnosis Exercise
  | CheckSolved Solved    Exercise
  | NextSteps             Exercise
  | WorkedOutStep         Exercise
  | Solution              Exercise

type Solved = Bool
```

Figure 1.1: Implementation of Interaction

The next example illustrates this representation. Consider a quadratic expression, like $x^2 + 5x + 6 = 0$. This expression can be solved by the simple factorisation strategy, or the quadratic formula strategy. The latter strategy consists of the following rules:

1. Rewrite to the correct form: $x = \frac{-5 \pm \sqrt{5^2 - 4 \cdot 1 * 6 \cdot}}{2 \cdot 1}$
2. Solve denominator: $x = \frac{-5 \pm \sqrt{5^2 - 4 \cdot 1 \cdot 6}}{2}$
3. Solve discriminant: $x = \frac{-5 \pm \sqrt{1}}{2}$
4. Solve numerator: $x = \frac{-6}{2}$ or $x = \frac{-4}{2}$
5. Solve for x: $x = -3$ or $x = -2$.

Suppose a student tries to solve a quadratic expression, using the quadratic formula strategy. She rewrites a term according to the 'solve discriminant'-rule. The information about the type of the `Exercise` is represented as `Exercise "quadratic expression" "quadratic formula" "solve discriminant"`.

If this student now makes a request to check whether the term is solved, and the domain reasoner concludes that the term is not solved, the `Interaction` is represented as `CheckSolved False (Exercise "quadratic expression" "quadratic formula" "solve discriminant")`

1.2 Existing diagnoses

Bastiaan Heeren et al. states that the domain reasoner may give the following six diagnoses [3]:

Not equivalent The new term is not equivalent to the old term. It is not known whether the procedure is followed, and no buggy rule is recognised.

Correct The new term is equivalent to the old term. It is not known whether the procedure is followed, and no rule is recognised.

Buggy A certain rule is applied incorrectly. That is, a buggy version of this rule is applied. It is not known whether the procedure is followed, but the buggy rule is recognised.

Expected A certain rule is applied correctly, as expected. This rule follows the procedure, and is recognised.

Detour A certain rule is applied correctly, but was not expected. This rule does not follow the procedure, but the rule is recognised.

Similar The new term is very similar to the old term (e.g. correct elimination of brackets).

The student model uses the interaction between a student and the domain reasoner to deduce information about the student's knowledge. Some requests indicate that the student lacks knowledge about a certain domain. Other requests may indicate that the student does have knowledge about a certain domain. What each interaction exactly indicates will be explained in the next chapter.

However, the diagnoses described above have some undesirable limitations with respect to the student model. In the next section, new diagnoses will be proposed. There are three reasons to prefer the use of these new diagnoses over the use of the existing diagnoses. These reasons will be explained below. As will become clear, the proposed diagnoses will be compatible with the existing diagnoses.

The first reason for changing the diagnoses is emphasising the different parts of the diagnoses. Except for the 'similar' diagnosis, all diagnoses can be split into two parts: one pertaining to the procedure, and one pertaining to the rule. A procedure is either followed, not followed, or this is unknown. A rule is either correctly applied, incorrectly applied, or this is unknown. The student model will make use of this distinction, since it keeps track of the student's knowledge concerning rules and procedures.

