A Flat Approach to Syntax-Checking, Shuffling, and Correcting Multiple-Choice Tests

Martin Eik Korsgaard Rasmussen
20112818

Master's Thesis, Computer Science
August 2016
Adviser: Olivier Danvy

Abstract

Ten years ago, Frandsen and Schwartzbach proposed axioms that should be satisfied by the scoring function of a multiple-choice test. They presented a scoring function that satisfies these axioms, and implemented this scoring function in a Java prototype to process multiple-choice tests expressed as a LaTeX document embedded in XML.

The goal of this MSc thesis is to study multiple-choice tests, build a concrete understanding of Frandsen and Schwartzbach's axioms, and implement a simpler prototype where multiple-choice tests are expressed as a flat, embedding-free LaTeX document that can be syntax-checked. This document is then split into parts, these parts are shuffled, and the shuffled parts are turned into many multiple-choice tests. We also designed another flat format for the candidate answers, to correct the multiple-choice tests. The result of the correction can then be used to grade the candidates.

Resumé

For ti år siden forslog Frandsen og Schwartzbach aksiomer, som burde tilfredsstilles af scoringsfunktioner af en multiple-choice test. De præsenterede en scoringsfunktion der tilfredsstiller disse aksiomer, og implementerede denne scoringsfunktion i en Java prototype til at behandle multiple-choice tests udtrykt som et LaTeX dokument integreret i XML.

Målet med dette speciale er at studerer multiple-choice tests, bygge en konkret forståelse af Frandsen og Schwartzbach's aksiomer og implementerer en simplere prototype. I denne prototype er multiple-choice tests udtrykt i et flad, integreringsfrit LaTeX dokument, som kan blive syntax-tjekket. Dette dokument bliver så splittet op i dele, delene bliver blandet og de blandede dele bliver omdannet til mange multiple-choice tests. Vi har ydermere designet et andet fladt format for besvarelser, til at rette multiple-choice testene. Resultatet af rettelsen kan så blive anvendt til at give testtagerne karakterer.

Contents

Chapter 0

Notation

"Anyone can speak Troll" . . .
"All you have to do is point and grunt."

-Fred Weasley (J.K. Rowling)
Harry Potter and the Goblet of Fire

Definition 1 (Candidate)**.** *Person taking a test, most often some kind of student.*

Definition 2 (Stem)**.** *What is being asked about in a multiple-choice question.*

Definition 3 (Option(s))**.** *The possible answers for a multiple-choice question, both correct and incorrect.*

Definition 4 (Key)**.** *The correct answer to a multiple-choice question*

Definition 5 (Distractor(s))**.** *The options which are not the key. Also known as the wrong or incorrect answers.*

Chapter 1

Introduction

In our lives we gather knowledge of things big and small and skills to do many a thing, and sometimes this knowledge and these skills need to be tested. This testing can be done in a number of ways. In this work we will focus on one of these test forms: multiple-choice tests. While this kind of testing may seem trivial, it is not. Many things need to be taken into account for a test. Among these things are guessing of the students, partial knowledge, and weights of questions. In this work, we first take a look at what has previously been proposed in the area and then propose suggestions for expansions.

As part of the work, we implemented a system for syntax-checking, shuffling, and correcting these multiple-choice tests. The need for these actions is first discussed. The system take a structured and well-defined input, which then is transformed and used to generate multiple new tests. The difference between the tests is to make it more difficult for candidates to cheat on a test, by spying on their fellow candidates in the room. In this thesis, the input is defined, the additions are presented, and the system is explained.

RoadMap: In Chapter 2, we take a look at what knowledge and skills are, why testing is needed, and different forms of testing. Chapter 3 talks about security risks in conjunction with testing. A look into work related to this thesis and areas that are touched upon can be found in Chapter 4. In Chapter 5 we take a look at how to score multiple-choice tests and make additions to current theories. Information about the system that has been made can be found in Chapter 6. Chapter 7 contains the conclusion of this work.

Chapter 2

Testing

2.1 Introduction

Our life is a learning process: we learn to survive, we learn to live healthily (more or less), we learn to tie our shoelaces, we learn to read, etc. In this chapter, we look at what knowledge and skill are – Section 2.2, the reasons behind testing – Section 2.3, and different ways to test – Section 2.4.

2.2 Knowledge and skills

There are two different things we can learn: skill and knowledge. Let us first look at what skill and knowledge are.

- According to Merriam Webster, skill is *"the ability to do something that comes from training, experience, or practice"*.
- According to Merriam Webster, knowledge is *"information, understanding, or skill that you get from experience or education"*.

Skill is often considered when talking about craftsmen, though this is not the only area, often artists are described as being skillful. As seen from the definition of knowledge, there exist some overlaps between skill and knowledge.

Through this chapter, skill will be looked as practical ability and knowledge as theoretical information. Taking some wood and from it creating something is a skill and the information of how to multiply matrices is knowledge.

Why do we need this distinction, when talking about testing? Because not everything can be tested in the same way. There is a difference if we want to test if somebody knows how a guitar works theoretically or if they know how to play the guitar.

Now let us sketch a brief history of humanity.

- **The stone age** At that time, humans didn't have much theoretical knowledge, we had skills. We needed to hunt, gather things, heal ourselves and such things, all of which are skills. There were two ways of teaching, trial and error and master/apprentice. Old skills were passed on from generation to generation by the old generation showing the new generation how to do the things they knew. New skills or improvements of older skills were discovered by trial and error.

- **Around 400 BC** At that point in history, humans had started gather theoretical information, knowledge. At this point in history, Pythagoras lived, and many people today have heard about him and his theorem $a^2+b^2=c^2$. While mathematics like Pythagoras theorem have practical application, the theorem in itself is purely theoretical: it is an abstraction of many skilled observations.

- **The 1950s** The world was on the upswing from the two world wars and the depressions seen earlier in the century. So what did this mean for knowledge and skills? Of course at this point of time, humans had invented many machines, electronics, and such based on knowledge. Knowledge was ever expanding and increasing, leading to new inventions like the washing machine, programming languages, rock and roll, the 911 call, and many others. In spite of this growing focus on knowledge, it was a time where skill was still widely demanded.

- **The 1960s** In this time many people did not get a lot of education, some even stopping after the 7th school year. These people went out and became workers, in factories, on construction sites, and the like. Some would of course stay in such positions their whole life, but some would work their way up the "ladder". Some would go on and take long educations, instead of becoming workers. This was a time where there were mathematicians, physicists, chemists, (slowly growing field of) computer scientists, and many other fields of study. Though many of these academics would stay in the pure academics fields forever, some would go out and work for large companies.

Let us take a look at our society today. Today we put a lot of weight on knowledge and skills, more then ever before in fact. While it is still possible to work one's way up the ladder, we do not always hire based on experience, instead we hire based on education. This means that people need higher levels of education, though often also a bit of experience. There are no longer many job opportunities for people without an education, the only jobs left are things like standing behind an counter, garbage collector, and taking up a paper-route. Even to serve a

cup of coffee one needs to be educated as a barista. So through the years of human existence we can observe that we slowly shift to higher and higher levels of education, thus knowledge and skills. Knowledge seems to be the main focus of this shift. Though skills have not been forgotten, but have instead been raised to higher levels as well. Who knows how it will change in the future?

2.3 Why we test

"Testing shows the presence, not the absence of bugs"

-Edsger W. Dijkstra
Software Engineering Techniques

Why do we test ourselves and others? We do it, in many settings, be it martial-arts where we get a new belt, getting a drivers license, or passing an exam at the university. Some of tests have a great influence on our lives and the lives of others. If you are in a bus or a taxi you do not want your driver to just be anybody from the street, you want them to have at least a driver's license. It is quite comforting that your doctor has spent years learning his/her field, and passed a lot of tests to show they have the needed skills for treating you. So tests are used for surveying the knowledge and skills of people, to see if they are at a suitable level.

We are tested all through our lives, even in our first moments of life. The moment we are born, doctors tests us to see if our heart, lungs, and such function well. When your father picks you up for the first time, as a newborn, he looks at you and try to see himself in you, so you try to mimic his face and expression to pass this test.

