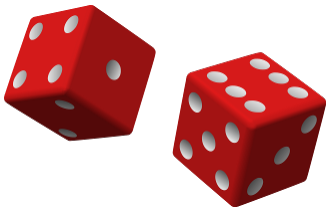
Generation of German narrative probability exercises

Masters Thesis

by

Roan Boer Rookhuiszen



January 26, 2011

Supervisors:
Dr. M. Theune
Dr. Ir. H.J.A. op den Akker
Prof. Dr. C.A.W. Glas
H. Geerlings MSc.

University of Twente
Human Media Interaction

Acknowledgement

I am very grateful for everyone who helped me during my research. First of all, I thank Mariët Theune for her supervision, our regular meetings and her encouraging feedback during those meetings. Second, I thank Rieks op den Akker for helping me by asking tough questions about probability theory and other parts of my report. I also enjoyed our sometimes rather philosophical discussions. I also thank Cees Glas and Hanneke Geerlings for their guidance and feedback during every phase of my research. I appreciated the help of Nina Zeuch by giving helpful answers to the questions I had for her.

I am also very grateful to everyone who helped me during the evaluation of my research for their time and effort. I especially thank Verena Stimberg, also for helping me many times to understand the characteristics of the German language.

Finally, I thank all my family and friends for their support, their interest in the progress of my research and the research itself. I especially thank Elisa Alvarez, also for her thorough check of the final report.

Finally, this research project would not have taken place without the funding from the Deutsche Forschungsgemeinschaft (DFG), Schwerpunktprogramm "Kompetenzmodelle zur Erfassung individueller Lernergebnisse und zur Bilanzierung von Bildungsprozessen" (SPP 1293), Project "Rule-based Item Generation of Algebra Word Problems Based upon Linear Logistic Test Models for Item Cloning and Optimal Design."

Contents

1 Introduction **1**
- 1.1 The purpose of our research . . . 1
- 1.2 Variation in exercises . . . 2
- 1.3 Requirements . . . 3
  - 1.3.1 Requirements on exercises . . . 3
  - 1.3.2 System requirements . . . 3
- 1.4 Overview of this document . . . 4

2 Previous research **7**
- 2.1 Research on Narrative Exercise Creation . . . 8
  - 2.1.1 The ModelCreator software . . . 8
  - 2.1.2 WebALT . . . 13
- 2.2 The Narrator in the Virtual Storyteller . . . 14
  - 2.2.1 The NLG processes in the Narrator . . . 15
- 2.3 Aggregation and ellipsis . . . 16
  - 2.3.1 Conclusion . . . 18
- 2.4 Resources . . . 18
  - 2.4.1 GermaNet . . . 18
  - 2.4.2 FrameNet . . . 19
  - 2.4.3 Canoo.net . . . 19
  - 2.4.4 Conclusion . . . 19
- 2.5 DEG: Demonstration Exercise Generator . . . 19
  - 2.5.1 The functionality of DEG . . . 20
  - 2.5.2 The limitations of the DEG . . . 20
- 2.6 General conclusions . . . 21
  - 2.6.1 Existing systems . . . 21
  - 2.6.2 Reuse of techniques and resources . . . 22

3 Genpex: the developed program **23**
- 3.1 Global system design . . . 23
- 3.2 Graphical User Interface . . . 25
  - 3.2.1 Status box . . . 25
  - 3.2.2 Probability problem tab . . . 26
  - 3.2.3 Exercise Text tab . . . 27
  - 3.2.4 Other features . . . 28
- 3.3 Resources . . . 29
  - 3.3.1 Context files . . . 29
  - 3.3.2 Dictionary for the inflection of nouns and verbs . . . 32

3.4 Tools . . . 32
3.4.1 Context file tester . . . 33
3.4.2 Inflection retriever . . . 33
3.5 Implementation details . . . 34
3.5.1 Java and Netbeans . . . 34
3.5.2 Jaxb: Java Architecture for XML Binding . . . 35
3.5.3 Information embedded in the source . . . 35

4 Probability problems **37**
4.1 Analysis and terminology . . . 37
4.1.1 Probability problem derived from example exercise . . . 38
4.1.2 Terminology . . . 38
4.1.3 Probability theory . . . 40
4.2 The structure, meaning and limitations of statements and questions 42
4.2.1 The context . . . 42
4.2.2 Total number of entities . . . 43
4.2.3 Single attribute value with or without a count . . . 43
4.2.4 The count for a combination of two attribute values . . . 43
4.2.5 Dependency information . . . 44
4.2.6 Questions . . . 44
4.3 Question types . . . 45
4.3.1 Restrictions . . . 45
4.3.2 Overview of all question types . . . 48
4.3.3 The naming convention of question types . . . 48
4.4 Generating Probability Problems . . . 50
4.4.1 An overview of the generation process . . . 50
4.4.2 Input: context and question types . . . 50
4.4.3 Select attribute values and make statements . . . 51
4.4.4 Generation of counts . . . 54
4.4.5 Add counts and order statements . . . 57
4.5 Calculating the answer to a question . . . 58
4.5.1 Terminology and basic calculations . . . 59
4.5.2 Calculation rules . . . 60

5 Natural Language Generation in Genpex **63**
5.1 Natural Language Generation pipeline . . . 64
5.1.1 The NLG pipeline in Genpex . . . 65
5.2 Sentence trees . . . 65
5.2.1 Analysis of example sentences . . . 65
5.2.2 Information embedded in a sentence tree . . . 67
5.2.3 Canned Text: simple representation of a sentence in a single node . . . 69
5.3 Document Planner . . . 70
5.3.1 The construction of sentences trees for statements . . . 70
5.3.2 The construction of sentences trees for questions . . . 71
5.4 Micro Planner . . . 73
5.4.1 Aggregation . . . 73
5.4.2 Adjectivication . . . 74
5.4.3 Entity substitution . . . 75
5.4.4 Change word order to marked word order . . . 76

5.4.5 Techniques in combined statements and questions 77
5.4.6 Ellipsis 77
5.4.7 Techniques used in the Micro Planner 78
5.4.8 Variation 78
5.5 Surface Realizer 79
5.5.1 Morphology 79
5.5.2 Orthography 81

6 Evaluation of Genpex **83**
6.1 Overview of the tests 83
6.2 Functionality of Genpex 84
6.2.1 An extensive test of all functionality 85
6.2.2 Global evaluation of Genpex v2 88
6.3 Correctness of mathematical exercise 89
6.4 Generated language 89
6.4.1 Grammar correctly applied 90
6.4.2 Readability of the exercises 90
6.5 Ambiguity 92
6.5.1 Ambiguity in questions with conditional probabilities . . . 92
6.5.2 Exclusive or inclusive 'or' 94

7 Conclusion **97**
7.1 Discussion of the requirements 98
7.2 Evaluation 98
7.3 Future improvements 99

8 Future work **101**
8.1 Introduce new types of exercises 101
8.2 Context and mathematical exercises 102
8.2.1 Develop a context file editor 103
8.2.2 Add and use more context specific information 103
8.3 Language 104
8.3.1 Generate referring expressions in the questions 104
8.3.2 Use synonyms and hyperonyms 104
8.3.3 Investigate the preferred (amount of) text variation . . . 105
8.3.4 Make Genpex multilingual 105
8.4 Add new import and export functionality 106

Bibliography **108**

A Word list **109**

Chapter 1

Introduction

Most people are familiar with mathematical word problems or narrative exercises. A narrative exercise is a mathematical exercise embedded in a story or text. Narrative exercises are used to test or train a student's understanding of the underlying mathematical concepts. A student is required to derive the underlying mathematical question from the story of the exercise and to calculate the correct answer to this mathematical problem.

1.1 The purpose of our research

In ongoing research at the department of Research Methodology, Measurement and Data Analysis (faculty GW, University of Twente) and the department of Statistics and Quantitative Methods for Psychology (University of Münster) there is a need for narrative exercises. The departments are jointly developing a model that will be used to support optimal problem and test construction. Therefore, they need a large collection of narrative exercises. Those exercises are used to perform field trials and test the developed models. All of these narrative exercises should be different, but the properties that define the difficulty of the exercise should be known.

During earlier research, some exercises were created by hand, one of these exercises is shown in Figure 1.1. The text of the exercise is in German, because field trials were performed with German high school students. All exercises tested the users' knowledge about statistics. If for further research more exercises have to be designed by hand, it will take a lot of time and effort. It is difficult to make many different exercises and add variation to an exercise without changing the difficulty.

In this thesis, we will discuss the development of Genpex, the Generator for Narrative Probability Exercises. This program should be able to construct exercises automatically, similar to the exercises previously designed by the researchers at the University of Münster.

Der Besitzer des Fahrradladens “Rad ab” möchte neue Kunden gewinnen und hat sich überlegt, etwas mehr in Werbung und Dekoration zu investieren. Neben allerlei saisonalen Deko-Artikeln will der Besitzer jede Woche ein Fahrrad in seinem größten Schaufenster neu ausstellen. Dabei soll jedes Mal ein zufällig ausgewähltes Fahrrad aus seinem Sortiment im Fenster stehen.
Der Besitzer hat insgesamt 400 Fahrräder im Angebot.
Davon sind 80 Räder Mountainbikes, 100 Räder Rennräder, und die restlichen Räder sind Hollandräder. 40 Räder sind silberfarben, 120 Räder schwarz und 240 Räder weiß. 240 Räder haben eine 3-Gang-Schaltung und 160 Räder eine Schaltung mit mehr als drei Gängen. 96 Räder sind weiß und haben eine 3-Gang-Schaltung. 96 Räder sind Hollandräder und haben eine Schaltung mit mehr als drei Gängen.
Fahrradtyp und Gangschaltung sind abhängig voneinander, alle anderen Merkmale sind unabhängig voneinander.
Wie groß ist die Wahrscheinlichkeit, dass ein Fahrrad ins Schaufenster gestellt wird, das nicht sowohl ein Rennrad als auch weiß ist?

Figure 1.1: The text of an exercise designed by hand, during earlier research performed at the University of Münster.

1.2 Variation in exercises

In our research, we will develop a program that is able to generate narrative probability exercises. The wording of those exercises is dynamic in the sense that it could change each run of the program.

The story of the example exercise in Figure 1.1 describes a situation in which there are 400 bikes. In this situation, every bike has a colour, a number of gears and is a mountain bike, an upright bike or a racing bike. After the description of the situation, a question asks for the probability of a bike with certain properties.

We can make a variation of this exercise by replacing all numbers. We can for example divide all numbers by 2. We get a new exercise that discusses a situation with 200 bikes, of which 40 are mountain bikes etc.

We can change the context or theme of the exercise and make another variation of the same exercise. We can for example make an exercise that describes a situation of a hotel with 400 rooms, where every room has a price, has a view and is a certain type of room. The same kind of question can be included in the text, but now it will ask for the probability of a room with certain properties. The underlying mathematical exercise for this new exercise might be equal to the underlying mathematical exercise for the exercise about bikes. If the numbers used in the exercise remain the same, the calculation and answer will be the same too.

We introduce variation in the language of the exercise, but it is important that the text of the generated exercise is unambiguous: the formulation should not leave the reader uncertain about the underlying mathematical exercise. The student should be able to correctly identify the underlying exercise. Moreover: the difficulty of an exercise should be completely known and defined. The vari-

ation in the used language should not be a factor in the difficulty of an exercise. If certain variation unintentionally changes the difficulty of the exercise, that type of variation should not be used in the creation of new exercises.

1.3 Requirements

The program that we are going to develop should create exercises similar to the exercises that were constructed by hand during the research at the University of Münster. In this section we give an overview of all requirements on the exercises that the program should be able to create. We also specify other requirements on the functionality of the program.

1.3.1 Requirements on exercises

The most important output of Genpex is the text of the exercise. As we have seen, those exercises should be similar to the exercises used in the research of Münster. However, the program that we are developing should not be restricted to the generation of only these exercises. We should make a program that is able to generate these exercises and various other exercises that are similar. There are several restrictions on the exercises that should be generated:

1. The used language in the created exercises should be grammatically correct German.
2. The text that expresses the underlying mathematical problem has to be unambiguous: if there are two variations of the same exercise, they should have the same meaning.
3. The program should be able to create at least the type of exercises that are used in the research of Münster
4. If multiple questions are embedded in one exercise, the system should ensure that a student can not use the answer of a previous question to answer other questions.
5. The numbers that are used in the statements section of the underlying mathematical problem should be different. Otherwise, it is possible that the student provides the correct answer by performing a calculation that is different than the intended one.

1.3.2 System requirements

While the text of the exercise is the most important output of the system, we will also list other system requirements. Some of the requirements concern other output of the system, others concern the usability of the system.

1. The researcher that uses the program has to be able to easily create new exercises.

2. The researcher needs to have control of the output of the system. He should be able to correct minor errors by hand. If the researcher makes changes, the system should help him whenever possible, by automatically checking the correctness of the specified information.

3. In the output of the system, information about the applied variation should be supplied. This information could be used in further research in which field trials are performed with the generated exercises. The output should also give insight in the processes that formed the exercise.