The second reason for changing the diagnoses is the distinction between optimal and suboptimal steps. The student model distinguishes knowledge about strategies, and knowledge about the efficiency of these strategies. A problem may be solved by several strategies. Some of these strategies are optimal, and some may be suboptimal. If a student correctly applies a rule following a certain strategy, this indicates that the student has knowledge about both the rule and the strategy. But only if this rule is the next step within an optimal strategy, it indicates that the student has knowledge about the efficiency of all compatible strategies. That is, if the rule is the next step within a suboptimal strategy, the student does not have knowledge about the efficiency of the compatible strategies, or else she would have followed another (optimal) strategy.

The third reason for changing the diagnoses is the incomplete information given by the 'buggy' diagnosis. This diagnosis does not hold any information about whether a procedure is followed. If the 'buggy' diagnosis is made, the student model can only deduce that a student does not have knowledge about the correct rule corresponding to the buggy rule. It can not deduce anything about the student's procedure knowledge. However, this is desirable for the student model. For example, if the next step according to a procedure is 'addition', but if the student applied a buggy addition rule, it is desirable that student model may still deduce that the student has knowledge about the procedure.

1.3 New diagnoses

Proposal

The three reasons discussed in the previous subsection indicate that the definitions of the existing diagnoses need to be revised. The new diagnoses will differ from the existing diagnoses in three ways:

- the procedure-rule distinction will be emphasised;
- a distinction between optimal and suboptimal steps will be made; and
- the buggy rule diagnosis indicate whether a strategy is followed.

To emphasise the distinction between a diagnosis about a procedure and a diagnosis about a rule, the new diagnosis will consist of two parts: the rule-rewriting diagnosis and the procedure diagnosis. The 'similar' diagnosis is an exception; this remains as it is.

The rewriting diagnosis has two properties: recognition and equivalence. The rewriting of a term to a new term wiil be diagnosed as either recognised or unrecognised, depending on whether a (buggy) rule is recognised. It will also be diagnosed as either equivalent or not equivalent, depending on whether the old term is equivalent to the new term.

The procedure diagnosis holds information about the optimality of a rewriting rule in the context of a procedure. A rule may be optimal, suboptimal, a detour, or – in case the rule is unrecognised – this may be unknown. A rule is optimal if, according to the domain reasoner, applying this rule is the best step towards solving the exercise. In case there are multiple rules considered to be the best step, these rules are all optimal. A rule is suboptimal if it takes the term closer to being solved, but it was possible to apply a better rule. A rule is a detour if it does not follow any strategy that solves the exercise.

As a result of these new diagnoses, information about whether a procedure is followed is available even if a rule is not applied correctly. That is, if a student applies a rule incorrectly, it can still be diagnosed that the student followed a procedure. In contrast, the existing diagnoses do not hold information about the procedure if a rule is applied incorrectly.

As will become clear in the next chapter, the new diagnoses make it easier to implement the student model. In addition, the more detailed diagnoses will make it more likely for the student model to be compatible with other, more detailed domain reasoners.

Implementation

In Figure 1.2 the implementation of the new diagnoses is shown. A `Diagnosis` is either `Similar` when no important changes are made, or a complex `Diagnosis` consisting of a `RewriteDiag` and a `ProcedureDiag`.

The `RewriteDiag` is a tuple of two `Booleans`: `Recognised`, which is `True` if and only if the rule is recognised, and `Equivalent`, which is `True` if and only if the new term is equivalent to the old term.

The `ProcedureDiag` is either `Optimal`, `Suboptimal`, `Detour` or `Unknown`.