Over the decades, we have seen a growing focus on tests in education and work, which stems from the ever growing skill level required in our society. The requirement for the increase of skills and knowledge is a direct consequence of the more specialised jobs that people hold today. Specialised jobs require specialised knowledge and skills. In history we have polymaths like Leonardo da Vinci and Galileo, who at their times were masters of many fields, but it is hard to follow in their footsteps today. The challenge is that today we humans have gained a much deeper knowledge of every field, mathematics, engineering, psychology, and so on. This expansion of knowledge makes it almost impossible for one person to have in depth knowledge of multiple fields. This impossibility leads humans to become masters of a single field, or even a more specific sub-field. Learning these more and more specific skills or knowledge requires a greater amount of tests, as mistakes often become increasingly expensive.

2.4 How we test

"Well, tests ain't fair.
Those that study have an unfair advantage.
It's always been that way."

-Aiken (Allan Dare Pearce)
Paris in April

How do we test the skills and knowledge of ourselves or others? There are of course different ways. In a pure test of skills, the candidate can simply show his/her skills to the one(s) grading him/her. Though this is often not enough, as skills are often accompanied by knowledge.

To test the knowledge of a person can be a non-trivial thing to do, as knowledge comes on many levels. These levels of knowledge has been defined in Bloom's taxonomy [19]. According to Bloom's taxonomy, there exists 6 levels of knowledge. The levels are known as remembering, understanding, applying, analysing, evaluating, and creating. So before one starts to test the knowledge of someone, one needs to understand which level should be tested. It would not help testing remembering level knowledge, if you in reality wanted to test creating level, or vice versa.

Let us take a look at three ways to test knowledge: oral, written and multiple-choice tests. These are all based on giving the candidate some form of question and then having them answer it. There is of course advantages and disadvantages to each type of test.

Before I go on I would like to note that I base the parts of this text, when talking about test in education on the Danish education system. Here every grade deciding test requires some kind of censor. This censor is a neutral third party, either from the institution itself or another institution of a similar level of education. In the case of oral tests the censor will be there listening in, in the written and multiple-choice tests the censor will double check the answers. The censor is a way to guarantee that the candidate will get the correct grade/score for their test. It should be noted that with some automated tests a censor does not have as much work. In theses tests the censor still needs to check the distribution of grades. By doing automated tests a great amount of trust must be put into the system, and it therefore needs to be thoroughly tested. Now let us take a look at the test forms mentioned earlier.

2.4.1 Oral tests

"Orals are an antiquated useless tradition meant to make professors feel superior."

-Temperance Brennan (Emily Deschanel)
Bones, Season 9, Episode 16

Oral testing is a widely used way of testing, it can even be combined with a show of skill. This combination can be done as the testers can ask the candidate questions, while they show of their skills. We have our whole lives been subjected to oral testing, though many tests are not considered such. There is little doubt that most parents pick up their toddlers and try to make them say "Mama" or "Papa", this is one of the many tests we are subjegated to before we even know what a test is. Every time we talk, it is a small test, where we try to show what we have learned. If it is to show our parents that we have learned a new word, or telling our employer how the work on something is going.

This kind of testing can also help showing more of the candidates knowledge. As the testers can ask questions in a direction, where they feel the candidate still has knowledge they have not shown. There is however one issue with this kind of test. Candidates can be nervous of oral tests, thus they forget things they normally know and can be afraid to show knowledge they are not completely sure of.

Even with this issue oral testing is preferred by many testers, as they give good way to measure the candidates knowledge and confidence in their knowledge. A problem for testers can be the time oral tests can take. Let us imagine an oral test, where each candidate are set to have 30 minutes of testing, and let us say there are 25 candidates. That would mean 12.5 hours of testing, and that is without breaks and delays. So it is not hard to imagine that it would take at least two days for one set of testers to get through all the candidates. For an oral test with many candidates it might take a week or more to hold all the tests. So the time needed would quickly grow with the number of candidates.

Testers can also get tired of listening to the same few topics for a number of hours. This tiredness can make the testers unfocused or sleepy, but if this is an advantage or disadvantage for the candidate is hard to know. The only real fault one could argue for, in this kind of testing, is that some candidates could have an unfair advantage. This advantage would be having more confidence in oneself. This confidence is an advantage, as it will help a candidate seem more knowledgeable. It would be hard to remove this possible advantage, as people simply are that different.

2.4.2 Written tests

"This is risk."

-Unnamed student
Philosophy essay about "What is Risk?"

Another common form of testing is written tests. This form of testing is often used when candidates need to show a lot of knowledge and the ability to discuss it, or when they need to show that they can use things like mathematical formulae on examples. A discussion could be done as an oral test, but often the candidate would need a good amount of time to think about the things to discuss. A question like "Look at the above graphs, then summarise the information by selecting and reporting the main features, and make comparisons where relevant." would be hard to do in a 20 min oral exam. A question like that would however be more suitable as part of a longer written test.

Traditionally this kind of test has been used to show knowledge rather than skills, except when the skill is applying knowledge. This comes from the fact that this kind of testing is often longer, normally taking several hours. With this longer time, it is possible for candidates to use knowledge they already have, read new information, and then combine these. So candidates would have the time to use mathematical formulae on values, answer questions that require analysis of given information, or think about how an algorithm would run on some given data.

The time it takes to designing a written test depends on many things like the level of what is being tested, the length of the test, the experience of the author, possible guiding answers and more. The time can be anywhere from a few hours to multiple days. Having candidates take the test will not take any time for the educator, as he/she would not need to be there, just some kind of guards that make sure the candidates are not communicating.

The time it takes to grade a written test is often quite long, as one needs to read all the individual answers, think about the reasoning behind them, and then find out how close they are to the correct answers. All in all, this will require a lot of time, if the educator uses 1 hour per test, it would take 25 hours to grade the tests of 25 candidates, but others can correct the tests too, like a teacher's assistant. The grading would also be subject to human variation in judgement and performance over time [5].

2.4.3 Multiple-choice tests

"You make choices and you live with them."

-Phil Connors (Bill Murray)
Groundhog day

Multiple-choice tests is the last common testing method we look at. This test method is also used to show knowledge, as it is finding the key (correct answer) for a question. Though instead of writing the answer like written tests, the candidate must choose the key from a number of options. Many different kinds of question can be asked during a multiple-choice test, it just changes the time needed to answer the questions. A simple question with a few options might take seconds to answer, a question were the stem is a page of text might take a few minutes to answer, and a question were you have to analyse a lot of data to answer might take even longer.

Writing a multiple-choice test will take a long time, as the writer needs to come up with the stem, key, and distractors of each question. How long time is quite hard to estimate, as it depends on the number of questions, distractors, and which level in Bloom's taxonomy the question belongs on. In [5] Brown argues that an author can only write 3-4 multiple-choice questions a day, but this does not take into account experience, the level of the questions, nor the number of distrators per question. Let us estimate it takes four days to make a multiple-choice test, for an experienced author. Just like the written test, the educator does not need to be present during the test. Grading a test will just take a few minutes, let us say 2 minutes per test, which for 25 candidates will mean a grading time of 50 minutes. However the grading time will of course increase if there are many questions. Some places have even automated the grading of the tests [5]. This however can introduce a problem, for a number of reasons:

- The test should be answered using pencil, so it is easy to change which option(s) that is marked.
- The candidates must clearly mark which option(s) they think are correct.
- The candidates must understand that there will parts of the test paper, which can not be written on, as they will be used for identification of the test.

While these reasons might seem trivial, they are not. Based on the experiences of some trials held when trying to use an automated system in the Department of Computer Science at Aarhus University, it is almost impossible to get all candidates to understand these few basic premises. For an example of how beautifully bad a test can be answered look at Figure 2.1. This example is

sadly not unique, in the June 2016 exam of the Programming Languages course at Aarhus University we saw at least 10 similar cases. The reason for not using a pencil is quite strange, some candidates think that someone might maliciously change what options they have marked. The best ways to get candidates to listen to this might be to either punish them or do the more humane thing and give them an extra 10 minutes after the end of the test to transform pencil marks into permanent marks. This option of giving the candidates time to change from pencil to pen was done in the June 2016 exam of Programming Languages, but still some candidates didn't seem to listen.

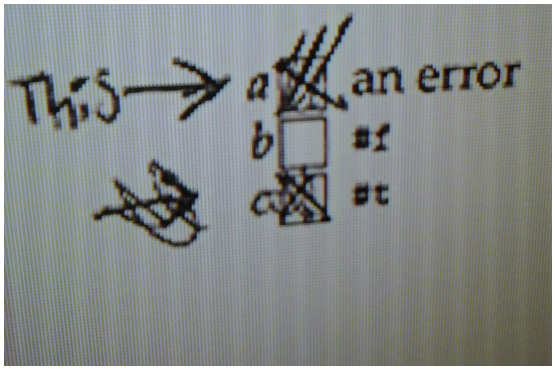

Figure 2.1: Options of a multiple-choice question, that have been marked badly. From the August 2015 exam of the Programming Languages course at Aarhus University.