4. The system should be prepared for future extension of its functionality, by using a modular design of the system.

It is expected that the variation that is added to the language, does not influence the difficulty of an exercise. To be sure, this should be evaluated. The evaluation will not be part of our research. However, it should therefore be possible to exclude certain variation techniques in the generation process if further research shows that the difficulty of the exercise is unintentionally changed by these text variation techniques.

1.4 Overview of this document

The structure of this document follows the methodology of the research. As we have said, we are going to develop a system that is able to create narrative exercises and we have already listed the requirements on the exercises and the system requirements. In this document, we discuss the development of this system by answering the main research question of our research:

"How do we make a system that meets the requirements on the exercises and the system requirements?"

To be able to answer this research question we performed a literature study. We discuss in Chapter 2 the relevant text generation systems we found in literature. Two systems that are capable of the generation of exercises are discussed in more detail. In that chapter, we will also discuss some preliminary research, in which we developed a demonstration version of an exercise generation program.

Based on the literature and preliminary research, we have created a global system design for our program, Genpex, which we discuss in Chapter 3. We shortly discuss the purpose of the different subsystems and the input and output of the system. The resources that are used by Genpex are also discussed in this chapter, just as some tools that we have made to create those resources. We show a couple of screenshots of Genpex in this chapter and discuss some technical details about the implementation.

All subsystems are discussed more detailed in the chapters that follow. First, we discuss the probability problems in Chapter 4. A probability problem is the representation of the mathematical exercise used in Genpex. We also discuss the creation of those probability problems. In Chapter 5 we continue with the generation of the text of the exercise based on those probability problems. In this chapter, the variation that is possible between texts is discussed too.

It is important to check whether the developed program meets all requirements. We have therefore performed an evaluation which is discussed in Chapter 6. Based on this evaluation, we give a conclusion in Chapter 7 and suggestions for further work in Chapter 8.

Throughout this thesis, some terminology is used to refer to specific parts of an exercise or its underlying mathematical problem. This terminology is listed in Appendix A.

Chapter 2

Previous research

We can divide this chapter and the types of previous research in two pars. First of all, we discuss the literature study that we performed to be able to answer our main research question. We then discuss DEG, a demonstration program that we have developed during earlier research.

In this literature study we try to answer the following questions:

1. How are narrative mathematical exercises generated by other systems? And can we benefit from their research and use the same techniques?

2. How is language generation performed in other systems?

   (a) What techniques are suitable to add variation in the language used in the text of an exercise?

   (b) What resources are available that can be used in the language generation process?

To be able to answer the first question, we have researched the literature on narrative exercise generation. In Section 2.1, we show two systems that we have found and discuss their usefulness for our research. To answer the second question we continue in Section 2.2 by discussing a system developed at the University of Twente in which language is generated. In this section, we discuss some techniques that can be used to add variation to the generated language. Some German resources that are available are discussed in Section 2.4, that answers the last question.

In Section 2.5 we discuss the 'Demonstrator Exercise Generator' that we created during earlier research. In the last section we will conclude by discussing what techniques and resources we will use in the development of our exercise generator.

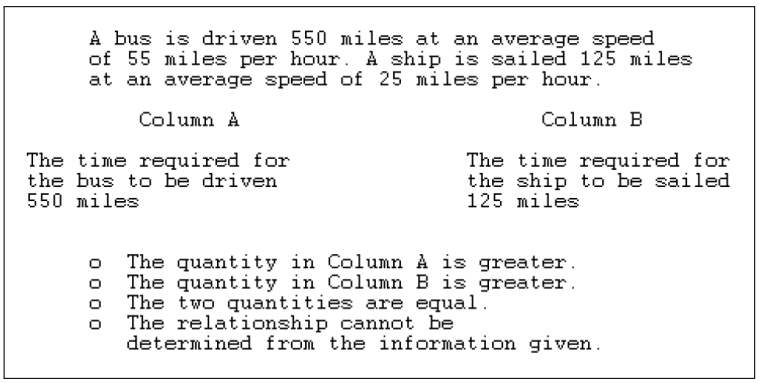


Figure 2.1: Example exercise created with the ModelCreator software. [HFD05]

2.1 Research on Narrative Exercise Creation

In this section, we will discuss two systems that are able to create mathematical exercises, expressed in a text. The domain of the exercises in both systems is different than the type of mathematical exercises that we consider: one system focuses on distance-rate-time problems, the other system gives mathematical formulas embedded in a text that explains what the student should do with that formula. While the type of exercises are different, they are both worth discussing in more detail because they try to express a mathematical exercise in natural language.

This section answers the first question, as we proposed in the introduction of this chapter. We will not only discuss the working of both systems, but we will also discuss how we can benefit from the research that has been performed.

2.1.1 The ModelCreator software

The ModelCreator software is a system that is capable of the automatic generation of English distance-rate-time problems and has been developed by Deane and Sheehan [DS03]. The mathematical exercises are expressed in the QC (quantitative comparison) format: all exercises have some general information, followed by two columns that express quantities. An example exercise that is generated with the ModelCreator software is shown in Figure 2.1. The question for the student is to compare the quantities given in both columns. The student has to answer a multiple choice question that asks for example for the column with the greatest quantity.

As Dean and Sheehan argue, the distance-rate-time domain of the exercises is well-suited for exercise generation systems. The verbal information needed to express those exercises is simple and well defined, and the underlying mathematical construction is straightforward (because only simple multiplications are necessary to be able to answer the question).

Deane and Sheehan argue that the majority of the variables used by a Natural Language Generation (NLG) system should not influence the difficulty of the

exercise. Those variables are for example the exact word choice, the used grammatical constructions and the details of phrasing. However, they noticed that the use of 'difficult' language influences the difficulty of the exercise.

It is argued by Enright et al. [EMS02] that not only wording could vary the difficulty of an exercise, but also the real world content to which a word problem is applied can influence the difficulty of an exercise. They have shown that one particular mathematical formula is more difficult when it is stated in a distance-rate-time problem, than the identical formula stated in a cost and price exercise. Similarly, an exercise about probabilities is more difficult than the same exercise rewritten in terms of percentages. Even when both the numbers used in the exercise and the required calculations to derive the answer are equal, an exercise with a question that requires the student to calculate a probability was found to be more difficult to answer compared to the exercise with percentages.

On the use of language in exercises the following conclusions are given by Enright et al.:

- You can assume that the variables within the NLG system, such as the exact word choice, do not influence the difficulty of the exercise. Exceptions are those systems that generate exercises to test the verbal encoding/decoding skills of a student.
- However, the use of difficult language or language structures can have an influence on the difficulty of the generated items

This sounds as a contrast. While it is argued by Enright et al. that variations introduced within the NLG system, should not influence the difficulty of the exercise, a different textual representation of a mathematical exercise might influence the students' capability of answering the involved questions. Therefore, if someone performs a research on the difficulty of those exercises, it is important to include the verbal contant in the analysis required for this research.

Template vs. in-depth NLG

The language generation process in NLG systems can be roughly divided in two parts: systems that use a template based approach and systems that use an in-depth NLG approach. Higgins et al.[HFD05] have made a multilingual version of ModelCreator, by adding support for the Japanese and Spanish languages, in addition to the English language. During their discussion of the existing ModelCreator, they discussed the choice for an in-depth NLG approach in favor of a template-based NLG approach.

In a template-based NLG system, there is a template-sentence available for all possible situations. This sentence contains empty slots that can be filled with one or more words. In an in-depth NLG system, the function of a word in a sentence is considered and together with the grammar of a language, the sentences are constructed. This is possible, because a language has a basic word order for the possible structures.

In contrast to in-depth NLG, templates are a simple approach: a large number of items could be generated with minimal effort. A template with empty slots has

to be specified, together with all possible variations for each slot. An example template can be: *A [VEHICLE] drives [DISTANCE] miles.* We can define that in this example, the first slot can be filled with 'car', 'bus' or 'bike', and the second slot can be filled with a number. Many combinations of vehicle and distance are possible, and therefore many different sentences can be generated.

This example shows a big disadvantage of template-based NLG systems: a simple change in the type of exercise requires the design of completely new templates. For instance, if we also want to express the speed of a vehicle in the previous example, a new template has to be designed.

There is also a so called *interslot dependency problem.* For example, it is reasonable in a distance-rate-time problem to create items that involve cars traveling at 100 kilometers per hour but not at 1000 kilometers per hour. The slots in those templates, in this case the vehicle and speed, cannot be chosen randomly. It is possible to embed such restrictions in a template based system, but it will result in many templates that each have different constraints.

Another important argument against a template-based approach is that it is not practical in a multilingual environment. For every new language that is added to the system, all templates should be translated.

The generation process in the ModelCreator software

The ModelCreator software introduces an approach that is different from these template or in-depth NLG approaches: it constructs its own item models based on some generation parameters. This item model is the formal and logical representation of the exercise, and is constructed with knowledge from a frame database to overcome the interslot dependency problem. A frame database contains information about the real world concepts of words. For example, if it is specified in the generation parameters that the speed is 60 miles per hour, due to the information in the frame database the vehicle 'bike' will not be chosen. There is a significant amount of interdependency knowledge and specialized lexical knowledge necessary in the frame database, because the possible speed is not the only property of a vehicle. The construction of the formal representation is followed by the language specific step where this representation is converted into natural language.

The generation process of ModelCreator is shown in Figure 2.2. We see that the input of ModelCreator is a list of 'generation parameters', which is converted into a 'logical representation' before it is translated into a 'finished item' (the text of the exercise).

The 'generation parameters' are more structured than the description of the empty placeholders in a template. Those 'generation parameters' are therefore more suitable to be edited in a graphical user interface, because the structure of those parameters (especially the name, such as 'distance') limits the possible values that are suitable. In the 'logic creation' process, the 'generation parameters' are converted in a logical representation of an exercise. The information is expressed as discrete propositions. In this process it is also determined what information has to be embedded in the introduction and in the two columns of

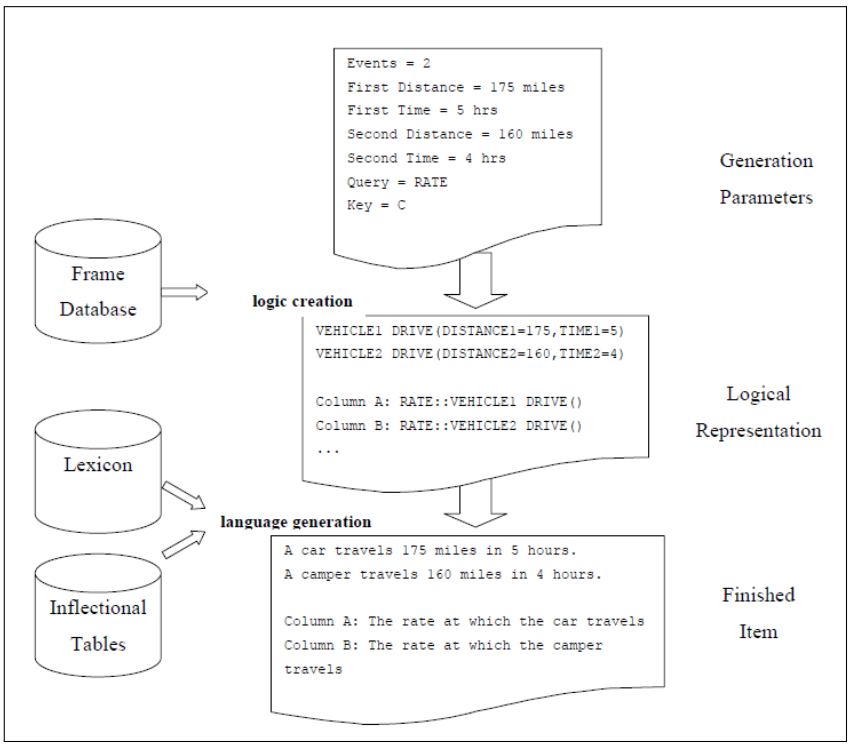


Figure 2.2: The steps in the ModelCreator generation process. [HFD05]

the exercise. The previous discussed semantic frame database is used to put the right constraints on the roles and actions of participants in the exercise. For example, if a distance and time is given, from the frame database a 'drive' action is selected. This will result in an exercise about two vehicles, in which the question requires a comparison of the speed of both vehicles. This representation is largely language-independent and therefore useful in a multilingual system.

Finally linguistic substance is added to the logical outline. A lexicon is used to look up appropriate terms for each of the used logical elements. This is similar to a template filling approach, however this filling is done in a linguistically informed way: using the knowledge from the lexicon, words are used with a correct inflection, to reflect the correct tense, case and gender.

Multilinguality

As we said, Higgins et al. [HFD05] adapted the existing ModelCreator software so that it is capable of the creation of exercises in multiple languages. Japanese and Spanish are now supported in addition to English.