```
data Diagnosis =
    Diagnosis RewriteDiag ProcedureDiag
  | Similar

type RewriteDiag =
    (Recognised, Equivalent)

type Recognised = Bool
type Equivalent = Bool

data ProcedureDiag =
    Optimal
  | Suboptimal
  | Detour
  | Unknown
```

Figure 1.2: Implementation of Diagnosis

Examples

This subsection will provide examples of the new diagnoses. Consider the term $x^2+5x+6=0$. The student may try to solve this term using simple factorisation (the optimal strategy), the quadratic formula (a suboptimal strategy), or by multiplying by two (a detour from any strategy). The student may rewrite this term to the following terms:

$x^2+6+5x=0$
: The student swapped the 6 and the $5x$ within the term. The diagnosis will be:
`Similar`.

$(2+x)(3+x)=0$
: The student used the optimal simple factorisation strategy. The rule is recognised and the terms are equivalent. The diagnosis will be:
`Diagnosis (True, True) Optimal`.

$x=\frac{-5\pm\sqrt{5^2-4\cdot 6}}{2}$
: The student used the suboptimal quadratic formula strategy. The rule is recognised and the terms are equivalent. The diagnosis will be:
`Diagnosis (True, True) Suboptimal`.

$2x^2+10x+12=0$
: The student multiplied the term by two, which is a detour from any strategy. This rule is recognised and the terms are equivalent. The diagnosis will be:
`Diagnosis (True, True) Detour`.

$(1+x)(2+x)=0$
: The student used the optimal simple factorisation strategy. The rule is recognised, but the terms are not equivalent. The diagnosis will be:
`Diagnosis (True, False) Optimal`.

$x = \frac{-1 \pm \sqrt{2^2 - 3 \cdot 4}}{2}$
: The student used the suboptimal quadratic formula strategy. The rule is recognised, but the terms are not equivalent. The diagnosis will be:
`Diagnosis (True, False) Suboptimal`.

$2x^2 + 10x + 6 = 0$
: The student multiplied the term by two, which is a detour from any strategy. This rule is recognised, but the terms are not equivalent. The diagnosis will be:
`Diagnosis (True, False) Detour`.

$x(x + 5 + \frac{6}{x}) = 0$
: The student did some rewriting. The rule is unrecognised, but the terms are equivalent. Since the rule is unrecognised, it is unknown whether a strategy is followed. The diagnosis will be:
`Diagnosis (False, True) Unknown`.

$8 \cdot 15x = 0$
: The student did some rewriting. The rule is unrecognised and the terms are not equivalent. Since the rule is unrecognised, it is unknown whether a strategy is followed. The diagnosis will be:
`Diagnosis (False, False) Unknown`.

Translating the existing diagnoses to the new diagnoses

The two ways to diagnose are compatible with each other. For the student model to be compatible to the domain reasoner described by Bastiaan Heeren et al., it must be possible to translate the existing diagnoses to the new. Although the new diagnoses have the potential to hold more information than the existing, it is still possible the formulate the existing diagnoses in terms of the new:

Not equivalent Some unrecognised rule is applied incorrectly. Both `Recognised` and `Equivalent` will be evaluated as `False`. It is `Unknown` whether the procedure is followed. Therefore, the new diagnosis will be:
`Diagnosis (False, False) Unknown`.

Correct Some unrecognised rule is applied correctly. `Recognised` will be evaluated as `False`, and `Equivalent` as `True`. It is `Unknown` whether the procedure is followed. Therefore, the new diagnosis will be:
`Diagnosis (False, True) Unknown`.

Buggy A certain rule is applied incorrectly. `Recognised` will be evaluated as `True`, and `Equivalent` as `False`. It is `Unknown` whether the procedure is followed. Therefore, the new diagnosis will be:
`Diagnosis (True, False) Unknown`.

Note that a domain reasoner probably is able to determine whether the correct rule corresponding to a buggy rule follows the procedure. In this case, the `ProcedureDiag` may take another value than `Unknown`.

Expected A certain rule is applied correctly, as expected. Both `Recognised` and `Equivalent` will be evaluated as `True`. The procedure is followed. So in this case both `Optimal` and `Suboptimal` are plausible values for `ProcedureDiag`. Since no expected rule is better then another, all expected rules are optimal. Therefore, the new diagnosis will be:
`Diagnosis (True, True) Optimal`.

Note that a domain reasoner that does make a distinction between optimality and suboptimality may assign the value `Suboptimal` to `ProcedureDiag`.