2.5 Summary and conclusion

In this chapter we looked at what knowledge and skill is – Section 2.2. In this section we looked at definitions of the terms and discussed how they will be used through this work. That knowledge is theoretical and skills are more practical, there were also arguments for that the two should be tested in different ways.

We have also looked at the reasons behind testing – Section 2.3. We here looked at why we test oneself and others, the reasons this testing matters were discussed.

Lastly we look at different ways to test – Section 2.4. We looked at oral, written, and multiple-choice tests, we also looked at where these kind of tests would be applicable. From what we have seen we can conclude that multiple-choice test have their use, even though they have some faults, though most of these can be avoided by careful construction of the test.

Chapter 3

Security risks

3.1 Introduction

"When Students cheat on exams
it's because our School System values grades more than Students value learning."

-Neil deGrasse Tyson
Twitter

In exams there are potential ways for attackers (malicious candidates) to try to cheat, or in some way try to make a test easier for themselves. Multiple-choice test will be the kind of test that is focused on here. This chapter takes a look into the possible weaknesses in tests or the process of making them, that I have come up with or heard of. In Section 3.2, we look possible problems when writing a test. Section 3.3 focuses on problems when preparing the test. The focus of Section 3.4 is problems during test taking.

Some will call the content of this chapter a negative view and decry it. I call it seeing the possibilities and warning against them.

3.2 Writing the test

"Be careful that what you write does not offend
anybody or cause problems within the company.
The safest approach is to remove all useful information"

-Scott Adams
The Joy of Work: Dilbert's Guide to Finding Happiness
at the Expense of Your Co-Workers

During the writing of a test the author most be careful, or he/she might leave hints for the attackers. One way for an attacker to get a hint is if the stem of a

question ends with “an”, “a”, or similar words. This kind of fault is a grammatical hint. As an example, if the stem ends in “an” the attacker will know that the key will start with a vocal sound. This problem of course relies on the author of the exam being grammatical correct. The second hint is looking at the length of the options. The problem here comes from that some authors find it necessary to make the key much longer then the distractors. From this textual hint some students has coined the term “too long to be false.” [20]. We will look a bit more on the length of options in Section 3.4. A third thing that can hint towards the key is repetition of words. If an option has many words that are repeated from the stem it is highly likely that there is a correlation between them. This correlation will then tell an attacker that a given option has a high likelihood of being the correct answer. If there are multiple options with a high correlation to the stem it just helps the attacker narrowing down the options to chose from.

3.3 Preparing the test

“Be prepared!
And be clean in word and deed.”

-Tom Lehrer
Be Prepared

The problems we will look in this section focuses on printing and storing the tests. When looking at these preparation steps there some concerns.

- The first concern being network security. If the printing is done on a network computer it is not infeasible that an attacker can catch the packages, and thus get access to the test.
- Another possible concern is the printer itself, as it might store the document in its memory, when processing it. If an attacker gained access to this memory he/she would again gain the questions of the test.
- The location of the printer can also cause concern. If it is in an open location, without observation, an attacker could simply sneak in and steal a copy of the test, or take pictures of it.
- Storage of tests can have the same weakness as the location of the printer.

By using on of these weaknesses the attacker can have gained the test questions.

Now we have looked at these ways an attacker could see the exam before time. Why is this knowledge bad? Simple, the attacker can then prepare for the questions and learn all the keys by heart.

3.4 Taking the test

For the attacker that has not managed to cheat before the test, the possibility to cheat under the test exist, though it may require some preparation. The most of the ways candidates think of to cheat are during the test. Let us look at some possibilities.

- The attacker can look at the test of another candidate and try to glance the key from here, though this would require the attacker to believe the other candidate to be knowledgeable.
- As mentioned in Section 3.2, there can be problems with the length of the options of a question. Here it should be noted that differing lengths of options can help the attacker see which option another candidate has chosen. The reason for these differences being bad is simple, as it will be easy for an attacker to see if a specific option is chosen, if it is much shorter or longer then the other options.
- The attacker could also try to sneak in notes of possible formulae, knowledge fragments, and such. Many ways have been suggested among students to sneak in these notes, the label of a water bottle, the paper around a rubber, on the arms, inside of underwear, and many other strange and creative places. This form of cheating would actually require studying and hard work from the attacker.
  - It should be noted that a recent technology has changed this kind of cheating, the smartwatch. The smartwatch being a small computer allows for the possibility of have text on the screen, which could be *illegal* notes.
- If there was a group of attackers they could try to communicate, and thus share knowledge. This communication would depend on a reliable communication channel and trusting the knowledge of the others. The communication could be over a web service - if they have access to a network, using notes - if they are good enough to send those around without notice, Morse code - by making noise, or some equivalent way of sending data between the attackers.

3.5 Summary and conclusion

We have looked at ways to cheat in multiple-choice tests. We have seen the weaknesses in the different steps surrounding the tests. Defeating the weaknesses in test writing is mostly a question about good writing and experiences. Some guidelines and advices for doing this have been suggested [20] [23].

The weaknesses in the preparation step can be overcome by simply having a separate room for printing and storing tests. This room must be locked, and the attackers should not be able to gain access in anyway. The documents to be printed should the be put on an USB pen and printed in this special room.

Cheating under the test requires many different defences. Blocking the communication would require blocking the communication channels, though how to do this depends on the channel. Avoiding people sneaking in notes can be challenging, as they can be well hidden.

We will in Chapter 4 look at how we can make it harder to cheat by looking at the tests of others, by doing shuffles of the parts of the test, so the testpaper of each candidate is permuted or shuffled.

Chapter 4

Related work

4.1 Introduction

In this chapter we take a look at the literature about different subjects that have a relation to this thesis. In Section 4.2 we look at multiple-choice questions, Section 4.3 at markup languages, Section 4.4 at LaTeX, Section 4.5 at document generation, and in Section 4.6 at shuffling.

4.2 Multiple-choice tests

"... we must choose between what is easy and what is right."

-Albus Dumbledore (J. K. Rowling)
Harry Potter and the Goblet of Fire

Multiple-choice examinations are subjected to much discussion, some educators argue that they are a bad choice for assessing students [4]. They were once viewed as a tool of the lazy educators, but many authors have since argued against this [23] [21]. In these works, they argue that multiple-choice test can be useful, but it requires carefulness when designing the questions. There have been proposed a number of general suggestions on how to avoid common mistakes when making multiple-choice questions [23] [20].

Improvements of multiple-choice questions have been suggested in a number of different ways. Negative marking tries to discourage the candidates from guessing, this being done by give some kind of negative score, whenever the candidate has chosen one of the distractors as their answer. Allowing candidates to show partial knowledge by allowing them to mark multiple options as their answer [6].

The work of Frandsen and Schwartzbach [11] combines the notions of negative marking and partial knowledge. They present five axioms that they state must

be self-evident properties of any fair scoring strategies. The scoring strategy presented by Frandsen and Schwartzbach forms the basis of the scoring strategy used in this work. There will therefore be an in-depth look at their axioms and strategy in Section 5.2. In Section 5.3 a slightly revised version of the scoring strategy is presented and the reasons for the changes are discussed. Another point that is worth looking at is the possibility of having more than one key. Some authors argue that multiple keys will increase the risk of errors and impression [14]. The assumption of only one key is also an important part of Frandsen and Schwartzbach's work, so their scoring strategy is not made to handle multiple keys. This is one of things that will be discussed in Section 5.3.

The impact of the work of Frandsen and Schwartzbach can however be discussed. Some papers find it to complicated to adopt [22] [17] [18], while others find that it does not work with what they want [24]. There have been done work to extend the work of Frandsen and Schwartzbach as in [16], it was however never shown if the extension satisfies the axioms. At the department of Computer Science at Aarhus University the work of Frandsen and Schwartzbach has had impact, as their work is basis for most multiple-choice test there.