Higgins et al. described the steps they took to extend the existing ModelCreator software. There is a big difference between the addition of the Spanish language, which is similar to English, compared to the Japanese language. An important

aspect of the Japanese language is the use of Japanese characters. The interface of the program should be extended in such a way that it is able to display those characters. Other differences are the word boundaries: where in English and Spanish words are separated with spaces, in Japanese words are not separated. It is therefore difficult to determine where line breaks should be used.

For our research, we focus on the differences found between English and Spanish, because we expect that these differences are similar to the differences between English and German. Higgins et al. discuss grammatical differences between English and Spanish, such as a somewhat different word order and different inflection of words. For example, in Spanish, nouns have inherent gender. Also, adjectives agree in gender and number with the noun that they modify.

The languages are also different in many other aspects, such as the expression of a time. In English one would say 'Three o'clock' but it is translated into Spanish as 'las 3:00'. There are also cultural differences: in English, it is common to express a distance in miles, where in Spanish a distance is expressed in kilometers. It is necessary to perform an exhaustive review of the generated sentences to ensure that no aspect of the grammar or cultural differences of a language is overlooked.

Conclusion

The type of exercises that can be created with ModelCreator is different than the type of exercises that we are going to create. However, we are also creating mathematical exercises and can learn from the choices that were made in the development process of ModelCreator.

First of all, the generation process is split into two steps: in the first step, a 'logical representation' of the exercise is generated, based on some 'generation parameters'. In the second step, this representation is converted into text. It is suggested that those parameters can be edited more easily via a graphical user interface than the 'logical representation' of the exercise, because those parameters are more abstract and more general.

There are more advantages of a 'logical representation' that are neither mentioned by Enright et al. [EMS02] nor Higgins et al. [HFD05]. This representation can be used to manually check the correctness of the generated language of the exercise. The information that someone derives from the text, should be equal to the information embedded in the logical representation. Another advantage is that the representation is well structured and therefore it is relatively easy to create a program that parses this representation. It will be possible to develop a system that is able to calculate an answer to the questions embedded in the exercise.

The 'logical representation' is converted to text in the second step of the generation process. This language generation process can be compared to a template based NLG approach: every structure in the 'logical representation' is expressed in natural language. Sentences are created in a language informed way, where inflections are correctly applied using information from a dictionary. We cannot speak of a real in depth NLG system, because the system still fills in the empty

Internal representation:
(lambda [f]. (diff(f))(1)) (inverse(lambda [x]. plus(power(x,2),1)))
Output in English:
Calculate $f'(1)$, where f is the inverse of the polynomial $x^2 + 1$.

Figure 2.3: Example exercise from WebALT, the internal representation and its representation in natural language. [SC05]

spots of a template sentence instead of completely constructing a sentence from scratch. We cannot use this approach without modifications, as every run of the program with the same 'logical representation' will result in equal text. We want the program to be able to create several versions of the same representation of the exercise.

While our system will not focus on multilingual output, the research performed by Higgins et al. [HFD05] shows us that there are not only grammatical, but also cultural differences between languages. Because German is not our native language, this is important to remember during the development and evaluation of a NLG system.

2.1.2 WebALT

The Web Advanced Learning Technologies (WebALT) project [SC05] developed a framework that is able to store language independent mathematical exercises, and express them in exercises in many languages. One of the main goals was to develop a program that is capable to output text in many African languages, using as many existing resources and frameworks as possible. With the created framework, exercises could be authored and presented to a student in the language and culture of that student. The typical exercise includes sentences with both words and formulas that together make a math problem. An example of the internal representation of an exercise and its representation in English is given in Figure 2.3.

T started as a research project, but is now commercially published by the WebALT company. Therefore we will not be able to use the WebALT software as a basis for our research. However, we will discuss their use of OpenMath and the Grammatical Framework software, as both are freely available and can possibly contribute to our research.

OpenMath

The OpenMath framework [ADS96] is one of the existing frameworks that is used in the WebALT software. OpenMath is used to save and show the mathematical formulas embedded in the exercises. OpenMath[1] is a standard for representing mathematical expressions and objects, and makes it possible to exchange those mathematical structures between several computer programs and to store them. Where other techniques (such as MathML or the functionality of LaTeX to

[1] http://www.openmath.org/

display formulas) are only able to display mathematical objects, OpenMath has also knowledge of the semantic meaning. This information can be used in the design process of an exercise.

Grammatical Framework (GF)

The WebALT project uses the Grammatical Framework (GF) [Ran04] for multilingual capabilities. With the GF, the system is able to render the mathematical exercises in various European languages. To make the system able to output in African languages, the GF had to be extended with African language resources.

The GF is a grammar formalism designed for defining multilingual grammars [Ran04]. The goal of GF is to design a system that could be made by linguists and then used by linguistically untrained programmers to make linguistically correct applications for a certain domain. The 'GF resource grammar libraries project' covers grammatical libraries for 10 languages, also for the German language. These libraries contain functions for syntactical combinations (sentences, verb and noun phrases etc.) and some structural words. The use of libraries decreases the amount of information that is needed by the system, because most languages share over 75% of all code. However, rules to apply the correct morphology to the text are not included in the libraries for GF. Instructions for the morphology have to be added for every new context that will be used. The GF system only guarantees that the sentences that are created are grammatically correct.

Conclusion

In the WebALT project, OpenMath is used to represent mathematical exercises. While our program also creates exercises, we expect that we will not use OpenMath to represent those mathematical exercises. The exercises that we will consider are relatively simple, and it will be much easier for a researcher if we create our own representation.

In order to be able to use the GF for our program, we have to create a lexicon with the words that we will use. If we use the available grammatical resources for GF, it is very difficult to vary the textual output: GF ensures grammatical correctness, but does not give many possibilities to influence the exact wording; it will always choose the same words.

2.2 The Narrator in the Virtual Storyteller

The Virtual Storyteller is a system created at the Human Media Interaction department of the University of Twente that is capable of creating fairy tales. The system will first generate a plot, which is based on the actions of character agents in a virtual world. Each character is 'played' by an agent and reasons logically and tries to achieve some personal goals. Based on the actions of all characters, a fabula is constructed. This fabula is the script of the story, and is implemented as a causal network. This network expresses the actions, goals

and emotions of characters, and the relations between them. In this fabula, the temporal structure of the story is also embedded.

Based on this fabula, the Narrator will construct a Dutch text that expresses the plot of the story in natural language. Originally, the narration was a simple mapping of the actions in the fabula to text by use of simple sentence templates, which resulted in very monotonous texts. To improve these texts, the narrator was improved by Slabbers [Sla06]. The more sophisticated Narrator also uses the structure and dependencies in the fabula as input.

2.2.1 The NLG processes in the Narrator

The Natural Language Generation (NLG) processes in this Narrator are mainly based on the general NLG pipeline as proposed by Reiter and Dale [RD97]. This pipeline, also the basis for many other NLG systems, is schematically shown in Figure 2.4. In this pipeline, the first module is the 'Text Planner'. In this process, the contents and structure of the text is determined, based on the goal of the text. This will result in a text plan that is used in the 'Sentence Planner'. This module is responsible for lexicalization, referring expression generation and aggregation[2]. In the 'Linguistic Realiser' the actual text is constructed, and the correct morphology (the use of the right inflection of words) is chosen.

This pipeline structure can be found in many NLG systems, however, the actual names of the modules could be different and some tasks like the aggregation of sentences are performed in different modules. In the Narrator, the modules have the following names:

- Text Planner is called 'Document Planner'
- Sentence Planner is called 'Micro Planner'
- Linguistic Realiser is called 'Surface Realiser'

The Text Plan (or 'Document Plan' in the Narrator) is structured as a tree, encoding the dependencies and rhetorical relations between the sentences in the text. Also, every sentence is represented as a tree.

Slabbers [Sla06] argues that the dependency trees and rhetorical relations that are used are largely language independent. It should therefore be relatively easy to adjust the Narrator to be able to generate German texts, because the first process in the pipeline is language independent. The other processes should be changed or replaced by processes that are able to apply the German grammar.

Closer inspection of the source code shows that not only the grammar rules have to be replaced, but also some language specific code should be replaced. Some text are not generated, but predefined. To be able to generate text in a different language, a rough inspection of all the source code of the Narrator has to be performed.

Furthermore, the language generated by the Narrator focuses on the structures used in fairy tale stories. In these fairy tales, there are many possible rhetorical relations between sentences that should be expressed in the text. The rhetorical

[2]All of these concepts are used in Genpex. We discuss these concepts in Chapter 5.

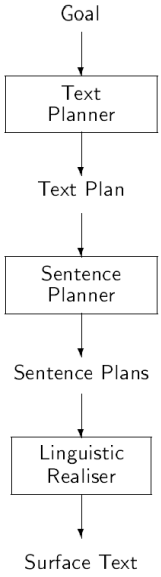


Figure 2.4: The NLG pipeline as proposed by Reiter and Dale [RD97] and used in the Narrator of the Virtual Storyteller

relations used in the Narrator are additive, cause, contrast, purpose and temporal. Certain cue-words are used to express these rhetorical relations in the text.

We will give some examples, the cue-words are marked in bold face:

- Additive: *De ridder sloeg de prinses **en** zij ging naar het bos.*
  *(The knight hit the princess **and** she went to the forest.)*
- Cause: *De ridder sloeg de prinses, **dus** ging zij naar het bos.*
  *(The knight hit the princess, **so** she went to the forest.)*
- Temporal: *De ridder sloeg de prinses, **voordat** zij naar het bos ging.*
  *(The knight hit the princess, **before** she went to the forest.)*

In the Narrator, this process of choosing correct cue words is performed in the last module in the NLG pipeline, but the information that is necessary for this process is embedded in the Document Plan in the first step of the whole process. It is expected that in the exercises that we need to generate, only an additive relation between sentences is used. It may be an overkill to translate and adapt the whole rhetorical relation model in the system that we are going to use, because only one relation will occur.

2.3 Aggregation and ellipsis

Theune et al. [THH06] discuss that in the Narrator, a lot of effort has been put into the use of aggregation and ellipsis to improve the quality of the generated

text. Aggregation is the process in which two or more sentences are combined into one new sentence. After aggregation is applied, ellipsis is a technique that removes unnecessary words from this new sentence.

For example, we could aggregate the following two sentences into one new sentence:

- De prinses at een appel.
  (The princess ate an apple.)
- De kabouter at een peer.
  (The dwarf ate a pear.)

The new sentence:

- De prinses at een appel en de kabouter ~~at~~ een peer.
  (The princess ate an apple and the dwarf ~~ate~~ a pear.)

The process of aggregation involves the following process:

1. Select the trees of the sentences that are suitable for aggregation
2. Select the correct cue word based on the rhetorical relation
3. Perform aggregation
4. Perform ellipsis

Ellipsis

Ellipsis, the removal of redundant words, is used in many NLG systems and also discussed by Harbusch and Kempen [HK09]. It is optional to perform aggregation or ellipsis. It is possible to vary the number of sentences that is aggregated. The number of redundant words that is removed during the ellipsis process can also differ. Both techniques might therefore be useful as a technique that adds variation in the language of a text.

It is not possible to delete random words in aggregated sentences. Ellipsis is only performed on identical nodes in a tree that represents the aggregated sentences. A node can represent a word, or group of words in the sentence. The words in aggregated sentences should fulfill the following constraints before they can be selected for ellipsis:

1. Lemma identity: Two words belong to the same inflectional paradigm (e.g. live, lives)
2. Form identity: Two words are the same, have the same spelling and belong to the same type of word (e.g. the word 'want', but only if both are nouns or both verbs, but not if one is a verb and the other a noun)
3. Coreferentiality: If the two words are nouns, they refer to the same object in the real world.

If one or more nodes in a tree are selected, one of the following forms of ellipsis is performed:

- Gapping (where the verb of the second conjunct is removed)
  Example: *The princess ate an apple and the dwarf ~~ate~~ a pear.*
- Right Node Raising (removal of the rightmost part of the first conjunct)
  Example: *The princess found ~~an apple~~ and the dwarf ate an apple.*
- Conjunction Reduction (removal of the subject of the second conjunct)
  Example: *The princess ate a pear and ~~the princess~~ found an apple.*

A combination of those forms of ellipsis is also possible, such as the combination of 'Right Node Raising' and 'Conjunction Reduction':

Example: *The princess found ~~an apple~~ and ~~the princess~~ ate an apple.*

Harbusch and Kempen [HK06] have developed ELLEIPO. This is a module that could be used to add all of those kinds of ellipsis to existing NLG systems. However, to be able to use ELLEIPO, the sentence that is used as input for this module should be well defined as a specific tree structure.

2.3.1 Conclusion

It is not likely to use the existing Narrator created for the Virtual Storyteller. The requirements on the texts are created by the Narrator are different that our requirements. We should focus our research on different aspects of language generating. However, we can still benefit from the research that has been performed in order to create the Narrator. We expect that the techniques that we discussed, such as aggregation and ellipsis, can be used in our program. Because those techniques are not necessary to generate correct language, we might use them to add variation to the texts that we are going to generate. We will also use the same sort of NLG pipeline as has been proposed by Reiter and Dale and used in the Narrator.