Detour A certain rule is applied correctly, but it was not expected. Both `Recognised` and `Equivalent` will be evaluated as `True`. In context of the procedure, the use of this rule is a `Detour`. Therefore, the new diagnosis will be:
`Diagnosis (True, True) Detour`.

Similar The term has been rewritten, but is very similar (e.g. correct elimination of brackets). This diagnosis will just be:
`Similar`.

Chapter 2

Automatic Assessment in a Student Model

The function of a student model within an ITS consists of assessing a student, and providing information about a student's level. This chapter will give an overview of automatic assessment [11]. Section 1 will explain how a student's knowledge is represented. Section 2 will explain how a student's knowledge is updated. Section 3 will discuss some applications of automatic assessment.

2.1 Knowledge Representation

Knowledge Domains

We assume that the main learning goal of a student is the ability to solve problems from a certain exercise class. This problem solving knowledge, can be divided into two aspects: knowledge about the strategies, and knowledge about the relative efficiency of these strategies [15]. That is, a student knows how to solve a certain problem correctly, if she knows how to apply suitable strategies, and also knows which of the strategies is most efficient given the problem.

Knowledge about a strategy, in turn, consists of knowledge about the rewriting rules, and knowledge about the procedure [4]. A procedure of a strategy is the sequence in which the rules should be applied. So a student knows how to apply a strategy, if he or she knows both how and in which order the rules should be applied. Note that procedure knowledge does not include rule rewriting knowledge. One might know which rule has to be applied next (procedure knowledge), without being able to correctly apply this rule (rewriting knowledge).

This hierarchy, visualised in Figure 2.1, shows that both problem solving knowledge and strategy knowledge can be expressed in terms of other knowledge domains. Therefore, it is only necessary to keep track of the three knowledge domains that can not be expressed in terms of other domains, which are:

- the domain of the rewriting of rules,
- the domain of the procedure of strategies, and
- the domain of the efficiency of solving a problem.

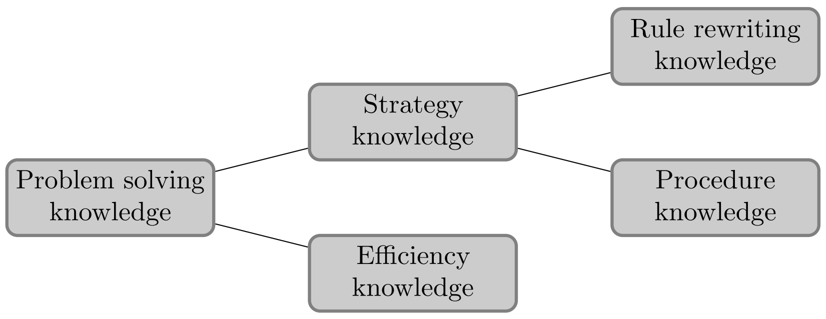

Figure 2.1: The hierarchy of problem solving knowledge

One way of representing the student's knowledge about a domain is a list of so called binary experiences. An experience can be qualified as either positive or negative. Interaction between the student and the ITS indicating that the student has knowledge about a certain domain, counts as a positive experience. On the other hand, interaction indicating that the student does not have knowledge about a certain domain, counts as a negative experience. What these indications exactly are will be discussed later in this section.

This knowledge representation is very simple. It can be argued that the range of binary experiences is too small, and that it is desirable to value an experience with a number. The time difference between two experiences may be important as well. However, if one wishes to use a more complex representation than a list of binary experiences, small adjustments should make this possible.

Implementation

Figure 2.2 shows the implementation of knowledge in the student model module. The knowledge of a student is implemented as a `Map`, which maps a certain knowledge `Domain` to the student's `Experience` within this `Domain`.

The `Domain` of knowledge is either the `Efficiency` of a certain `Problem`, the `Procedure` of a certain `Strategy`, or the `Rewriting` of a `Rule`. Each `Problem`, `Strategy` and `Rule` is identified by a `String`.

The student's `Experience` is represented as a list of `Boolean`s, where `True` represents a positive experience, and `False` represents a negative experience. New experiences will be added to the front of the list, so the most recent experience is the first item in the list.

For example, if a student tries to solve the quadratic expression $x^2+5x+6=0$ using the quadratic formula, and just correctly solved the term within the discriminant, then the three relevant `Domain`s are:

- `Efficiency "quadratic expression"` for the knowledge about efficiency of solving quadratic expressions,
- `Procedure "quadratic formula"` for the knowledge about the procedure of quadratic formula strategy, and
- `Rewriting "solve discriminant"` for the knowledge about the rule rewriting the discriminant.