4.3 Markup language

" Q:How do you annoy a web developer"

XKCD 1144

Markup languages are used for annotating documents. They can consist of tags, special characters and more. This marking or tagging indicates the logical structure and layout of the document, both for displaying on computers or printing. There exists a number of different types of markup languages

- *Punctuational* - This is the most well-known kind of markup language, and one that is used every time we write something. This type of markup language includes things like ",","." , and " ".
- *Presentational* - Is most commonly found in WYSIWYG (What You See Is What You Get) editors. When working in this kind of editor we press space we get a " " and if we press tab we get " ".
- *Procedural* - Is a series of commands, that indicate how text should be formatted.
- *Descriptive* - Text is marked with tags or tokens, which tell the format of the text. An example hereof could be and to start and end text paragraphs.

- *Referential* - This markup is used to refer to external resources, and is then replaced by these resources during compiling/processing. This could be "\qquad" used to represent " ".

- *Meta-markup* - Is a markup language that talks about markup languages, or alternatively a markup language that defines other markup languages, these are most often used to extend other markup languages with new parts.

For a more in depth study of markup languages the interested reader can refer to [7].

The most common types of markup languages are of course punctuational and presentational. Punctuational is naturally used in many places, as anything else would mean that text would be completely unreadable, an example of this could be "h e l l o w o r l d". Presentational markup languages, as the name suggests, are used to define how the text is presented, for an example we skip a line or do an indent, when we start a new paragraph. The next popular form of markup languages are descriptive markup languages, such as LaTeX, HTML, or XML. This kind of markup languages has gained popularity, as they make it easy to adjust entire documents, by just making change to how the tags/tokens are processed, as many know from working with HTML and CSS.

It should be noted that some markup languages like LaTeX are not only descriptive, but a combination of the different types of markup languages. In LaTeX we still put ".", we use tags "\begin{...}" and "\end{...}", we make references to other documents "\include{...}", we can write "\qquad" to get a tab in our text, and we can define new parts of the markup language "\newcommand{...}". The LaTeX markup language is therefore descriptive, meta-markup, punctuational, and referential.

4.4 LaTeX

One of the hardest things about using LaTeX is deciding how to pronounce it.

-Leslie Lamport
[15]

As mentioned in Section 4.3, LaTeX is a markup language for typesetting documents, but more correctly the syntax of LaTeX is the markup language.

Since the start of the LaTeX system in 1985 it has grown into one of the most used tools in academia and science. Some would even call it the *de facto* language or *lingua franca* for scientific documents. LaTeX is used for generating many types of documents like books, articles, slideshows, and theses (like the

present one). LaTeX was designed to free you from formatting concerns, allowing you to concentrate on writing [15].

While most common text editors are WYSIWYG (What You See Is What You Get), LaTeX is based on the idea of WYSIWYM (What You See Is What You Mean). By now LaTeX and TeX, the basis for LaTeX—made by Donald Knuth [13][1], is a well known and used tool. This means many integrated development environments (IDEs) have been developed for working with LaTeX. These IDEs are greatly varied, some need to be installed on your computer, some work online, some use MiKTeX, some use teTeX, and so on. The IDEs are not strictly necessary, as LaTeX can be written in text editors like Emacs or vi, but the lack of built-in compilers and graphical interfaces pulls some users away from these options. This along with many other boons are offered by LaTeX, but this is not why it is interesting in this setting.

LaTeX is easy to extend with new functionality and it is well structured. It is therefore easy to to have programs construct and destruct LaTeX files. It will also be easy for users to change the look of the generated documents, by simple changes/additions to the meta-markup of LaTeX. This will mean that the end users can work in something that they are quite used to and only add a simple package to write the new input files. Having the users work in something that they are used to work with will help their productivity as they do no need to learn a new tool, it will also mean that they are used to the error messages of the tool.

4.5 Document generation

Through the story of humankind we have documented many things, from hunting to history, from families to beliefs. This has of course been done in different ways: cave-drawings, stone tablets, scrolls, books, and now electronic documents. In electronic documents we can get the possibility of writing markup and then generate the documents, see Section 4.3 and Section 4.4. This generation is done using compilers. Let us here borrow the words of Olivier Danvy "*A compiler is a program for translating another program from one language (the 'source language') to another (the 'target language').*" [8]. I generalise this to: A compiler is a program for translating a structured input to a structured output. This is clear from looking at systems like LaTeX, org-mode [2] or Sphinx [3], where you give a structured input to a compiler, and then receive a structured output. In both cases you write your input in a clearly defined structured format, which then can be compiled. Sphinx even have compile flags to choose the language of the output, if it should be HTML, LaTeX, XML, or other possible structured output languages.

4.6 Shuffling

"You've got to know when to hold 'em.
Know when to fold 'em."

-Kenny Rogers
The Gambler

The problem of shuffling a number of items is a common problem, we encounter it every time we play a game of cards. An efficient approach for doing this is the Fisher-Yates shuffle, which was defined in [10], but later popularised by Knuth in [12]. The popular version of this shuffling was made by Durstenfeld in [9]. The algorithm can be stated as

```
1 Fisher-Yates_shuffle(Array a)
2     for i := (a.length-1) downto 1 do
3         j ← random int such that 0 ≤ j ≤ i
4         Swap(a[j], a[i])
5     return a
```

Figure 4.1: Fisher-Yates shuffle

This shuffling lays under the class of Monte Carlo methods, as it relies on randomness. As always with random numbers one could start writing a big discussion about what randomness means and how to achieve it, but that is out of the scope of this thesis.

One part of randomness should be noted though. When picking random items from a set of items (arrays, list, etc.), you can do it with or without replacement. With replacement means that we select a item, but leave it in our set. Without replacement is simply the opposite, that we remove the selected item from the set. From this it is clear that when shuffling a test, we should do it without replacement, or else the candidate could get the same question multiple times. Though the candidate would either love or hate this, depending on if the candidate knows the answer, it would not make for a good test of his/her knowledge and skills.

The Fisher-Yates shuffle is a good method for shuffling a great amount of data, when you need it to be random. However this is not needed for this project, as the only constraint is that there must be a low possibility of the tests of two people sitting next to each other being identical. Here being identical means that the parts of the test appears in the same order. Therefore only a few different tests are needed, and a less powerful shuffling function can be used. Such a shuffling function can be seen in Figure 4.2.

```
swap(list l)
    if seed != 0
        m = seed mod 4
        if m == 0
            return swapEven(l)
        else if m == 1
            return swapOdd(l)
        else if m == 2
            return reverse(l)
        else if m == 3
            return l
    else
        return l
```

Figure 4.2: Function that decides how to shuffle based on a global seed

The swap function in Figure 4.2 is quite simple and can easily be extended with more cases if more possible generated tests are wanted. It is also possible to just increment the seed every time the function is called, so each part of the test that is shuffled will be done so in a different way from the previous part. In the version shown this increment is not done, which means that this function is an involution, which can easily be observed from the fact that all the functions it applies to its input are also involutions. To help the reader see this, I have included the pseudo code of the swapEven function in Figure 4.3. The swapOdd function is identical except the list of odd indexed elements are swapped instead.

```
swapEven(list l):
    list even, odd = ()
    for i := 0 to (l.length - 1)
        if i mod 2 == 0
            even = cons(list[i]), even)
        else
            odd.add(list[i])
    return merge(even, odd)
```

Figure 4.3: Function that takes all even indexed elements from a list and reverses their order, then merges them back with the remaining elements and returns the new list.

4.7 Summary and conclusion

This chapter has reviewed relevant related work. We took a general look at multiple-choice tests, but will expand on this later in Chapter 5. We looked at markup languages, what they are and how they are used. We turned to the markup language LaTeX and how we might take advantage of some of its features. We looked at document generation as the next subject and how one input can be used to generate many things. The last part of this chapter was a look into shuffling, we here looked at a good general algorithm to shuffle data. With the realisation that this algorithm was a lot more powerful then what will be needed, we looked at another algorithm. This second algorithm is not nearly as powerful as the original, but it does enough work for it to be a good solution for what we need in this work.

Chapter 5

Scoring multiple-choice tests

5.1 Introduction

In this chapter we take a look at how to score multiple-choice tests. We review an earlier work on the subject – Section 5.2. This earlier work was presented in Section 4.2. We will then see the expansions done to this earlier work as part of this work – Section 5.3. In the same section we also take a look at a possible future expansion that can be made.

5.2 A singular choice for multiple choice

Your power to choose can never be taken from you.
It can be neglected and it can be ignored.
But if used, it can make all the difference.