2.4 Resources

For the language generation process, a system needs to have some knowledge of the grammatical rules of a language. Furthermore, inflections of words need to be known to be able to create correct sentences. It can be useful to use existing resources. In this section, we discuss the most important resources for the German language.

2.4.1 GermaNet

GermaNet[3] is a German WordNet. It is a resource in which nouns, verbs and adjectives are grouped into lexical units, so called synsets. Every synset is a collection of synonymous words. There are relations defined between synsets, indicating for example antonyms, hypernyms and hyperonyms. GermaNet is primarily intended to be a resource for word sense disambiguation. This is mainly

[3] http://www.sfs.uni-tuebingen.de/GermaNet/

necessary in information retrieval applications. To be able to use GermaNet in a language generation system, it should be connected to other resources. While it is able to retrieve useful hypernyms or hyperonyms, GermaNet will not give the inflection tables or correctly inflected words.

2.4.2 FrameNet

In a FrameNet, the connection between words is defined. The basic idea is that a single word has no meaning: for example, it is impossible to know the meaning of "sell" without knowing anything of the situation: you'll have to know that sell also involves (among other things) a seller, a buyer, some goods that are sold and money. All this information is embedded in a frame and is used to describe an object, state or event. It takes a lot of time and effort to create a FrameNet. There is an English based FrameNet, developed at Berkeley[4]. However, a German FrameNet is still in development[5].

2.4.3 Canoo.net

The website Canoo.net[6] is an online German morphology dictionary based on the 'Canoo language products'. It was developed in a cooperation between researchers at the University of Basel, the Vrije Universiteit Amsterdam, the IDSIA Lugano (translated as: Dalle Molle Institute for Artificial Intelligence) and Canoo Engineering AG. The dictionary contains 250.000 lexemes and generates more than 3 million word forms.

In addition to the available dictionary, the Canoo.net website contains a lot of information about the word and sentence grammar, and other information about the German Grammar such as inflection and word formation rules.

2.4.4 Conclusion

There are resources available for the German language, but none of the discussed resources is directly suitable to use in our application. Even if we are not going to use these resources directly in our program, we can use the information that is available during the design of our program. During future extension of our program it might be useful to still add these resources.

2.5 DEG: Demonstration Exercise Generator

During a Capita Selecta assignment, a demonstration program was developed, which we refer to as DEG. [BR09] With this Demonstrator we have shown that it is possible to generate multiple different exercises that vary in domain and wording but are (supposed to be) equally difficult to solve. This variation was

[4]http://framenet.icsi.berkeley.edu/
[5]http://www.laits.utexas.edu/gframenet/
[6]http://www.canoo.net

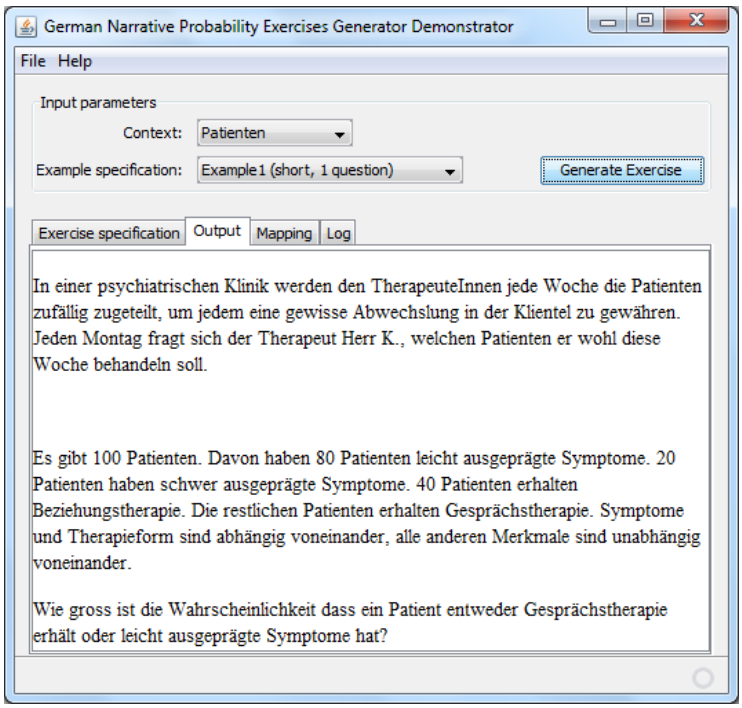


Figure 2.5: Screenshot taken from the Demonstration Exercise Generator

made by using different context stories and by randomly applying (a simple form of) aggregation of two sentences into one new sentence. Because of the demonstration purposes of the system, it was very limited and not usable for further research with the created exercises.

2.5.1 The functionality of DEG

During earlier research, the department of Statistics and Quantitative Methods for Psychology (University of Münster) made a number of narrative exercises. As part of their research, they performed some field trials with these exercises. All exercises were made by hand. With DEG, we tried to generate exercises that were similar to these exercises, shown in Figure 1.1.

DEG is able to generate exercises given the mathematical structure of the exercise. We called this input the exercise specification, and after choosing a context a text was generated. A screenshot of this program is given in Figure 2.5. For the Language Generation in DEG we used the Exemplars framework, a template based NLG system. [WC98]

2.5.2 The limitations of the DEG

The functionality of DEG was very limited: the exercise specification had to be completely designed by hand. This is both a lot of work and sensitive for

mistakes. For example, it was possible to create exercises with DEG that are impossible to answer.

We created the system by using a template-based NLG system, in our case Exemplars. We developed only two simple templates (one for statements and one for questions) and therefore all generated sentences looked very similar. It was possible to introduce some variation by adding more templates to the system, and randomly apply one of these templates. However, if we wanted to use more variation techniques, we should have added a new template for every combination of variation techniques. If we wanted to introduce more sophisticated variation, we were thus limited by the template model.

In DEG, a simple form of aggregation (combining two or more sentences in one new sentence) was applied, but the ellipsis of words (removing unnecessary words) was not used. To be able to perform ellipsis, words should meet some requirements (as discussed in Section 2.3). Due to our template based approach, some information was not directly available. It was unknown whether two words did belong to the same type of word, for example if both words were nouns.

Another disadvantage of Exemplars was that the documentation was limited: while the software was written in Java, the source code was not available and it was therefore difficult to debug DEG.

2.6 General conclusions

In the previous sections, we have seen several systems that are capable of generating exercises. Some language generation systems have been discussed, and also some resources for the German language. We have also discussed the previously created demonstrator, DEG. In this section we will discuss what techniques we can use in the development of Genpex. We will also discuss why we will not change an existing language generation system, but have create our own.

2.6.1 Existing systems

As a template-based system is likely to restrict our possibilities of variations in the generated language, we will not use such a system, including Exemplars. The output of the Narrator is Dutch; to change this into German will take a lot of time. Furthermore, the program has a different focus compared to the system that we are going to develop. As a result, some unnecessary functionality needs to be translated or removed. For example: in the Narrator several possible rhetorical relations between sentences are distinguished. Those relations will not be used in the text of the exercises that we are going to generate.

Some other systems are less suitable to use, because we are not able to vary the output when using them. Both GF (the Grammatical Framework, used in the WebALT project and discussed in Section 2.1.2) and ELLEIPO (a module that can add ellipsis techniques to existing NLG systems, discussed in Section 2.3) restricts us in possible variations, as they will always apply the same techniques

when supplying a certain input. A possible source of variation is excluded if we cannot choose to apply a certain technique or not.

If you want to use an existing system, an interface to interact with these systems has to be created. Information should be formatted so that the existing system can process it, and the output of this system should be processed too. The time to implement and test this interface might not be compensated by the extra functionality or information that is added by using these existing systems. It might save time by adding information manually.

2.6.2 Reuse of techniques and resources

We have seen many existing systems, some capable of constructing whole exercises, others focused on language generation tasks. Unfortunately, we cannot use any of these systems without large changes to the software. Nevertheless, we can still benefit from the research that has been performed and use some ideas and techniques used by these systems.

We have argued that we want to use a more in-depth NLG approach, instead of a template-based NLG system. In the Narrator, sentence trees are used to represent the contents of a sentence. With those sentence trees, both the structure of the sentence and words are represented. It is also possible to represent the word order of a sentence in a sentence tree.

Possible techniques that can be used to add variation are aggregation and ellipsis. We could randomly apply those techniques when possible, and create several versions of the same sentence.

A logical representation of the mathematical exercise, as is being used by the ModelCreator software, is useful. Because the representation used in the ModelCreator software is focused on QC type exercises, and we will make exercises reasoning about probability problems, we will use another representation. As we discussed in Section 2.1.2, we will define our own representation instead of using for example OpenMath, because of the simple structure of our exercises.

A valuable resource for the German language will be the information available on Canoo.net. The information in GermaNet can be used, if we want to support synonyms and hyperonyms of words.

It is argued by Enright et al. [EMS02] that variations introduced within a NLG system should not have influence on the difficulty of the exercise. However, a different textual representation of a mathematical exercise might influence the students' capability of answering the involved questions. During our research we will not test for a possible connection between a variation technique and the difficulty of an exercise. It is therefore important that we keep track of all text variation techniques that are applied. This information can be used in further research. Certain techniques might then be excluded from the text generation process.

Chapter 3

Genpex: the developed program

During our research, we have developed Genpex, the Generator for Narrative Probability Exercises. In this chapter, we give a global overview of the program in Section 3.1. The most important subtasks of Genpex will be discussed in seperate chapters. These subtasks are the creation and checking of probability problems (that we discuss in Chapter 4) and the conversion of the probability problems into text (Chapter 5).

In this chapter, all other aspects of Genpex are discussed. This includes the Graphical User Interface (GUI), discussed in Section 3.2 and the contents of the context and language files that are used by Genpex in Section 3.3. In Section 3.4 we show the working prototypes of two tools that can help the user with the development of new contexts. Finally, we discuss some details of the technical implementation of Genpex in Section 3.5.

3.1 Global system design

Similar to the ModelCreator software discussed in Section 2.1.1, the main task of our system is split into two subtasks, the creation of a representation of the mathematical exercise and based on this representation the generation of the language. The structure of Genpex is similar to the structure of ModelCreator. The input, the mathematical representation and the processes are different.

As we have seen in the introduction, an exercise is not just a story with a question, but it is a text in which a mathematical exercise is embedded. In our case, a student is required to calculate a certain probability. We will therefore call the mathematical exercise that is embedded in the text a **probability problem**. A probability problem defines the mathematical exercise of our word problems completely.

In this chapter, we give an overview of the global tasks of Genpex and shortly discuss their functions. The two most important processes will each be discussed

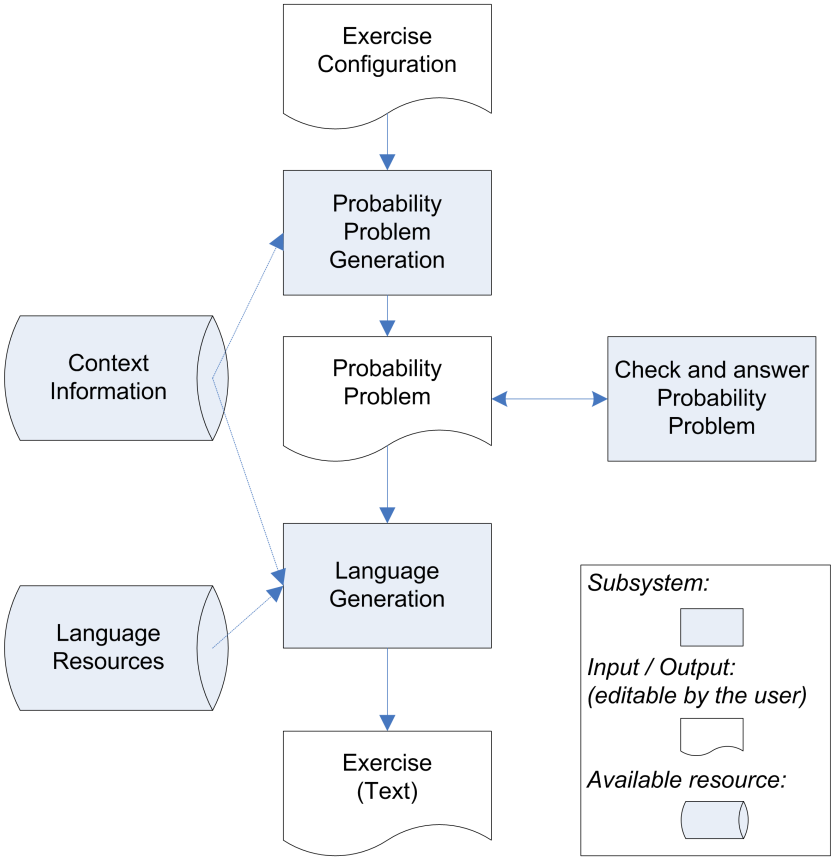


Figure 3.1: Global System Design

in more detail in separate chapters:

1. Creating a probability problem in Chapter 4
2. Creating a text in which this probability problem is embedded in Chapter 5

To create a new exercise, one should run both processes directly after each other. We use this structure to make it possible to create a probability problem and edit it before the text is created. It is therefore an interactive system. Furthermore, because variation is introduced in both subtasks, it is possible to create different exercises by skipping the probability problem generation process and applying the language generation process multiple times on the same probability problem.