```
type Knowledge = Map Domain Experience

data Domain =
    Efficiency Problem
  | Procedure  Strategy
  | Rewriting  Rule
```

Figure 2.2: Implementation of Knowledge

A knowledge map representing these three domains may look like this:

```
[(Efficiency "quadratic expression", [False]),
 (Procedure  "quadratic formula",    [True]),
 (Rewriting  "solve discriminant",   [True])]
```

2.2 Updating Knowledge

Every interaction between the student and the ITS may contain information indicating the student's knowledge about a certain domain. How these interactions are implemented is shown in Figure 1.1. This section will discuss what information can be deduced from these interactions.

The representation of the student's knowledge is updated with each interaction by a function called `updateK`. The implementation of this function is shown in Figure 2.3. This function takes two arguments: the `Interaction` and the `Knowledge` of the student model. The function returns the `Knowledge` updated with information from the `Interaction`. Recall that the `Knowledge` type is a `Map` from a knowledge `Domain` to the student's `Experience` within this domain.

Before going into detail about the interactions, the function `add` will be explained. This function has the following type:

```
add :: Bool -> Domain -> Knowledge -> Knowledge
```

That is, the function takes the following three arguments:

- the element (a `Bool`ean) with which the `Knowledge` should be updated,
- the `Domain` of which the `Experience` should be updated, and
- the `Knowledge` that should be updated.

The function adds a `Boolean` value to the `Experience` of a `Domain`, and returns the updated `Knowledge`. That is, if a student correctly applies a rule called `"modulo"`, the `Knowledge k` will be updated as follows:

```
add True (Rule "modulo") k
```

The implementation of the function `add` can be found in Appendix A.

This section will show how the knowledge is updated for the different kind of interactions. For convenience, the knowledge map that is updated is always empty within the examples.

```
updateK :: Interaction -> Knowledge -> Knowledge
updateK (CheckSolved sd e) =
  let Exercise _ s _ = e
  in  add sd (Procedure s)

updateK (NextSteps e) =
  let Exercise _ s _ = e
  in  add False (Procedure s)

updateK (WorkedOutStep e) =
  let Exercise _ _ r = e
  in  add False (Rewriting r)

updateK (Solution e) =
  let Exercise p _ _ = e
  in  add False (Efficiency p)

updateK (CheckStep d e)  =
  case d of
       Similar          -> id
       Diagnosis rd pd -> pUpdate pd . rUpdate rd
  where Exercise p s r = e

        rUpdate (True, e) =
            add e (Rewriting r)

        rUpdate (False, e) =
            add e (Rewriting "unknown")

        pUpdate Optimal =
            add True (Procedure  s) .
            add True (Efficiency p)

        pUpdate Suboptimal =
            add True  (Procedure  s) .
            add False (Efficiency p)

        pUpdate Detour =
            add False (Procedure  s)

        pUpdate Unknown =
            id
```

Figure 2.3: Implementation of the update function

Request to check whether the problem is solved

If the student requests to check whether the problem is solved, the `Interaction` is:
`CheckSolved Solved Exercise`.

The student's `Experience` is updated for the `Procedure` domain. The `Boolean` value added to this `Experience` has the same value as `Solved` (`sd`).

If the exercise is indeed solved (`sd`), the request indicates that the student has knowledge about the `Procedure`, because she apparently understands that there are no rules left to apply according to the strategy. The knowledge map will be updated as follows.

```
updateK (CheckSolved True (Exercise _ "StrategyName" _)) [] =
add True (Procedure "StrategyName") [] =
[(Procedure "StrategyName", [True])]
```

However, if the exercise is not solved, the request indicates that the student does not have knowledge about the `Procedure`, because she apparently did not know there was still a rule to apply according to the strategy. The knowledge map will be updated as follows:

```
updateK (CheckSolved False (Exercise _ "StrategyName" _)) [] =
add False (Procedure "StrategyName") [] =
[(Procedure "StrategyName", [False])]
```

Request for next steps

If the student requests possible next steps, the `Interaction` is:
`NextSteps Exercise`.

Requesting possible next steps indicates that the student does not know which rule to apply next. This implies that the student does not have knowledge about the `Procedure`. The student's `Experience` is updated negatively for the `Procedure` domain:

```
updateK (NextSteps (Exercise _ "StrategyName" _)) [] =
add False (Procedure "StrategyName") [] =
[(Procedure "StrategyName", [False])]
```

Request for worked out next step

If the student requests the next step to be worked out, the `Interaction` is `WorkedOutStep Exercise`.

This request indicates that the student does not know how to apply the rule. Therefore, the student's `Experience` is updated negatively for the `Rewrite` domain:

```
updateK (WorkedOutStep (Exercise _ _ "RuleName")) [] =
add False (Rewrite "RuleName") [] =
[(Rewrite "RuleName", [False])]
```

Request for the solution for the exercise

If the student requests the solution for the exercise, the `Interaction` is:
`Solution Exercise`.

This request indicates that the student does not know which strategy to apply. Therefore, the student's `Experience` is updated negatively for the `Efficiency` domain:

```
updateK (Solution (Exercise "ProblemName" _ _)) [] =
add False (Efficiency "ProblemName") [] =
[(Efficiency "ProblemName", [False])]
```

Request for diagnosis

If the student requests a diagnosis, the `Interaction` is:
`CheckStep Diagnosis Exercise`.

If the student did not make important changes, the `Diagnosis` will be `Similar`. This does not provide any information about the knowledge of the student, and therefore nothing changes:

```
updateK (CheckStep Similar _) [] =
id [] =
[]
```

Otherwise, if the student's input is diagnosed, information can be deduced from both the rewrite diagnosis (`rd`) and the procedure diagnosis (`pd`). This means that the variable `k`, representing the current `Knowledge`, needs to be updated twice: once according to `RewriteDiag` (`rUpdate`), and once according to `ProcedureDiag` (`pUpdate`).

```
updateK (CheckStep (Diagnosis rd pd)
                   (Exercise "ProblemName"
                             "StrategyName"
                             "RuleName")) [] =
pUpdate pd $ rUpdate rd []
```

Update according to the rewrite diagnosis

The subfunction `rUpdate` distinguishes the two cases where the rule is either recognised or not. If the rule is recognised, the `Knowledge` map is updated with the new experience. This experience is `True` if and only if the rule was applied correctly. The rule is applied correctly if and only if the two terms are equivalent. So the value added to the `Experience` list has the same `Boolean` value as `Equivalent` (`e`).

That is, if a student rewrites a term correctly according to a rule, the rule diagnosis variable `rd` will be `(True, True)` and the knowledge map will be updated as follows:

```
rUpdate (True, True) [] =
add True (Rewriting "RuleName") [] =
[(Rewriting "RuleName", [True])]
```

If a student rewrites a term incorrectly according to a buggy rule, the rule diagnosis variable `rd` will be `(True, False)` and the knowledge map will be updated as follows:

```
rUpdate (True, False) [] =
add False (Rewriting "RuleName") [] =
[(Rewriting "RuleName", [False])]
```

If the used rule is not recognised, there is insufficient information to deduce the student's knowledge about rewriting rules. However, it may be still be useful to keep track of how many times a student has (in)correctly applied some unrecognised rule. Therefore, a rule called `"unknown"` is updated just like a recognised rule would be updated.

If a student rewrites a term correctly, but the rule is not recognised, the rule diagnosis variable `rd` will be `(False, True)` and the knowledge map will be updated as follows:

```
rUpdate (False, True) [] =
add True (Rewriting "unknown") [] =
[(Rewriting "unknown", [True])]
```

If a student rewrites a term incorrectly and the buggy rule is not recognised, the rule diagnosis variable `rd` will be `(False, False)` and the knowledge map will be updated as follows:

```
rUpdate (False, False) [] =
add False (Rewriting "unknown") [] =
[(Rewriting "unknown", [False])]
```

Update according to the procedure diagnosis

The subfunction `pUpdate` distinguishes four cases, namely the cases where the `procedureDiag` (`pd`) is `Optimal`, `Suboptimal`, `Detour` or `Unknown`.

The procedure diagnosis variable `pd` will be `Optimal` if the student applies a rule that is, according to the domain reasoner, the optimal step. In this case, the student has knowledge about both the procedure of the strategy and the relative efficiency of each suitable strategy. Therefore, the student's `Experience` is updated positively for both the `Procedure` domain and the `Efficiency` domain:

```
pUpdate Optimal [] =
add True (Procedure  "StrategyName") $
add True (Efficiency "ProblemName" ) [] =
[(Efficiency "ProblemName",  [True]),
 (Procedure  "StrategyName", [True])]
```

The procedure diagnosis variable `pd` will be `Suboptimal` if the student applies a rule that is, according to the domain reasoner, a suboptimal step. In this case the student has knowledge about the procedure of the strategy. However, the student does not have knowledge about the relative efficiency of each suitable strategy, because if she had, she would have applied an optimal rule. Therefore, the student's `Experience` is updated positively for the `Procedure` domain, but negatively for the `Efficiency` domain:

```
pUpdate Suboptimal [] =
add True  (Procedure  "StrategyName") $
add False (Efficiency "ProblemName" ) [] =
[(Efficiency "ProblemName",  [False]),
 (Procedure  "StrategyName", [True])]
```

The procedure diagnosis variable `pd` will be `Detour` if the student applies a rule that is, according to the domain reasoner, not following any of the suitable strategies. This implies that the student does not have knowledge about the strategy, because if she had, she would have applied a suitable rule. Therefore, the student's `Experience` is updated negatively for the `Procedure` domain:

```
pUpdate Detour [] =
add False (Procedure  "StrategyName") [] =
[(Procedure  "StrategyName", [False])]
```

The procedure diagnosis variable `pd` will be `Unknown` if the student applies an unrecognised rule. Unfortunately, nothing can be said about the student's knowledge concerning the student's procedure knowledge or efficiency knowledge. Therefore, the knowledge is not updated:

```
pUpdate Unknown [] =
id [] =
[]
```

2.3 Applications of Automatic Assessment

Now that it is possible keep track of a student's knowledge, this knowledge can be used by the ITS. Two possible uses for this knowledge map are task selection and grading. Both of these can be based on the chance that a student will give a correct answer to an exercise from a certain domain.

The chance that a student will answer a certain exercise correctly can be obtained using Bayesian statistics. This method is based on the item response theory, which uses a number of variables characterising the knowledge of a student to calculate the chance of a correct answer [9]. In the presented student model, these variables are the experience lists of the knowledge domains. Note that this method does not necessarily use the student's experiences within just one domain. This method can base the chance on all domains. This makes it possible to, for example, predict the chance that a student will correctly apply a strategy, based on the student's knowledge about the rules used by this strategy.

Assessment is critical to the selection of a suitable next learning task [8]. The ITS can base this selection on the knowledge map within the student model to select a suitable exercise for the student. It could, for example, prefer to present an exercise from an exercise class of which the student model has the least information [17]. That is, the ITS will prefer to present an exercise where the student has a chance of nearly .5 to give a correct answer. Once the model contains enough information, exercise could be chosen based on the chance that a student will answer an exercise from a certain domain incorrectly.

Grading the student's knowledge of a certain domain may also be based on the student's chance that the next experience will be positive. For example, if this chance is .8 for a certain domain, the student may receive an 8 out of 10 grade.

Implementation of these applications should be done within the Tutoring Module of the ITS, and requires more research on task selection and grading based on a list of experiences.

Chapter 3

Conclusion and Future Work

This thesis answered the question: how can automatic assessment be implemented in ITSs? It described how diagnoses from the domain reasoner can contribute to automatic assessment. New diagnosis types were proposed to improve compatibility with this automatic assessment. This thesis discussed how the knowledge in a student model is represented and updated according to the diagnoses.

Further research may build upon this thesis by actually implementing automatic assessment within an ITS based on the proposed structure. For the best result, the diagnoses provided by the domain reasoner need to be adjusted accordingly.

Other research may determine what the best knowledge representation is. In this thesis, knowledge about a certain domain is represented as a chronological list of experiences with a binary domain. However, this is a very simple representation. It can be argued that the binary domain is too small, or that the time difference between experiences should be taken into account as well.

A last suggestion for further research is the implementation of the uses of automatic assessment. The applications described in this thesis are automatic grading and task selection. These two tasks should be performed by the tutoring module of an ITS.

Appendix A

Implementation of the Student Model

```
module StudentModel where
import Data.Map
import Data.Maybe (fromMaybe)

data Exercise = Exercise Problem Strategy Rule

type Problem  = String
type Strategy = String
type Rule     = String

data Interaction =
    CheckStep   Diagnosis Exercise
  | CheckSolved Solved    Exercise
  | NextSteps             Exercise
  | WorkedOutStep         Exercise
  | Solution              Exercise

type Solved = Bool

data Diagnosis =
    Diagnosis RewriteDiag ProcedureDiag
  | Similar

type RewriteDiag =
    (Recognised, Equivalent)

type Recognised = Bool
type Equivalent = Bool

data ProcedureDiag =
    Optimal
  | Suboptimal
  | Detour
  | Unknown

type Knowledge = Map Domain Experience
```

```
data Domain =
    Efficiency Problem
  | Procedure  Strategy
  | Rewriting  Rule
  deriving (Eq, Ord)

type Experience = [Bool]