- Steve Goodier

The work by Frandsen and Schwartzbach [11] presents five axioms and a scoring strategy for multiple-choice tests, like what was mentioned in Section 4.2. The scoring strategy is based on the five axioms, but before looking at these axioms we need a bit of notation. Let k_i be the number of possible answers for a question, a_i be the number of checked answers, and c_i be a boolean denominating if the correct answer is checked. For each question a student will then receive a scoring $S(k_i, a_i, c_i)$. The five axioms are

Axiom 5.1. *Zero:* $S(k, k, TRUE) = S(k, 0, FALSE) = 0$.

Axiom 5.2. *Monotonicity:* $S(k, a, FALSE) < S(k, a, TRUE)$.

Axiom 5.3. *Anti-monotonicity: if* $a \leq b$ *then* $S(k, b, c) \leq S(k, a, c)$.

Axiom 5.4. *Invariance:* $S(k_1, a_1, TRUE) + S(k_2, a_2, TRUE) = S(k_1k_2, a_1a_2, TRUE)$.

Axiom 5.5. *Zero-Sum:* $\frac{\binom{k-1}{a-1}S(k,a,TRUE)+\binom{k-1}{a}S(k,a,FALSE)}{\binom{k}{a}} = 0$.

Let us take a look at what the axioms say.

- Axiom 5.1 states that if for one question the answer consists of no options or all the possible options, the score of the answer is zero.
- Axiom 5.2 states that if the candidate has chosen a options out of the k possible for a question and none of them is the key, then the score is lower then if he/she had chosen the same number of options and the key was among them.
- Axiom 5.3 states that if one candidate chooses a options for a question, another candidates chooses b options for the same question, so that $a \leq b$, and they both have either chosen the key or only distractors (the boolean value in their scoring is the same), then the candidate who has chosen a options gets the highest score for this question.
- Axiom 5.4 states how the score of two questions, where the key has been chosen, can be put together as either addition or a combined calculation.
- Axiom 5.5 states in simple terms how guessing will make the score go towards zero. So this is the part that will discourage candidates from randomly guessing at what options to mark.

In their work Frandsen and Schwartzbach then present a scoring strategy, which they claim is the only one that satisfies these axioms. This scoring strategy is used as the basis for marking tests later in this work, though we add some things to it, as discussed in Section 5.3.

Definition 6 (Scoring strategy)**.**

$$S_{axioms}(k, a, c) \triangleq \begin{cases} 0 & \text{if } a = 0 \vee a = k \\ \log(\frac{k}{a}) & \text{if } a > 0 \wedge c = TRUE \\ -\frac{a}{k-a}\log(\frac{k}{a}) & \text{if } a > 0 \wedge c = FALSE \end{cases}$$

5.2.1 Implementation

Frandsen and Schwartzbach have also made an implementation of their scoring strategy. This implementation was done in Java with XML files as the input. In the XML files there is embedded LaTeX. This embedding can of course introduce some complications, e.g., if one will write some LaTeX like `$a < b$`, the < will be viewed as part of a tag by XML.

5.3 Additions to multiple-choice tests

While Frandsen and Schwartzbach take many parts of multiple-choice tests into account in their work [11], there are still some possible features missing. In this section, we look into two extensions of their work. Both these extensions or additions give the users more features to work with, which can improve the tests they write. We will also take a look at a possible future feature of the system.

5.3.1 Multiple keys

One feature that some test authors might want is the possibility to have questions with multiple keys. The reason for this being that some options might be equivalent, and therefore it could be nice to test if the candidates realise this. This feature has therefore been added in this work. In Figure 5.1, an example of such a question is shown. In this question, there are three keys, all marked with an X. Most candidates can figure out that option K must be a key, if they know what an identity function is. Options P and O are not as trivial, to mark these the candidate must read the stem carefully and realise that it is the identity function for numbers and not the general identity function, they must also know what the + and * functions can be given one argument in Scheme.

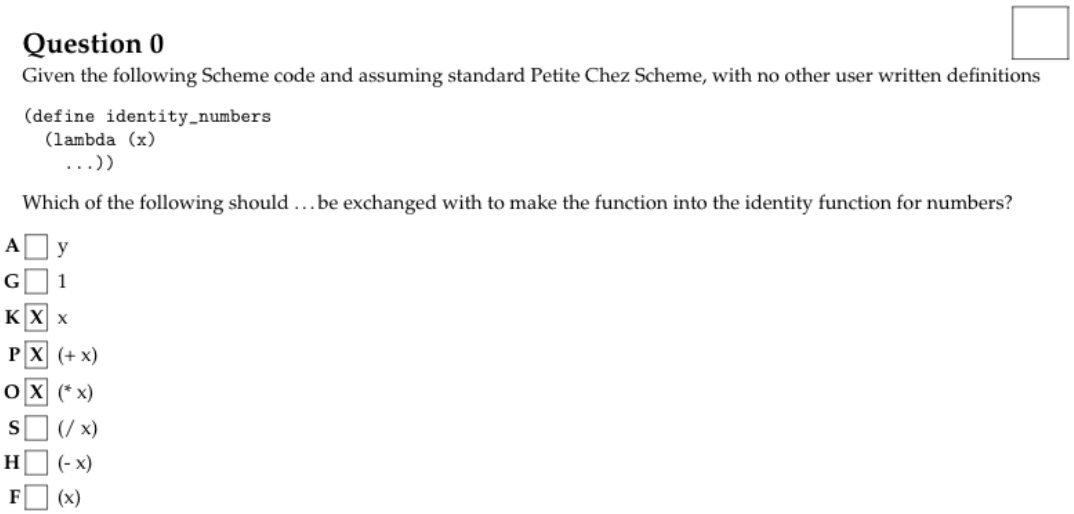

Figure 5.1: Question with multiple keys.

It should be noted that based on how the test is written, this way to ask the question might be bad. If the test consist of questions with one key except one question, it should be noted clearly that one question has multiple correct answers in the start of the test, and it should also be noted in the stem that there are multiple keys for this question. However if it is common in the test that the questions have multiple keys, it should just be noted in the start of the test,

and not necessarily written in the stem. Though a test with many keys for each question is rarely seen, as many consider it bad form [23].

Now let us look at how we can include the multiple keys in the scoring strategy of Frandsen and Schwartzbach. To do this, we change the notation a bit. Remember that in their work c_i was a boolean determining if the key is checked or not. It was *TRUE* if the key was checked and *FALSE* otherwise. Now we let c_i be a boolean denominating if one of the keys are checked. It will be *TRUE* if one or more of the keys have been checked and *FALSE* otherwise. We let n be the total number of keys for the question and m be the number of keys the candidate has marked. The only change to the scoring strategy is then in the case of $a > 0 \wedge c = 1$. We will keep the $\mathsf{log}(\frac{k}{a})$ from the work of Frandsen and Schwartzbach, but divide it by n and then multiply with m. The modified scoring strategy can be seen in Definition 7.

Definition 7 (Scoring strategy with multiple keys)**.**

$$S_{axioms,multipleKeys}(k,a,n,m) \triangleq \begin{cases} 0 & \text{if } a = 0 \vee a = k \\ \mathsf{log}(\frac{k}{a})/n \cdot m & \text{if } a > 0 \wedge m \geq 1 \\ -\frac{a}{k-a}\mathsf{log}(\frac{k}{a}) & \text{if } a > 0 \wedge m = 0 \end{cases}$$

Since no other modifications has been made to the scoring strategy it should hold to the axioms.

5.3.2 Question weights

It is natural that when we ask questions that some might be more important then others. This happens in both oral and written tests, so why should this not be a part of multiple-choice tests too?

During my years at the university I have seen multiple-choice tests, where the author tried to simulate a weight. It was done by having the same question repeated a number of times. While it is a good attempt to simulate a weight it can have the side effect of confusing the candidate, as they stop and think "Did I not just answer this question?". Luckily this can be avoided, like it was in the ones I have seen, by clearly writing that there will be multiple copies and mark which questions are identical. It also opens up for the possibility of candidates trying to guard against getting to many negative marks. Here the guard being to in the first occurrence of the question marking one option and in the next occurrence marking another option. A guard like that could have some influence especially if it is a binary question, i.e., it has two possible options.

The option to weighting questions has been added in this work. How to write the weight in the input is discussed in Section 6.2. Again we need to introduce a bit more notation to our scoring strategy. We let w be the weight of a question, the default value is 1, and we assume that w is always a natural number.