The global system design of Genpex is visualized in Figure 3.1. We describe every element in this figure briefly and refer to the chapters or sections in which it is discussed in more detail.

Exercise configuration
The input for the system is the exercise configuration. This configuration, that can be configured via the GUI (Graphical User Interface) of Genpex, guides the generation process. It includes information about the context that should be used and the (type of) probability problem that has to be created, which

is discussed in Chapter 4. Also, it specifies the probability that a certain text variation technique will be applied. The options for text variation are discussed in Chapter 5 during the discussion of the developed program.

Probability Problem Generation
A probability problem is generated based on the information specified in the exercise configuration. To be able to create a correct probability problem, the system needs information about the context of the exercise (see below). Probability Problems are discussed in Chapter 4.

Context information
The context is the theme of the exercise, for example 'bikes' or 'rooms in a hotel'. The information about a context is used in both the generation of probability problems and the language generation process. The exact content embedded in a context is discussed in the chapters about the probability problem and language generation. For every available context, a context file with this information is available. The structure and contents of this file is discussed in Section 3.3.1.

Check and answer probability problem
As soon as a probability problem is edited by the user, it should be checked. It is possible that the user has created an incorrect problem that is for example impossible to answer. To be able to check the correctness, this process will calculate the answer to all questions embedded in the probability problem. Obviously, the probability problem is only correct if all embedded questions can be answered. The answers will be embedded in the probability problem. In Section 4.5 we will discuss this process in more detail.

Language generation
The language generation or NLG (Natural Language Generation) process in Genpex is split into several smaller subtasks and is discussed in Chapter 5. This process converts the information embedded in the probability problem into text. This process is guided by information specified in the exercise configuration. This process also uses the information embedded in the context. A Language Resource is used, a dictionary containing information about words that are used in the text of the exercise. This resource is also discussed in the chapter about the language generation process in Genpex.

3.2 Graphical User Interface

The GUI (Graphical User Interface) is split into two tabs. On the first tab, shown in Figure 3.2, the probability problem is given. There are also various options available to create or change this probability problem. The other tab, shown in Figure 3.3, gives the text of the exercise and options to adjust the variations used in this text.

3.2.1 Status box

There is a text box called 'status' placed on the bottom left corner on both tabs. This 'status' box gives information about the actual status of the program. The

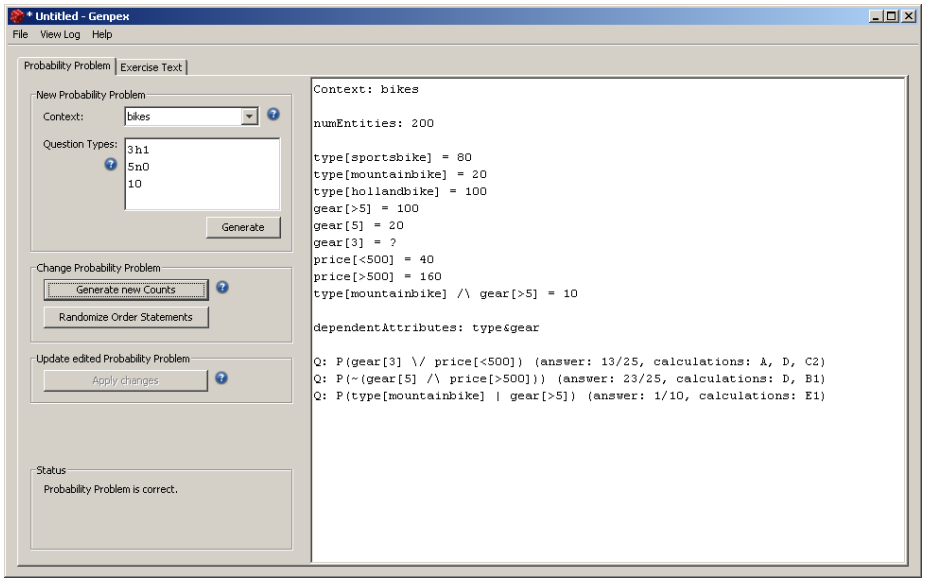


Figure 3.2: Screenshot of Genpex, showing the Probability Problem tab

text is colored red if an error has occurred. The exercise is not updated until this error is resolved. The information in this box will help the user to resolve this problem. Messages that are not errors have a normal black font.

3.2.2 Probability problem tab

In the probability problem tab, the current representation of the probability problem is shown. This representation can be edited by hand, or as we will discuss in Section 4.4 automatically generated based on a context and one or more question types.

To create a new probability problem, the user should select one of the available contexts. Genpex will only show a context if it is available and correct. In Section 3.4.1 we show a tool that checks context files and shows the errors if they exist.

After the user has generated a probability problem, two buttons become available. Those buttons can be used to easily change the generated probability problem. It is easy to make several variations of the same exercise. The first button generates new counts that will be used in the new probability problem. The second button randomly reorders the statements in the probability problem.

If a probability problem is edited by hand, the syntactical structure of this probability problem is checked automatically. If this is correct the button 'Apply changes' will be available. After a user presses this button, the given textual representation of the probability problem is parsed and an internal representation is constructed. Based on this internal representation, Genpex will calculate the answers to all questions embedded in this probability problem. The answers

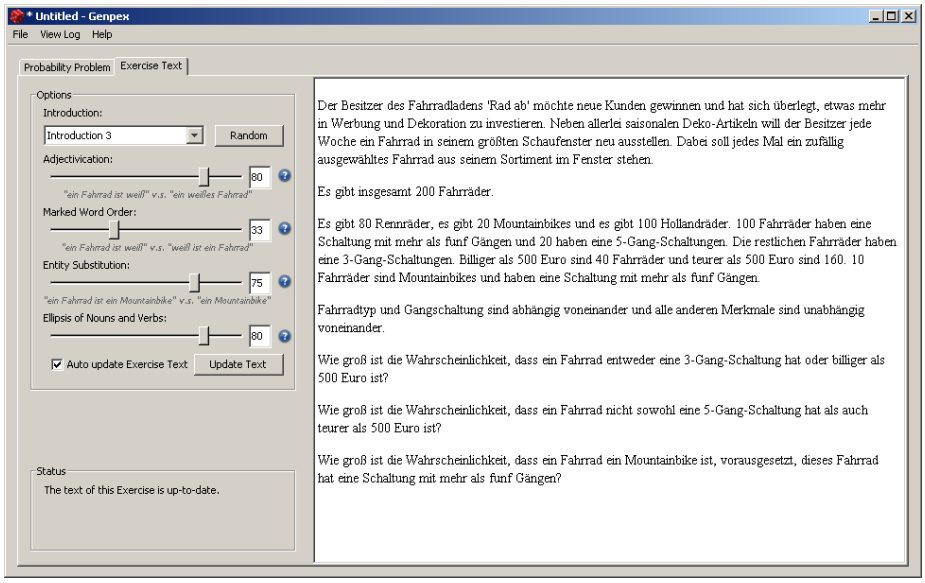


Figure 3.3: Screenshot of Genpex, showing the Exercise Text tab

are added to the textual representation of the probability problem that is shown to the user. The text of the exercise on the exercise text tab is also updated.

After the internal representation of the probability problem is constructed another check is performed: whether the specified question types still correspond to the questions that are specified in the probability problem. The user could have entered the question types and a probability problem manually. Therefore, it is possible that they do not match. This check is performed by comparing the calculations that Genpex executed to derive the answer to a question with the calculations that are expected for each question type. If they match, it is expected that a question matches a question type, otherwise a message is shown to the user.

3.2.3 Exercise Text tab

In the exercise text tab, the exercise text is shown. This text is always an actual representation of the probability problem in the probability problem tab. If the text could not be updated, this is indicated in the status box.

There are several options to vary the text. It is possible to (randomly) choose another introduction text (if there is more than one introduction available in the context). Selecting a different introduction will directly be reflected in the text. The introduction text will be replaced, the rest of the text is not changed.

All other text variation options can be controlled with a slide. The user can choose the probability that a technique is applied. This probability (also represented by a number between 0 and 100) can be changed with this slide. If one selects a probability of 50%, that technique will be applied in every situation that that technique is possible with 50% chance. As soon as the user has

changed one of the values of the slides, the text is updated. We discuss this process in more detail in Section 5.4.

The text is always updated after a change of one of the slides. The user can remove a check from the checkbox 'Automatic update text' to prevent this behavior. There is a button that can be used to manually update the text.

3.2.4 Other features

Genpex has some other features that are available via the menu of the program.

View log files

As we have seen, as soon as a new correct probability problem is given or generated, the corresponding exercise text is (re)generated. Several log files are created during the generation process. We will discuss in Section 4.4 that by generating a new probability problem an internal world is designed. This world is logged and can be viewed. The answers and required calculations to the questions in this probability problem are logged. The log files concerning the language generation process list the sentence trees in different situations. There is also a log file where the language variation techniques are logged. It is listed how many times a technique is applied, summarized per attribute. All log files can be used for debugging or for analysis in future research.

Help

A help function is available throughout Genpex. This help function consists of multiple pages and discusses all functionality of the program. It can be opened via one of the menu options. A number of help icons is placed directly next to buttons on both the probability problem and text tab. These icons will open the help files directly at the page that discusses the functionality that is located next to the help icon.

In the help files, there is also an overview of the possible question types and calculations that can be performed by Genpex.

Open and save

An exercise that is created can be saved. A saved exercise can also be opened again. Not only the probability problem and exercise text, but also all specified options and log files are saved in this file. Because this file is saved in plain text, it can be easily used in further research. Genpex will give a warning whenever the program is being closed without saving the changes.

A probability problem and the exercise text can be exported separately as an HTML file. They can therefore be opened in a normal internet browser.

3.3 Resources

Within the system, some resources are available. Those resources have been created specifically for this system, and can be used to easily extend the system. In the future Genpex may be connected to already available resources, as we will discuss in the chapter about future work.

Both files are encoded as an XML file. We first discuss some technical details and continue by discussing the contents of the context files and our dictionary.

3.3.1 Context files

The **context** is the topic or theme of the exercise. For every context, there is a context file that defines this context. The information in these files is used in the creation of probability problems, which we discuss in Section 4.4. Except for the information that is used to create the probability problem, there is also information needed for the textual representation of the concepts used in the context. Therefore, context files contain all information about a context, and are used in both the generation of the probability problems and the language generation.

Some information in the context files that is needed for the language generation is encoded as a (partial) sentence tree. The textual representation of the sentence trees is used in the context file. An example of this representation is shown in figure 5.4. Also, the language of this sentence tree is indicated. In our examples, this is always 'de' to indicate that this language information is specific for the German language. A simple sentence tree, consisting of a single noun is embedded as follows:

```
<string language="de" value="[noun]Fahrräder[/noun]"/>
```

If a string is specified, but does not have the structure of a textual representation of a sentence tree, it is considered as canned text (see Section 5.2.3).

In the context file, some global information is specified about the context it describes. Every exercise reasons about an entity with certain values for certain attributes. In a context file, some information specifies the context and its properties. Every context reasons about one entity, and therefore, some information is specified once per context. Other information is specified for every attribute of the entity. We will discuss the information per context and per attribute separately.

Information per context

A part of the XML file that specifies the information of the context is given in Figure 3.4. Of course, in a context an entity is defined. An entity has a name, and a representation that is used in the text to refer to this entity. In the stories we generate, the entities are objects and therefore the representation is mostly a noun, but a noun phrase can be used too. This representation is included as a tree node, and is encoded as the textual representation of a sentence tree.

```
<entity>
<representation>
<string language="de" value="[noun]Fahrräder[/noun]"/>
</representation>
<range min="50" max="100"/>
<dependentattributes attributename1="price"
   attributename2="type"/>
</entity>
<introduction>
<string language="de" value="Ein Fahrrad-Fachgeschäft
   vertreibt verschiedene Arten von Fahrrädern in seinem
   Laden."/>
</introduction>
```

Figure 3.4: The beginning of the context file with information about the context 'bikes'. In this part the information that is once specified per context is shown. The text of the introduction is abbreviated.

For the entity, a range is specified. This range defines the minimum and maximum number of entities that are present in the world, sketched by the probability problem. This maximum of the range should be large enough to be able to create a correct probability problem.

As we have said, every entity has a number of attributes. Within a context, some attributes of the entity are dependent on each other, all others are not dependent. Every attribute couple that is dependent should be specified.

Every context includes one or more introductions. These introductions are just plain text, and should be correct German sentences. We discuss in Section 5.2.3 that these texts are just copied into the exercise. It is therefore important that they are already grammatically correct.

Information per attribute

In Figure 3.5, a part of the context file that describes a single attribute is shown. In every context (and thus for every entity) multiple attributes are defined. Every attribute has a name that is used to refer to this attribute in for example the probability problem.