updateK :: Interaction -> Knowledge -> Knowledge
updateK (CheckSolved sd e) =
  let Exercise _ s _ = e
  in  add sd (Procedure s)

updateK (NextSteps e) =
  let Exercise _ s _ = e
  in  add False (Procedure s)

updateK (WorkedOutStep e) =
  let Exercise _ _ r = e
  in  add False (Rewriting r)

updateK (Solution e) =
  let Exercise p _ _ = e
  in  add False (Efficiency p)

updateK (CheckStep d e)  =
  case d of
       Similar          -> id
       Diagnosis rd pd -> pUpdate pd . rUpdate rd
  where Exercise p s r = e

        rUpdate (True, e) =
            add e (Rewriting r)

        rUpdate (False, e) =
            add e (Rewriting "unknown")

        pUpdate Optimal =
            add True (Procedure  s) .
            add True (Efficiency p)

        pUpdate Suboptimal =
            add True  (Procedure  s) .
            add False (Efficiency p)

        pUpdate Detour =
            add False (Procedure  s)

        pUpdate Unknown =
            id

add :: Ord k => a -> k -> Map k [a] -> Map k [a]
add b = alter (Just . (b:) . fromMaybe [])
```

Bibliography

[1] O C Agbonifo. An online model for assessing students' understanding with stepwise solving of calculus questions. *Nigerian Journal of Technology*, 33(1):86–96, 2014.

[2] Michael Gage, Arnold Pizer, and Vicki Roth. WeBWorK: Generating, delivering, and checking math homework via the internet. In *ICTM2 international congress for teaching of mathematics at the undergraduate level*, 2002.

[3] Bastiaan Heeren and Johan Jeuring. Feedback services for stepwise exercises. Software Development Concerns in the e-Learning Domain, 2014.

[4] Bastiaan Heeren, Johan Jeuring, Arthur Van Leeuwen, and Alex Gerdes. Specifying strategies for exercises. In *Intelligent Computer Mathematics*, pages 430–445. Springer, 2008.

[5] Ann Johnson. Computer-assisted learning. *Technology in Education: Looking Toward 2020*, page 139, 2013.

[6] Simon L Peyton Jones. *Haskell 98 language and libraries: the revised report*. Cambridge University Press, 2003.

[7] Ray Kurzweil, Martin L Schneider, and Martin L Schneider. *The age of intelligent machines*, volume 579. MIT press Cambridge, 1990.

[8] Jeroen JG van Merriënboer, Dominique Sluijsmans, Gemma Corbalan, Slava Kalyuga, Fred Paas, and Colin Tattersall. Performance assessment and learning task selection in environments for complex learning. *Advances in learning and instruction*, pages 201–220, 2006.

[9] Robert J Mislevy, Russell G Almond, Duanli Yan, and Linda S Steinberg. Bayes nets in educational assessment: Where the numbers come from. In *Proceedings of the fifteenth conference on uncertainty in artificial intelligence*, pages 437–446. Morgan Kaufmann Publishers Inc., 1999.

[10] Hyacinth S Nwana. Intelligent tutoring systems: an overview. *Artificial Intelligence Review*, 4(4):251–277, 1990.

[11] Martha C Polson and J Jeffrey Richardson. *Foundations of intelligent tutoring systems*. Psychology Press, 2013.

[12] Antti Rasila, Linda Havola, Helle Majander, and Jarmo Malinen. Automatic assessment in engineering mathematics: evaluation of the impact. In *ReflekTori 2010 Symposium of Engineering Education*, pages 37–45, 2010.

[13] Vicki Roth, Volodymyr Ivanchenko, and Nicholas Record. Evaluating student response to webwork, a web-based homework delivery and grading system. *Computers & Education*, 50(4):1462–1482, 2008.

[14] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 3. Prentice hall Englewood Cliffs, 2010.

[15] Jon R Star and Bethany Rittle-Johnson. Flexibility in problem solving: The case of equation solving. *Learning and Instruction*, 18(6):565–579, 2008.

[16] Robert J Sternberg. *Handbook of intelligence.* Cambridge University Press, 2000.

[17] Michael Villano. Probabilistic student models: Bayesian belief networks and knowledge space theory. In *Intelligent Tutoring Systems*, pages 491–498. Springer, 1992.