We know that weights of questions can be simulated as having multiple repetitions of one question. We also know that using the invariance axiom that questions can be combined into one. So a question with weight w means combining a question with itself w times. So therefore the score with a weight can be calculated as:

$$S_{weights}(w, k, a, c) = \sum_{1}^{w} S_{axioms}(k, a, c)$$

5.4 Summary and conclusion

In this chapter, we looked at the the work of Frandsen and Schwartzbach [11] about scoring multiple choice tests in a fair way. We looked at some additions to their work, which have been made as part of this work. With the additions we expanded on the scoring strategy made of Frandsen and Schwartzbach, to make the strategy that will be used later on to score multiple-choice tests, Definition 7. We also looked at possible future addition to scoring multiple-choice tests.

Chapter 6

Implementation

6.1 Introduction

This chapter is about the implementation of a system that shuffles multiple-choice tests as mentioned in Section 4.6. The system also takes care of syntax-checking, by using LaTeX as its input language.

The part of the system that syntax-checks is described in Section 6.2.

The part of the system that shuffles the test to generate new documents is described in Section 6.3.

The last part of the system is the part that taxes the generated shuffled tests and the answers for these from the candidates, which are then used to give the candidate a score. This last part is described in Section 6.4.

The last part of this chapter, Section 6.5 takes a look at how to use and run the different parts of the system.

A side note should be that this implementation is flat, unlike that of Frandsen and Schwartzbach. Here flat means that the input is simply a LaTeX file, so there is no need for embedding, like in their implementation.

6.2 Input

As mentioned in Section 4.4, LaTeX is a nice structured input, which the end users will be mostly comfortable with already. Using LaTeX also removes the requirement of writing a syntax checker, as this is done when compiling LaTeX, when it knows how the environments and commands are defined. LaTeX already has the option of extending the language with new environments and commands, which can then be put into packages. Such an package can easily be included in the work of an user. For this application, a package would need ways to markup:

- The start and end of a test, so the code that generates shuffled tests knows what part of the document it should work on.

- Groups of questions, with
  - A required heading.
  - An optional text giving information needed for all the questions in the group.
- Questions, which will consist of
  - A stem.
  - A list of options.
  - An optional weight, so the writer can mark questions that are more essential.
- Lists of options, which have
  - At least one distractor.
  - At least one key.

Even if we in Section 4.2 looked at how others disagree with multiple keys for one question, and that the scoring strategy of Frandsen and Schwartzbach [11] does not take them into consideration, I allow them. While it is an option to have more then one key I implore any user to only use it sparingly and clearly state in the test when they are present. The choice is taken with the hope the authors of tests will only use them when strictly necessary and when it will truly test something. I base my choice on the fact that I am not an experienced test author and therefore do not know if it will be useful in any case. With this I have changed the scoring strategy a bit to reflect it, which I have discussed in Chapter 5.

To make the markup required for writing multiple-choice tests, a number of new LaTeX environments and commands are needed. These have been gathered in an package called multiple-choice.sty. The structure of the commands and environments are like described earlier and can be seen in Figure 6.1. The package also makes some checks, like that things are inside the correct surrounding environment and that things that need to be unique are. If any check is violated a LaTeX error is thrown, so the author can see something is wrong. With the input being in LaTeX, the test authors are also be used to the style of errors thrown in LaTeX, as mentioned in Section 4.4.

```
<test>          ::= \begin{test}
                        <group>+
                    \end{test}

<group>         ::= \begin{questionGroup}
                            <groupHeading>
                            <groupText>?
                            <question>+
                     \end{questionGroup}

<groupHeading> ::= \groupHeading{ <string> }

<groupText>     ::= \begin{groupText}
                        <string>
                    \end{groupText}

<question>      ::= \begin{question}
                            <weight>?
                            <stem>
                            <options>
                    \end{question}

<weight>        ::= \weight{ <int> }

<stem>          ::= \stem{ <string> }
                  | \begin{stem}
                        <string>
                    \end{stem}

<options>       ::= \begin{options}
                        (<key> | <option>)*
                        (<key> <option>)
                        (<key> | <option>)*
                    \end{options}
                  | \begin{options}
                        (<key> | <option>)*
                        (<option> <key>)
                        (<key> | <option>)*
                    \end{options}

<key>           ::= \key{ <string> }
                  | \begin{key}
                       <string>
                    \end{key}

<option>        ::= \option{ <string> }
                  | \begin{option}
                        <string>
                    \end{option}
```

Figure 6.1: BNF of LaTeX additions.

6.3 Generating tests

In this section we take a look at how the program that takes in a test, destructs it, shuffles the parts, and then generates a new document, works.

I have for the programming language chosen Python, which is a multi-paradigm language, therefore giving the possibility to program following the object-oriented paradigm. Even though it is used functionally in this work, it is not used purely functionally. This comes from the fact that some side-effects are needed, like writing to files.

It would have been possible to implement everything in TEX/LATEX, as they offer full Turing completeness. It would however have been a lot harder, they lack some of the nice features of other higher-level programming languages.

It is also possible to use packages like the Python Package, to embed scripts into LATEX. This could have been a feasible solution, it is often nice to split things into logical parts.

6.3.1 Destructuring the test file

The first major part of generating tests is destructuring the test itself. Here the work is done on a part of the full test document, the part being `\begin{test}...\end{test}`. Everything in this environment is then taken and destructured into a list of elements. The first part that the test is split into is a list of groups.

Each group is then a list consisting of a start element, a heading, an optional grouptext, a list of questions, and an end element.

Each question then consist of a start element, a stem, an optional weight, a list of options, and an end element.

Each list of options then consist of a start element, at least one key, at least one distractor, and an end element.

If you compare this to the BNF in Figure 6.1, you see that it is almost identical, the BNF is just a representation of the test in LATEX and the lists just described is the tests represented using lists. The reason for destructuring the test part of the document is so it is easier to do the needed transformations, like shuffling. It should also be noted that during the destruction of the options environment, each element is given a deterministic randomly chosen letter, which we look at in Section 6.3.2.

6.3.2 Letters for options

As mentioned in Section 6.3.1, every option for the questions are assigned a deterministic randomly chosen letter, to be used as an identifier. These letters see their use when tests are being corrected. The uses of the letters are discussed in Section 6.4.1.

The reason for doing random letters, and not just go a, b, c,. . . , stems from a story from my adviser, Olivier. The story goes, he was running a correction on a number of exams, when suddenly there was a error, which was that for some question in some test an option was chosen, that was not a possibility. The error was made by the correctors who had written all the answer into a file. It was a human fault. This got Olivier thinking, how often do faults like this happen, and how often does a similar fault happen. This similar fault being that something is written that the candidate did not mark, but it is still a valid option for the question.

Let us look at an example, Figure 6.2. The question we see here has letters that are simply the successor of each other starting from a. If you are a human and are correcting let us say 15 tests with 95 questions each, then you get to repeat the same letters often. It is the obvious to see that there is a chance of saying a letter that does not exists for the question. In this example f would not be a valid letter, as there exists no option with f as its letter. Though since f does not exist it is easy for a program to spot this mistake.

That was the fault that Olivier spotted by getting an error message. Now let us look at the similar fault. The candidate has marked a as his/her chosen option, but the correctors writes down the letter b. This can not be detected by the program, as b is a valid option. This is a fault that can always happen, when humans are part of the system, but we hope to minimise the chance of it happening. This is done by using the random letters. Since it is not the same letters repeated again and again, at least not that often, the persons writing down the answers has to look closer to see what letter they need to write.

It should be noted that in the correction done by my adviser and some correctors that help him, they do it in pairs, one reads the test and call out the marked options, while the other writes them down, and then vice versa. This introduces another possible reason for such faults, letters that sound alike, since most people are not trained in using the NATO phonetic alphabet. Of course time could be spend on everybody learning this, which they might or might not find useful. A more realistic solution would be to simply remove some letters. The letters m and n sound quite alike, so instead of having them both I have chosen to remove n and keep m. A similar choice for other letters that sound alike has been made.

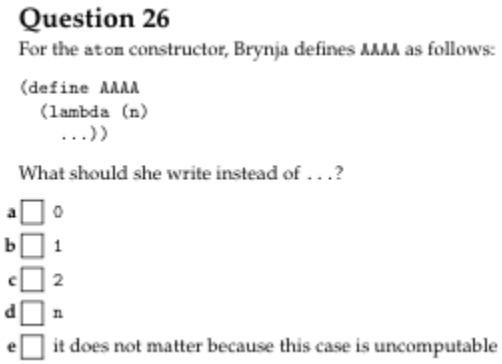

Figure 6.2: Question with succeding letters.