For every attribute, a word that connects the entity with the attribute value is given, mostly a verb or verb phrase. In our example, a colour is connected to the entity (a bike) by the verb 'sind' (are). This indicates for example that bikes 'are' white.

There is also a textual representation of the name of the attribute. In our example this is 'Farbe' (colour), and will be used to express in the exercise text which attributes are dependent on each other.

Every attribute has a list of options, each option has a value that is a number between 0 and 100. The possible options (adjectivication, entity substitution, ellipsis and marked word order) refer to the text variation techniques that we

```
<attribute name="colour">
<connection>
<string language="de" value="[verb]sind[/verb]"/>
</connection>
<representation>
<string language="de" value="[noun]Farbe[/noun]"/>
</representation>

<options>
<adjectivication>100</adjectivication>
<entitysubstitution>0</entitysubstitution>
<ellipsis>100</ellipsis>
<markedwordorder>100</markedwordorder>
</options>

<value name="white">
<string language="de" value="[adj]weiß[/adj]"/>
</value>
<value name="black">
<string language="de" value="[adj]schwarz[/adj]"/>
</value>
<value name="green">
<string language="de" value="[adj]grün[/adj]"/>
</value>
<value name="blue">
<string language="de" value="[adj]blau[/adj]"/>
</value>

</attribute>
```

Figure 3.5: The information specified in a context file for a single attribute.

discuss in Section 5.4. The number indicates the probability that this technique should be used in sentences that discusses this attribute.

The attribute has multiple possible values, all are specified separately. Each value has a name and a representation in natural language. The order of the attribute values is not important; if an attribute value is needed, one attribute value is selected randomly.

Due to the structure used, there are some restrictions on the possible contexts. It is for example impossible to phrase an attribute value in two ways. In a normal situation you can say that a person comes from the Netherlands, or that a person is Dutch: both mean exactly the same. In this structure there cannot be more than one representation of an attribute value. It is also impossible to have more than one representation of the connection of an attribute value with the entity.

The same sort of restriction is that different attribute values from the same attribute cannot be connected to the entity by using different connection words. If we talk about the attribute ‘caffein’ for the entity ‘coffee pads’, then the

```
<noun name="colour" language="de" singular="Farbe"
  plural="Farben" flexionsklasse="-/en" artikel="die"/>
<verb name="sind" language="de" singular="ist" plural="sind"/>
```

Figure 3.6: A small part of the dictionary file. Every noun and every verb that is used in any of the contexts has to be included in this file.

possible attribute values are 'yes' and 'no'. If we express both attribute values in a German sentence, the preferred way is: '5 Kaffeepads enthalten Koffein' and '5 Kaffeepads sind Koffeinfrei'. Because of the restriction on the structure, it is impossible to embed this information. It is still possible to include this attribute value by rephrasing the second sentence so that it uses the same connection word as the first sentence: '5 Kaffeepads enthalten kein Koffein'. Unfortunatly, this representation is less likely to be used in German.

3.3.2 Dictionary for the inflection of nouns and verbs

Genpex needs only a small amount of information to inflect words, because all sentences are in the present tense, and all words should be inflected for the third person.

We therefore designed our own resource, a simple dictionary containing the inflections for the nouns and verbs used in the context files. For verbs, we include a inflection for the singular and plural case. Nouns also include an article, and an inflection class. In future research this dictionary can easily be extended or replaced by a resource that contains more information.

This dictionary is also encoded as a simple XML-file, with an entry for each verb and noun. An example is given in Figure 3.6.

3.4 Tools

As we have discussed in the previous section, the context files and dictionary file are separated from the rest of the program. Because those files are encoded as XML-file, it is possible to use any text editor to edit those files. However, as we discuss in Section 8.2.1, a specialised editor will have many benefits. This editor should have special features to help the user with creating and editing those files. Because it takes a significant amount of time to develop a specialized editor, this was not done.

It is hard to edit the context and dictionary files manually, because there are many restictions on the files. To help the user, we have created two tools that can be used by the development of new context files. Some of the source code of Genpex was reused for these tools. It was therefore possible to create the tools in a short time. The tools should be used in combination with a normal text editor. They are already useful for creating or editing contexts and might be included in a specialised editor.

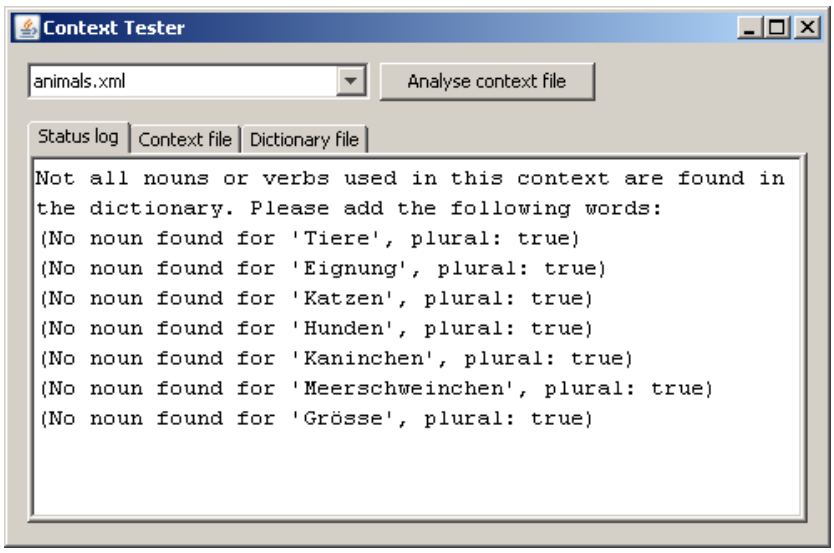


Figure 3.7: A screenshot of a tool that tests context files. It will test the structure of the context file and whether all nouns and verbs are available in the dictionary file.

3.4.1 Context file tester

As we have seen in Section 3.2, in the GUI of Genpex the user can choose a context. Genpex tests the structure of the file and only shows a context if its context file is correct.

To help current users of Genpex, we created a working prototype of a tool that analyses context files. This tool checks the correctness of a context file, its structure and some other restrictions. If an error is found, it shows instructions to the user to help fix the problem. It also checks whether the words that are used in the context are available in the dictionary file. A screenshot of this tool is shown in Figure 3.7.

3.4.2 Inflection retriever

For the language generation, there is a dictionary file available that contains information for the conjugation of words. The researcher that adds a new context to Genpex is also responsible for the correctness of the dictionary. All verbs and nouns used in the context file should be available in the dictionary.

Of course, it would be a great improvement if the system could make use of existing resources instead of its own dictionary file. Now, if a researcher adds information to this dictionary, he uses the Canoo.net website to find this information. This process can be automated easily. To illustrate this, we have built a working prototype that is able to perform this automatically. A screenshot of this tool is shown in Figure 3.8. In this tool, one can enter a German noun or verb. The system will retrieve all information that is required from Canoo.net.

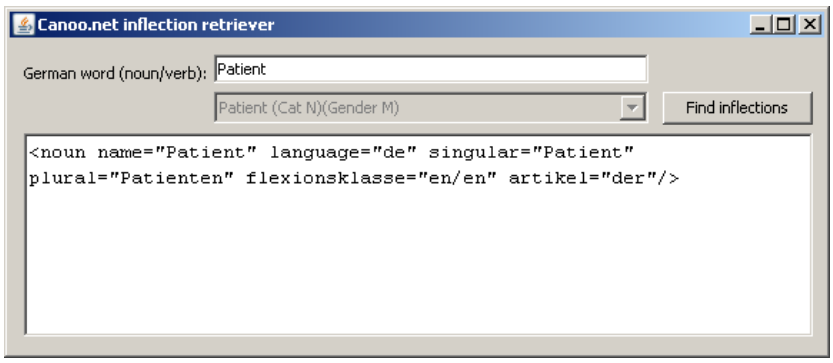


Figure 3.8: A screenshot of a (working) prototype of a tool that is able to retrieve information about the inflection of German nouns and verbs from Canoo.net. The output of this tool is in a format that can directly be added to the dictionary file of Genpex.

It will parse and format this information so that it can be directly copied into the dictionary file. We suggest to add this functionality to a context file editor.

3.5 Implementation details

We give a short overview of some of the implementation details in this section. We will not discuss the implementation in much detail, because our implementation follows the design that we will show in the following chapters. Someone who is interested in the details of the implementation can read the documentation that was created using Javadoc, which we attached to the source files.

3.5.1 Java and Netbeans

Genpex was created in the programming language Java[1]. We used the NetBeans IDE[2], an integrated development environment. The GUI was designed using the Swing GUI Builder that is part of NetBeans. The development of Genpex followed the processes that we describe in the following chapters. The most important Java packages and classes correspond to the elements in the global system design that we described in Section 3.1.

The most important packages of java classes are responsible for the following tasks:

- The package 'gui' contains all classes that are responsible for the GUI of the program. The most important class is called 'Genpex', that creates the main interface, as is shown in the screenshots in Section 3.2.
- In the package 'nlg' we implemented the natural language process, as will be discussed in Chapter 5. The documentplanner, microplanner and

[1]http://www.oracle.com/nl/technologies/java
[2]http://www.netbeans.org/

surface realizer are implemented in seperate classes. A sentence tree is represented by the class 'TreeNode'.

- The package 'probabilityproblem' has an implementation of all probability problem tasks, as will be discussed in Chapter 4.
- The package 'xml.schema' is created automatically using Jaxb (discussed below). The classes in this package are used to read the contents of the context and dictionary files.

Some of the information shown in tables and figures is hard coded in the source files of Genpex. The question types shown in Table 4.1 are implemented in the function 'addQuestion()' in the java class 'ProbabilityProblemCreator' in the 'probabilityproblem' package. The calculation rules that are shown in Table 4.7 are implemented in the function 'calculateProbability()' in the java class 'ProbabilityProblem', also in the 'probabilityproblem' package.

3.5.2 Jaxb: Java Architecture for XML Binding

For the context and dictionary files, we used XML-files because they have some advantages: the information embedded in those files is well structured and can be easily loaded into Java. The files can easily be viewed and edited by a simple text editor.

We have used the Jaxb technology (Java Architecture for XML Binding)[3] for these resource files. By using Jaxb, we were able to automatically create Java classes that represent the information in the XML-files. The files created by Jaxb can be used to easily read and write XML-files. Data embedded in the XML-files is mapped to the Java-objects created with Jaxb.

3.5.3 Information embedded in the source

As we have seen in the global system design in Figure 3.1, both language and context information is used throughout the system. We will shortly discuss the methods that use this information.

Context information in source files

To add a new context to the system, the source code does not need to be edited. A new context can be added by creating a new context file, as we have discussed in Section 3.3.1. If in this context file new nouns or verbs are used, those words should be added to the dictionary file. As soon as a correct context file is available, the user of Genpex will be able to use it. The tools discussed in Section 3.4.1 and Section 3.4.2 can be used during the development of the new context file.

If someone wants to extend the type of exercises, it is likely that he should add extra information to a context file. This information can be used everywhere in

[3] http://www.oracle.com/technetwork/articles/javase/index-140168.html

the generation process. To add new information, he should first edit the file that describes the structure of the context files. Then the Jaxb tool should be used to automatically update the Java-classes that access the context files. Existing context files should be edited to match the new structure. Of course, the tool that checks the context files should be extended too. The new information is directly available in all processes that make use of the context information.

Embedded language specific information

It is possible that for a future version of Genpex, someone wants to make changes to the language that is used in the exercises. If someone wants to change the language used in an exercise, for example to add more variation to the language that is used, it is not likely that the processes that create the probability problems will have to change too.

All modules in the NLG-process share their in- and output via sentence trees. A change in the generated language will be reflected in those sentence trees. Because of the pipeline structure that is used, only the modules that follow the module that is changed have to be checked. It is possible that the change in the language used changes the structure of the sentence tree, so that it is impossible to apply certain techniques on the changed sentence trees.

If someone wants to extend Genpex with a new language, such as English or Dutch, he should make more radical changes. First, information for these new languages has to be added to the context information files. Then the document planner should be changed so that it can distinguish the languages. We expect that only small changes are needed in the techniques used in the micro planner, because of the similarities between the languages. Of course the surface realizer has to be changed, so that it will be able to correctly apply the inflections and grammar rules of the new language.

In a new language, some text variation techniques might not be applicable. On the other hand, it might open opportunities to use other text variation techniques.

Embedded mathematical specific information

It is possible to change the process that is responsible for the generation of probability problems without affecting other processes of Genpex. However, as soon as changes are made that are reflected in the structure of the probability problem or the questions, it is likely that the language generation process has to be edited too. We have only tested Genpex for the generation of the language that represents the probability problems and questions. If someone wants to be able to use a new type of question in the probability problems, it should be checked whether Genpex is still able to answer the questions using the embedded calculation rules. Furthermore, the correctness of the generated language of expressing the new question types should be reviewed. It is likely that the language generation process has to be extended, especially the document planner that is responsible for constructing the initial sentence trees.

Chapter 4

Probability problems

In the introduction, we have shown an example exercise that was created by hand during earlier research performed at the University of Münster. One of the requirements that we have listed in Section 1.3, is that our program should be able to generate exercises that are similar to the exercises created for the research performed by Münster.

As we have discussed in Section 2.1.1, the ModelCreator software uses a 'logical representation' to express the mathematical exercise that is expressed in their exercises, and is used as input for the language generation process. We concluded that the use of a logical representation has several advantages. We need another representation than the one used in the ModelCreator software to be able to express the mathematical exercises that are embedded in our exercises.

As we have seen, the exercises that we consider are stories that describe a situation and in which one or more questions about a probability are embedded. The student has to read the text of the exercise and derive the underlying mathematical structure to be able to answer the questions expressed in the text. We call this underlying logical representation of the mathematical structure the **probability problem**.

This chapter will discuss all aspects of the probability problems used in Genpex. In Section 4.1, we analyze probability problems that we manually derived from example exercises. The structure of probability problems is discussed in Section 4.2, whereas in Section 4.3 the possible question types that we consider are discussed. We continue in Section 4.4 with the generation of probability problems, followed by the discussion of the calculation of the answer to the questions embedded in a probability problem in Section 4.5.

4.1 Analysis and terminology

If a student tries to answer the questions embedded in the text of an exercise, he will read the text and construct a representation of the mathematical exercise that is embedded in the text.

This is exactly what we did for several example exercises taken from the research of Münster. In this section, we show the probability problem that we derived from the example exercise shown in Figure 1.1. We use this example to explain the terms we use to refer to certain aspects of a probability problem.

4.1.1 Probability problem derived from example exercise

As we have discussed in the introduction, the program that we are developing should be able to generate exercises similar to the exercises used in field trials performed at the University of Münster. These field trials were part of the research called 'Rule-based item generation of statistical word problems based upon linear logistic test models for item cloning and optimal design' (or originally in German 'Regelgeleitete Aufgabengenerierung statistischer Textaufgaben auf der Grundlage von linear logistischen Testmodellen für Itemcloning und Optimal Design'). We have analyzed eight different exercises that were used in the field trials and formulated the probability problems that we derived from these texts. A publication that discusses these specific field trials is not yet available. Holling et al. [HBZ09] discuss exercises that are similar to these field trials. The basis of the exercise is equal, but the type of exercises is slightly different.

In Figure 4.1 we show the textual representation of the probability problem that we derived from the exercise shown in the introduction in Figure 1.1. Only the mathematical information that is embedded in the text of the exercise is included in this probability problem. The first paragraph of the exercise text starts with an introduction that is not necessary to be able to answer the questions. This is therefore not reflected in the probability problem.

We distinguish two parts of information in a probability problem: a list of statements and one or more questions. Every row in this textual representation is either a statement or question. The possible structures and meaning of the statements and questions are discussed in Section 4.2.

4.1.2 Terminology

We use many terms to refer to specific parts of a probability problem. Before we go into the structure and other details of probability problems, we will introduce some terminology. We explain each term by using the text given in the example exercise in Figure 1.1 and the corresponding probability problem shown in Figure 4.1.

As we have seen in Section 1.2, an exercise consists of a story that describes a situation in a certain **context**. Bikes are discussed in our example exercise. The situation is defined as a number of entities that have certain properties. In our example, there are 50 bikes (the **entities**) that are white. An entity is the main object that is discussed in the exercise, and is different in each context. Each type of entity has specific properties that we call **attributes**. Every attribute has an **attribute value**. In our example one attribute is the colour of the bike, with attribute values 'silver', 'black' and 'white', another attribute is the

```
Context: bikes

numEntities: 400

type[mountainbike] = 80
type[sportsbike] = 100
type[hollandbike] = ?
colour[silver] = 40
colour[black] = 120
colour[white] = 240
gear[3] = 240
gear[>3] = 160

colour[white] /\ gear[>3] = 96
type[hollandbike] /\ gear[>3] = 96

dependentAttributes: type&gear

Q: P(~(type[sportsbike] /\ colour[white]))
```

Figure 4.1: The textual representation of the probability problem that we derived from the exercise specified in Figure 1.1.

number of gears, with attribute values '3' and 'more than 3'. The attributes of entities are exhaustive and have disjunct values: an entity has exactly one value for every attribute.

We call every combination of two attributes **attribute couples**. An attribute couple is either dependent or independent. If we have an attribute couple with attributes A and B, they are dependent as soon as the value of B is influenced by the knowledge of the value of A. The attribute values that are used in the exercise can be dependent on the value of at most one other attribute.

As we have said, we call the underlying mathematical exercise in a text a **probability problem**. It has **statements**: statements are expressed as a list of attribute values and the number of entities that have this value. We refer to this number as the **count**. Every probability problem includes the formal representation of one or more **questions**. With the information embedded in the statements, the student is required to answer the questions, by computing a joint, conditional or complementary probability, or a combination of these.

The student can use the statements to reconstruct the **world** on which the statements are based. A completely defined world is a virtual reconstruction of all entities, where for every entity a value is specified for each of its attributes. The statements in the probability problem do not necessarily define the complete world. In most probability problems it is only possible to construct a part of the world with the given statements. This part should be sufficient to be able to answer the questions.

A probability problem is embedded in the exercise text. The problem shown in Figure 4.1 is the textual representation of the probability problem derived from

the text of the exercise specified in Figure 1.1.

4.1.3 Probability theory

Now that we have introduced our terminology, we will discuss the relation of the these terms to what is commonly used in probability theory. In this section, we will relate the concepts that we used to the discussion of probability theory by Russell and Norvig [RN03].

Attributes and events

Russell and Norvig [RN03] discuss probability theory and also introduce some concepts. The basic element that is introduced is a random variable that has a certain domain. There are three kinds of random variables distinguished:

1. Boolean random variables, which are either 'true' or false.
2. Discrete random variables, which have a countable and therefore finite domain. For example the weather is either 'sunny', 'cloudy', 'rainy' or 'snow'. All values in the domain are mutually exclusive and exhaustive.
3. Continuous random variables, which take on values from real numbers and have therefore an infinite domain.

It is obvious that our attributes are 'discrete random variables': they have a finite domain and the attribute values are exclusive and exhaustive.

Probability theory reasons about the probabilities of events given some knowledge of the world. In our exercises, all events are similar: There is always randomly one entity picked out of a finite set of entities. The probability that has to be calculated, is the probability that an entity has certain specific properties. The finite set of entities from which the entity is drawn, is either the complete set of entities that are available in the described world, or it is a subset of entities that have a given property.

If we describe an event, such as $P(A = a_1 \wedge B = b_1)$, capital letters refer to attributes, small letters to attribute values, where a_i is a value of attribute A. The event that is described here, is that we take one random entity out of the defined world, and that this entity has a_1 as value for attribute A and b_1 as value for B. The question is to calculate the probability of this event.

Dependencies

Because we can draw one random entity from a subset of entities, it is important to discuss the concept 'dependency'. Russell and Norvig explain this concept as follows: The notation $P(x|y) = 0.8$, where x and y are events, should be read as *"the probability of x is 0.8, given that all we know is y."*.

In the type of exercises that can be created with Genpex, we limit ourselves to attribute couples for which the attribute values are either always dependent or

always independent of each other. We exclude thus certain probability distributions. We will explain this by giving an example world and discuss why this example will not be used in Genpex.

Imagine a world where every entity has two attributes, each with three possible attribute values. We define attribute A with values a_1, a_2 or a_3 and attribute B with values b_1, b_2 or b_3. It is possible to define a world in which it is ensured that for all entities with property $A = a_1$ the distribution of the values for attribute B is the same as the distribution of the values of attribute B for a random entity in the world.

Let attribute B have the following probability distribution

$$P(B = b_1) = \frac{1}{3},\ P(B = b_2) = \frac{1}{3} \text{ and } P(B = b_3) = \frac{1}{3}$$

In this world, the probabilities are equal if we randomly draw a entity from the subset containing only entities with $A = a_1$:

$$P(B = b_1|A = a_1) = \frac{1}{3},\ P(B = b_2|A = a_1) = \frac{1}{3} \text{ and } P(B = b_3|A = a_1) = \frac{1}{3}$$

Because for every value of B, $P(B|A = a1) = P(B)$, we have defined that the probability of B is independent from $A = a_1$. In contrast, it might be possible that the probability distribution for attribute B is different if we draw from a subset that contains the entities with either $A = a2$ or $A = a3$. For example, the probabilities with $A = a_2$ are:

$$P(B = b_1|A = a_2) = \frac{2}{3},\ P(B = b_2|A = a_2) = 0 \text{ and } P(B = b_3|A = a_2) = \frac{1}{3}$$

The latter distribution is impossible in the worlds that we will consider: We will not look at the dependencies of a pair of attribute values separately, but all values of two attributes are either dependent or independent. If for example attribute A and attribute B are indendent, we ensure that $P(B|A) = P(B)$ holds for every value of A and B, otherwise the attributes are dependent.[1]

We can formalize our concept of dependency as following: Two attributes A and B are independent if

$$\forall a \in A, \forall b \in B : P(a \wedge b) = P(a) * P(b)$$

if not, the attributes are dependent.

In probability theory, a dependency may include more than two attributes. Still, we restrict ourselved to at most two attributes. Therefore questions like $P(B = b_1|A = a_1 \wedge C = c_1)$ are excluded from our system.

As we said, an event describes the probability that if we take one random entity out of a (sub)set of entities, that it has certain values for one or more attributes. This excludes a whole range of exercises: In probability theory, an event can also be defined as the a draw of two entities from a set, where at least one of the entities has certain attribute values. This type of questions is now excluded from Genpex.

While the restrictions discussed in this section limit the possible type of exercises, there are still many types of exercises possible. These restrictions make it

[1] If A and B are dependent, it is not necessary that for every value of A and B the statement $P(B|A) \neq P(B)$ is true. It might be possible that this statement is only true for some values.

easier to express the dependencies in natural language and we are more flexible in randomly using attribute values in questions. Even with these restrictions, we will still be able to generate exercises similar to the exercises created in research performed at the University of Münster.

4.2 The structure, meaning and limitations of statements and questions

In a probability problem, we distinguish questions and five types of statements, specifying the context, the total number of entities, single attributes with (or without) a count, the count for a combination of two attributes and dependecy information. In the textual representation of the example probability problem in Figure 4.1 all those types are used. We discuss each type of statement and use one or more examples taken from this figure to discuss the structure. We will also show the structure of questions, but they will be discussed in Section 4.3 in more detail.

It is possible to construct an exercise that has a correct structure, but that is not a correct exercise. In this section, we define together with this structure also the meaning, the options and limitation for each statement type.

The order of the statements in the probability problem, define the order of the sentences that express those statements. Therefore, if someone would change the order of the sentences in the exercise, he should change the order of the statements. Also, if in a statement attributes or attribute values are used, their order will be reflected in the generated text too.

4.2.1 The context

```
Context: bikes
```

The first type of statement defines the context of the probability problem. In this example the context 'Bikes' is used. A separate context file should be available for the system. This file contains information about this context, the exact contents has been discussed in Section 3.3.1.

The context contains information that is needed to create the text of the exercise, but also influences the generation process of probability problems. The context specifies information on the kind of entities described (in our example the entities are bikes) and lists the names of available attributes and attribute values. The names are used in every statement and in the questions. A probability problem is only correct when the references to attributes and attribute values are available in the context file.

In every probability problem there is exactly one statement that defines the context.

4.2.2 Total number of entities

```
numEntities: 400
```

The second type of statement defines the number of entities that exists in this world. In our example there are 400 bikes. In every probability problem, exactly one statement of this type should be specified.

4.2.3 Single attribute value with or without a count

```
type[mountainbike] = 80
type[hollandbike] = ?
```

In a probability problem there are multiple statements that define the number of entities that have a specific attribute value for the specified attribute. The first statement in the example above defines that in this world there are 80 mountain bikes. We will refer to this number as the 'count' of entities with this attribute value.

For every attribute value that is used in the probability problem, a statement of this type will be included in the probability problem. If a count is given, it should be greater than 0. If the count is zero, this attribute value is not used and therefore it should not be specified. If the count is replaced by a question mark, this indicates that this attribute value is used for at least one entity in the world. If the exact number is needed, the student has to derive this number by using the other statements. In our example for the type of bike 'holland bike' no count is specified. We will refer to this as a hidden count.

The attribute values that are used in these statements should be available in the context information. For every attribute value that is used in this probability problem, exactly one of the statements discussed in this section is included. The sum of the counts of all attribute values of the same attribute should be equal to the number of entities that is defined by the statement introduced in Section 4.2.2. In our example, the sum of all counts for the attribute values from attribute gears is 400, just as the total number of bikes available in this context.

All statements of the type we showed in the example in this section together define the attribute values that are used in this world. Those statements define thus the domain of the attribute.