A last note about the letters in the system I have made should be that I have chosen to use capital letters. This comes from the simple fact that they are easier to see and often more distinct.

6.3.3 Shuffling

The shuffled method used in this work is the involution swap function presented in Figure 4.2. As this is enough for the purpose needed here. While some things might come in the same order in tests of candidates close to each other, the point is to make it harder to cheat through the whole test, instead of doing some complicated shuffling. This is easy to implement and a simple algorithm should be enough for that.

6.3.4 Generating the new files

The generation of the new parts is split into two parts. The first part is everything that is not part of the test environment. Almost everything that is in the first part is simply written to the new files without any change at all. The only thing in the first part that are change is the place where `\testNumber` is present in the input file. All instances of `\testNumber` are replaced with a four-digit number that represents the test, this number is also the seed for the randomly chosen letters and the value used by the swap function to decide what to do. The four-digit number is then written to the file instead of `\testNumber`, which is the only change that happens.

The second part is then the test itself. This part is destructed, assigned letters, and then shuffled like discussed in Section 6.3.1, Section 6.3.2, and Section 6.3.3. The second then consist of a list with many sub lists, which all have been transformed as mentioned. The list is then taken and recursively written to the file being generated.

6.3.5 Hash files

When we correct the tests we need to make sure that the files that were generated as the tests have not been changed. To do this the common approach of a hash file is used. It is a file containing the tests files hash values, which can then be checked during correction. The hash algorithm chosen is sha256, as it is well known. The files are spilt into chunks of 65536 bytes, so we do not work on to large amounts of data. Each chunk is then hashed and written as a line in a .sha256 file.

6.3.6 Future improvements

One possibility to make the generation of shuffled tests faster would be to make the process multi-threaded. This would of course speed the generation significantly up. The multi-threading has not been done yet, as the system seems to be quite fast already.

As an example of the run time, let us generate 1000 tests. It will be done on a macbook pro from late 2011 running OS X Yosemite version 10.10.5. The computer has the following specs: Processor 2.4 GHz Intel Core i7 and Memory 4 GB 1333 MHz DDR3. The computer is at the same time used normally, which in this case means playing Youtube videos, a pdf viewer, Chrome with some tabs open, and a few background processes running. Under these circumstances the generation took 820.91 seconds or a bit over 13.5 minutes, which is an acceptable run time for something like this. Compared to the 5 hours it take to print the tests, this is quite low.

The speed-up might be need if the tests being worked on are big with hundreds or thousands of questions or there need to be generated thousands of tests, but these are not something that is expected to happen often.

6.4 Scoring tests

6.4.1 Scoring individual tests

The scoring strategy from Section 5.3.2 is used to score the answered tests. The answers of each candidate is then put into a file using the format, uniqueID testNumber answers. This is expressed in Figure 6.3. Let us look at an example

```
20112818 0001 0A1HJ3Q4P5OB6-...
20112819 0002 0A1FA3QT4-5-6V...
```

The uniqueID is expected to be a number, it could be a studentID or something along those lines. The four-digit number is the number present on the tests, which represents the seed used to generate the specific version of the test. The last part is the questions and the answers the candidate gave, for the first entry

0 means Question 0 and A means that Option A has been chosen. For Question 1 the first candidate has chosen two options H and J. For Question 6 the first candidate has not chosen any options at all, thus the -.

The correction program then takes the entries and destruture them, for each one it then finds the previously generated test files. It then for each question in a test files finds the valid letters and keys. The options chosen by the candidate are then matched against the valid letters, and if any does not match an error is raised. The options chosen by the candidate are then matched against the keys, and the score is calculated using the scoring strategy. The answers are also recorded and used for the total score document.

A file named scores.txt is generated, which contains data of the following format

```
Student number: 20112818 scored: 91.16%
Student number: 20112819 scored: 65.32%
```

So based on the answers file and the previously generated tests, a file containing the percentage score of each candidate is generated. These scores can then be used to decide if the candidates passed, what grade they get or other things along those lines.

```
<candidateAnswer>     ::= <uniqueID> <testNumber> <answers>

<uniqueID>            ::= <int>

<testNumber>          ::= <int>

<answers>             ::= <answer>+

<answer>              ::= <int><string>
```

Figure 6.3: BNF answer files.

6.4.2 Total score document

The total score document contains some statistics about the overall answers of the test. An example of the statistics can be seen in Figure 6.4.

For each question there is a fraction, which tells how many candidates out of the total has answered, in the example it is $\frac{123}{157}$.

For each option there is also a fraction, the first number being the number of times this options has been chosen and the second the total number of times the question has been answered.

As one can see in the example the total number of times the question has been answered and the number of candidates that has answered the question is

not the same. This comes from the fact that the candidates can mark multiple options.

The key(s) of the question is also marked with a X. On the right side by the key(s) there are the percentage of how many of the total number of candidates has marked the key.

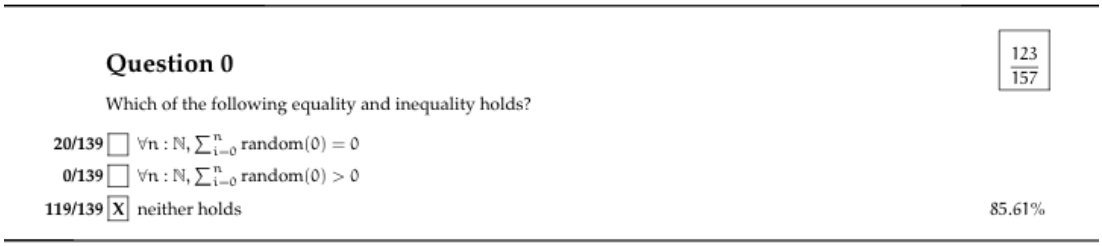

Figure 6.4: Example of a question in the total score file

6.5 Using the system

6.5.1 Getting the system

The system has been made available online at
www.github.com/LordFXS/multiple-choice.

6.5.2 Syntax-checking tests

The syntax-checking of the input, discussed in Section 6.2, can be checked in two ways.

The first way is using the power of LaTeX, meaning that it can be done using the package that defined the input. The author of the test can simply try to compile it using a LaTeX compiler, which will then throw errors if anything is wrong.

The second way is having a program that runs through the document and checks that the BNF of Figure 6.1 is being followed. There is however some problems with this. To syntax-check the document in this way one could run into the problem of having to check all of LaTeX. This problem is non-trivial as LaTeXis Turing complete. Instead it is more feasible to only check the part of the document that holds the test. And instead of checking if things are in the right order one will then check that they are not in the wrong order. It is easier to check if something is there, then checking if it is not there.

6.5.3 Generating tests

The program for generating shuffled tests from an input file is fairly simple to run. Let us look at an example

```
python generate.py full_exam/june_2014.tex 1000 test
```

As can be seen in the example the program takes three parameters.

1. This parameter is the test that the author has written, i.e., a tex file containing a test that holds to the BNF of Figure 6.1. So in the above example june_2014.tex is the test that has been written.

2. This parameter is the number of shuffled versions of the test to generate. So in the above example 1000 tests will be generated.

3. The last parameter is the name of the generated files. So in the above example the files will be called test. This means that test1 to test1000 will be generated.

The only parameter that is required is the first. The second parameter has the default value 1. The third parameter has the default value test.

After the program has been run it will have created a new folder in the location it was called. This new folder, generated_tests, contains two sub folders and a few files. Most of these parts are only for use during the correction of the tests. The only part the user needs to worry about is the sub folder called pdf. This sub folder contains all the pdfs that are the generated shuffled tests.

The user can then simply print all the pdfs that were generated and he/she will have the tests ready to go.

6.5.4 Correcting tests

The program for correcting test requires five parameters. An example call can be seen here under.

```
python correctTests.py full_exam/
                       answers.txt
                       scoresFile.txt
                       generated_tests/
                       test
```

Let us look at what the parameters are.

1. This parameter is the only one that is need and has no default value. It is the path to were the original input file is stored. I.e., it is the location of the file used for generating all the shuffled files.

2. The second parameter is the name of the file with the answers of the candidates. This file is described in Section 6.4.1. The default value is answers.txt.

3. The third parameter is the name of the file that will be generated and which will contain the scores of all candidates. The default value is scoresFile.txt.

4. The fourth parameter is the location of the TeX folder, which is a sub folder of generated_tests. The default value is generated_tests.

5. The fifth parameter is what the generated tests are called. The default value is test.

When the correction program is done it will have generated a file which is named by the third parameter. This file contains scores for the candidates, like described in Section 6.4.1. There will also have been made the file containing the total scoring described in Section 6.4.2, has been made inside the folder generated_tests.

6.6 Summary and conclusion

In this chapter we have looked at the system made for syntax-checking, shuffling, correcting multiple-choice tests. We looked at the a input defined in LaTeX that represents a multiple-choice test. The transformation of this input into multiple tests were the next thing we looked at. The numbers chosen as "names" for the options has been discussed, and we looked at arguments for doing random letters from a subset of the alphabet, without replacement. Another part of this chapter was the total score document, which is generated when the tests are corrected. As a part of this document we save what information was written, so the user can use them for statistics. We also looked at how the programs can be run.

Chapter 7

Conclusion

The focus of this thesis has been multiple-choice tests. We wanted to know a number of things about this test form:

- How feasible is it?
- How does this test form compare to others?
- What is needed and used to make the test form good and fair?
- Does there exist any security risks by using multiple-choice tests?
- Is there anything that should be added to multiple-choice tests to improve their usability?
- Is it possible to create a system that can be used to write multiple-choice tests?
- Is it possible for the system to guard against security risks?
- Is it possible in the system to give the author access to some of the things that gives higher usability?
- Can the system have a flat input? This means that the input should not consist of two languages, where one is embedded into the other, instead a single language would be nice.

To see if multiple-choice tests are a feasible testing form, we had to first look at: what skill and knowledge are, why do we test them, and how do we do it. During this we also did some comparisons of testing forms and talked about their advantages and disadvantages. We here found that multiple-choice test should be as good as the other test forms studied. We have looked at what others have written about multiple-choice tests and what should be done to score them fairly and correctly. During all of this we found that it is a feasible testing form, if

partial knowledge and negative marking are used. We found how good a test is depends on how well it is written.

Security risks were found when writing a test, printing a test, storing a test, and when candidates take the test. We found suggestions to handle most of these risks. Some of these suggestions can be part of the system that we talk about, while others are matters of protocol.

To score multiple-choice tests we found the work of Frandsen and Schwartzbach [11] to be a good way to go. There can however still be added features to make the tests even better, like weights of questions. We then looked at possible ways to change the scoring strategy, so it can handle some of these additions.

The work of Frandsen and Schwartzbach also showed that a system, where in multiple-choice tests can be written, can be made. In the system they have made the input is not flat. We then looked at a way to make a similar system: how the parts are made, how to use them, and what it needs to run. We have seen how a shuffling can be added in the system, so tests are not identical, which will help mitigating some of the security risks. We have also seen that a flat input can be used, as we use LaTeX as our input for generating tests, and specially formatted text file to generate scores.

7.1 Future work

During this thesis a few things to work with in the future have been suggested. Let us look at some more:

- Formalising the scoring strategy of [11] and prove it satisfy the axioms stated in the same paper.
- See if a simpler equivalent scoring strategy can be found, then formalised and proven to hold to the axioms.
- Formalising the updated scoring strategies of Chapter 5 and prove they satisfy the axioms of [11].

7.2 End notes

To end this thesis let us take a look at the title once again: *"A Flat Approach to Syntax-Checking, Shuffling, and Correcting Multiple-Choice Tests"*. We have found that it is feasible to make a system to realise this. We can make system that multiple-choice tests:

1. Takes a flat input.
2. Syntax-checks the input.

3. Shuffles the input to generate multiple new tests.

4. Corrects the tests given then answers of the candidates that took the test. This input is also flat.

During the work of this thesis, we designed and implemented a system that does all of this. So we have fulfilled the promise of the title. This system will be made available and open-source, so anyone can download it, use it, and maybe expand on it further. It is the author's hope that the system will be used and help educators to write good tests.

Acknowledgements

"Now, now, my good man,
this is no time for making enemies"

-Voltaire (François-Marie Arouet)

There are many who have my thanks for things big and small. The first person is my adviser, Olivier Danvy: he has been a great source of inspiration and has always taken the time to discuss things with me until we found an answer. I am also grateful to Erik Ernst for serving in my thesis committee, Mathias V. Pedersen for proofreading this dissertation and for discussing its content, and Aarhus University as a whole for its educational facilities. Lastly my thanks go to my family and friends, for their patience and their moral support all through my studies up to and including the work reported here

Thank you.
Martin Eik Korsgaard Rasmussen
Aarhus, Wednesday 17th August, 2016.

Bibliography

[1] Donald E. Knuth. http://www-cs-faculty.stanford.edu/~uno/. 20

[2] Org-mode for emacs. www.orgmode.org. 20

[3] Sphinx - Python documentation generator. www.sphinx-doc.org. 20

[4] John B. Biggs. *Teaching for quality learning at university: What the student does*. McGraw-Hill Education (UK), 2011. 17

[5] Robert W. Brown. Multi-choice versus descriptive examinations. In *Frontiers in Education Conference, 2001. 31st Annual*, volume 1, pages T3A–13. IEEE, 2001. 10, 11

[6] Martin Bush. A multiple choice test that rewards partial knowledge. *Journal of Further and Higher Education*, 25(2):157–163, 2001. 17

[7] James H. Coombs, Allen H. Renear, and Steven J. DeRose. Markup systems and the future of scholarly text processing. *Communications of the ACM*, 30(11):933–947, 1987. 19

[8] Olivier Danvy. Programming languages lecture notes, week 16, 2015. http://users-cs.au.dk/danvy/dProgSprog/Lecture-notes/. 20

[9] Richard Durstenfeld. Algorithm 235: random permutation. *Communications of the ACM*, 7(7):420, 1964. 21

[10] Ronald A. Fisher, Frank Yates, et al. Statistical tables for biological, agricultural and medical research. *Statistical tables for biological, agricultural and medical research.*, (Ed. 3.), 1949. 21

[11] Gudmund S. Frandsen and Michael I. Schwartzbach. A singular choice for multiple choice. *ACM Inroads*, 38(4):34–38, 2006. 17, 25, 27, 29, 32, 44

[12] Donald E. Knuth. *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms.* Addison-Wesley, 1969. 21

[13] Donald E. Knuth and Duane Bibby. *The TEXbook*, volume 1993. Addison-Wesley Reading, Mass., 1986. 20

[14] Rosemarie K. Kolstad, L. D. Briggs, and Robert A. Kolstad. Multiple choice rules prevent the selection of wrong options in examinations. *Education*, 110(3):360, 1990. 18

[15] Leslie Lamport. *LaTeX: A document preparation system: User's guide and reference manual. Second edition.* Addison-Wesley, 1994. 19, 20

[16] Simon McCallum. Game design for computer science education. *Gjøvik University College*, 2010. 18

[17] Annie W.Y. Ng and Alan H.S. Chan. Evaluation of three multiple-choice assessment methods in a human factors engineering course. *Journal of the Chinese Institute of Industrial Engineers*, 29(7):466–476, 2012. 18

[18] Leonidas Oxouzoglou. Students's attitudes towards alternative multiple choice questions. 2013. 18

[19] Benjamin S. Bloom, Max D. Engelhart, Edward J. Furst, Walker H. Hill, and David R. Krathwohl. *Taxonomy of Educational Objectives Book 1: Cognitive Domain*. Longmans, Green and Co Ltd, 1956. 8

[20] Jim Sibley. Seven mistakes to avoid when writing multiple-choice questions, 2014. www.vetmed.wsu.edu/docs/librariesprovider16/Docs---Teaching-Academy/seven-mistakes-to-avoid-when-writing-multiple.pdf?sfvrsn=0. 14, 15, 17

[21] Des Traynor and J. Paul Gibson. Synthesis and analysis of automatic assessment methods in CS1: generating intelligent MCQs. *ACM SIGCSE Bulletin*, 37(1):495–499, 2005. 17

[22] Jon Warwick, Martin Bush, and Sylvia Jennings. Analysis and evaluation of liberal (free-choice) multiple-choice tests. *Innovation in Teaching and Learning in Information and Computer Sciences*, 9(2):1–12, 2010. 18

[23] Karyn Woodford and Peter Bancroft. Multiple choice questions not considered harmful. In *Proceedings of the 7th Australasian conference on Computing education-Volume 42*, pages 109–116. Australian Computer Society, Inc., 2005. 15, 17, 28

[24] Andriy Zapechelnyuk. An axiomatization of multiple-choice test scoring. *Economics Letters*, 132:24–27, 2015. 18