4.2.4 The count for a combination of two attribute values

```
type[hollandbike] /\ gear[>3] = 96
```

The next type of statements includes a combination of two attribute values. This type of statements includes a count, the number of entities that have both attribute values. In this example, there are 96 bikes that are Holland bikes and have more than 3 gears. This type of statement always includes two different attributes, because an entity cannot have two different values for the same attribute.

Also, this type of statement always specifies a count. If no count is specified for a couple of two attribute values, this statement should not be included in the statements section of the probability problem.

4.2.5 Dependency information

```
dependent: type & gear
```

The last type of statement specifies the attributes that have a dependent relationship. As we have discussed in Section 4.1.3 we define any combination of two attributes (and not attribute values) as either dependent or indepent. We specify the attribute couples that have a dependent relationship. If a couple is not specified as being dependent, it should be considered as independent. The order of the attributes mentioned in this statement does not influence the meaning of the statement, but is reflected in the language.

In our example, it is expected that the number of gears depends on the type of bike and vice versa. It is more likely for a sports bike to have more gears than for a city bike. This information is not embedded in this dependency information, but can be defined indirectly by the statements that define a count for a combination of two attribute values.

There are some extra restrictions on the dependency as we discussed in Section 4.1.3. As a result, not every couple of two attributes can be dependent. Which attribute couples are dependent is defined in the context information. Furthermore, if the value of an attribute is dependent on another attribute's value, its value is independent from the value of any other attribute. In our example, the type of the bike and its number of gears are dependent. This excludes any dependencies between the type of bike with any other attribute.

We will see in Section 4.4 that some questions require attribute values from dependent attributes, others from independent attributes. This statement will therefore influence the attribute values that can be used in specific questions. Furthermore, we will also see that the generation of the counts for dependent attributes is different than the counts of independent attribute values. This is discussed in Section 4.4.4.

4.2.6 Questions

```
Q: P(~(type[sportsbike] /\ colour[white]))
```

Every question requires the student to calculate a certain probability. The structure of the question defines what probability has to be calculated. In this example, the question is to calculate the probability that a bike is not both a sports bike and white.

Grammar

The structure of all questions has the basic form $Q : P(A)$, where the capital letter can be extended using the following grammar:

1. $A \rightarrow X = x_i$ (where x_i is a specific attribute value from the attribute X)
2. $A \rightarrow \neg B$ (not B), represented by ascii character '$\sim$'
3. $A \rightarrow B \vee C$ (either B or C), represented by '\/'
4. $A \rightarrow B \wedge C$ (both B and C), represented by '/\'
5. $A \rightarrow B|C$ (a conditional probability, B given C)

In our example the structure of the question is a combination of grammar rules 2 and 4. This grammar suggests infinite possible types of question, but in the following section we will limit the possibilities by specifying the possible types of questions we consider in our research.

Answer

The answers to questions will be added to the textual representation of the probability problem as follows:

```
(answer: 17/20,calculations: D,B1)
```

If someone designs a probability problem manually, it is optional to specify the answer. However, if the system generates the probability problem, it will have to check whether the exercise is correct and whether the question can be solved given the information in the statements. As a by-product, the system will calculate the answer to the questions. If the system cannot calculate the answer, we may assume that the question cannot be solved by the student either. The required calculations to answer the questions will be discussed in Section 4.5.

4.3 Question types

The first system requirement, discussed in Section 1.3.2, specifies that it should be easy to generate new exercises. In this section, we introduce all types of questions that are possible in the exercises that can be created by Genpex. These so called question types are numbered and can be used to specify the type of questions that should be generated within the exercise.

4.3.1 Restrictions

In the previous section, we have already seen the structure of the questions that are embedded in the probability problem. There are many possible (combinations of) questions possible by using this grammar, but in this section we put some extra restrictions on the possible questions that are included in Genpex.

As we have listed in Section 1.3.1, there are some requirements on the (type of) exercises that should be created.

- Of course, the system should create correct exercises that can be answered

- The structure of the question is directly reflected in the language that is used. One question will result in one sentence and the order of the used attribute values is equal in both the question and sentence. If we do not have this restriction, it is possible that the system generates sentences that might help the student. If a question is split into two sentences, it might be clear to the student that it has to calculate the result of both sentences separately and combine the results.
- In every question, the attribute values that are used belong to at most two different attributes. If we use more than two attributes, there could be a complex dependency between the attributes. We want to prevent this kind of complex dependencies.
- Prevent recursion or questions that do not need a calculation to be answered
- Exclude questions that result in ambiguous language

Table 4.1 is a list with all possible questions that are not excluded by the previous restrictions.

Examples of questions that are excluded

Some questions are not included in the question types list, because the sentence that represents this structure is ambiguous. If we try to express the question

```
Q: P(~(colour[white]) /\ type[sportsbike])
```

in a English sentence, it will be phrased something like "What is the probability that a bike is not white and a sports bike?". This sentence can be misinterpreted: "What is the probability that a bike is not white and also *not* a sports bike?". The same ambiguity is found in the German sentence that represents this question.

The order of the attribute values in the statements defines the order of their representation in the text as we have discussed in Section 4.2. The following question, which is mathematical equivalent to the previous question, can be included as a question type, because this can be expressed without amiguity:

```
Q: P(type[sportsbike] /\ ~(colour[white]))
```

An example of recursion that we do not include in the question types is

```
Q: P(~ ~ ~ colour[white])
```

because the required calculation is equal to a question with only one negation[2].

Some questions such as

```
Q: P(colour[white] /\ colour[black])
```

express a situation that never occurs. A bike has never more than one colour, so the answer to this probability is always '0'.

[2]This question is also hard to express in a normal sentence: it asks for the probability that a bike is not not not white.

Another question that is never asked is the calculation of a conditional probability, where the attribute values are not dependent on each other. If colour and type are independent in the following example, the answer to the question

```
Q: P(colour[white] | type[mountainbike])
```

will be equal to the answer to the question

```
Q: P(colour[white])
```

that requires the calculation of the probability of a white bike. The values of the answers to these questions are equal, the extra information about the type of bike does not influence the value at all.

Correct representation of the world

Compared to the exercise we showed in the introduction in Figure 1.3.2, we put an extra restriction on the underlying probability problems. We will ensure that for every probability problem there exists a correct representation of the **world** on which this problem is based.

The world is an (internal) representation of a situation, where all entities are defined and for every attribute of every entity an attribute value is specified. Only a part of the world is reflected in the statements of the probability problem.

In this exercise, it is not possible to construct a correct and complete world. A probability problem can be specified in which a question can be answered, but that is not based on a correct world. We can give a (simple) example of such type of probability problem:

```
Context: bikes
numEntities: 10

colour[white] = 6
colour[black] = 4
gears[3] = 5
gears[5] = ?
gears[>5] = ?

Q: P(colour[white] /\ gears[3])
```

Based on this probability problem, it is possible to calculate the probability that a random bike is both white and has 3 gears[3]. It is however impossible to create a correct world and choose counts (which are integers) for $gears[3]$ and $gears[5]$ so that the gears and colour remain independent attributes: if colour and gears are independent, the ratio of white and black bikes with 5 gears should be equal to the ratio of white and black bikes that have more than 5 gears, and to the overall ratio of white and black bikes. It is impossible to choose numbers for these counts so that this restriction holds.

[3]Gears and colour are independent attributes, the answer is:
$P(colour[white]) * P(gears[3]) = \frac{6}{10} * \frac{5}{10} = \frac{3}{10}$

We want to overcome this problem. We create a correct world first and then fill in the counts in the statements using the information embedded in this world. Furthermore, we limit the number of hidden counts for an attribute value to one per attribute, as we showed in Section 4.2.3.

4.3.2 Overview of all question types

Respecting the restrictions, we systematically made a list of question types that should be available. In Table 4.1 we list all possible question types. In this figure, capital letters refer to attributes and small letters to their corresponding attribute values. For example, x_i and x_j are attribute values from attribute X. A question type does not only specify the structure of the question, but specifies also the information that is given in the statements section of the probability problem.

As we have seen in Section 4.2.3, the statements of a probability problem will generally contain the count for every single attribute value that is used in one of the questions. There are also question types that require some extra information. This is always a count for a combination of two attribute values and it is needed to be able to answer the question. The extra information that should be embedded in the statements section of the probability problem is specified the column *Extra information*. In the column *Dependency X Y* the restrictions on the dependencies of the attributes that are used in this question type are specified.

4.3.3 The naming convention of question types

The name of a question type is very important. Once the user of Genpex has specified the name of a question type in the exercise configuration, the system knows what type of question has to be generated. It will thus be easy to create correct new exercises, by which we fulfill our first system requirement.

The question types have structured names: The question types that express a question without negation are numbered 1 to 12. The other question types are based on the same question types, by the addition of one or more negations. If 'n0' is added to the name, the question is completely embedded in a negation. If the number that follows the 'n' is greater than 0, only a part of the question is embedded in a negation. All possible question types with negation are included in Table 4.1.

As we have seen in Section 4.2.3, some counts for single attribute values can be hidden from the statements. This is guided by the name of the question type. If we add the letter 'h' to the name of a question type, this indicates that the count of one of the attribute values that is used in this question is hidden in the statements. To hide a specific count, a number has to be added to the name. For example: '2h2' will hide the count for the second attribute value that is used in a question of type 2. If no number is specified, the count for a random attribute value is hidden from the statements. Every name of a question type shown in Table 4.1 can be extended with an 'h'.

Name	Template	Extra	Dependency X Y
1	$P(X = x_i)$	-	-
2	$P(X = x_i \vee x_j)$	-	-
3	$P(X = x_i \vee Y = y_j)$	-	independent
4	$P(X = x_i \vee Y = y_j)$	$x_i \wedge y_j$	dependent
5	$P(X = x_i \wedge Y = y_j)$	-	independent
6	$P(X = x_i \vee x_j \wedge Y = y_k)$	-	independent
7	$P(X = x_i \vee (X = x_j \wedge Y = y_k))$	-	independent
8	$P(X = x_i \vee x_j \wedge Y = y_k)$	$x_i \wedge y_k, x_j \wedge y_k$	dependent
9	$P(X = x_i \vee (X = x_j \wedge Y = y_k))$	$x_j \wedge y_k$	dependent
10	$P(X = x_i \mid Y = y_j)$	$x_i \wedge y_j$	dependent
11	$P(X = x_i \vee x_j \mid Y = y_k)$	$x_i \wedge y_k$, $x_j \wedge y_k$	dependent
12	$P(X = x_i \mid Y = y_j \vee y_k)$	$x_i \wedge y_j$, $x_i \wedge y_k$	dependent
1n0	$P(\neg(X = x_i))$	-	-
2n0	$P(\neg(X = x_i \vee x_j))$	-	-
3n0*	$P(\neg(X = x_i \vee Y = y_j))$	-	independent
3n1*	$P(X = x_i \vee \neg(Y = y_j))$	-	independent
4n0*	$P(\neg(X = x_i \vee Y = y_j))$	$x_i \wedge y_j$	dependent
5n0	$P(\neg(X = x_i \wedge Y = y_j))$	-	independent
5n1	$P(X = x_i \wedge \neg(Y = y_j))$	-	independent
5n2	$P(\neg(X = x_i) \wedge \neg(Y = y_j))$	-	independent
6n0	$P(\neg(X = x_i \vee x_j \wedge Y = y_k))$	-	independent
6n1	$P(\neg(X = x_i \vee x_j) \wedge \neg(Y = y_k))$	-	independent
7n0	$P(\neg(X = x_i \vee (X = x_j \wedge Y = y_k)))$	-	independent
8n0	$P(\neg(X = x_i \vee x_j \wedge Y = y_k))$	$x_i \wedge y_k, x_j \wedge y_k$	dependent
9n0	$P(\neg(X = x_i) \vee (X = x_j \wedge Y = y_k))$	$x_j \wedge y_k$	dependent
10n0	$P(\neg(X = x_i) \mid Y = y_j)$	$x_i \wedge y_j$	dependent
11n0	$P(\neg(X = x_i \vee x_j) \mid Y = y_k)$	$x_i \wedge y_k$, $x_j \wedge y_k$	dependent
12n0	$P(\neg(X = x_i \mid Y = y_j) \vee Y = y_k)$	$x_i \wedge y_j$, $x_i \wedge y_k$	dependent

Table 4.1: Question types. See Section 4.3.3 for the naming convention of the question types. The question types marked with * are excluded from the latest version of Genpex, because those questions will result in ambiguous sentences. This is discussed in the evaluation in Section 6.5.

4.4 Generating Probability Problems

In this section, we describe the process of the automatic generation of probability problems. It is difficult to create a probability problem completely by hand, because it should fulfill various restrictions. Genpex is therefore able to create probability problems automatically, based on one or more question types.

By using a question type as input for the generation of probability problems, the researcher can quickly create new probability problems. Because the probability problem is based on the question types, we automatically fulfill a number of requirements on the system and exercises, because we have only included question types that fulfill the exercise requirements in the question types list.

All restrictions and limitations on the generated exercises were discussed in Section 1.3. The following requirements on the exercises are not ensured via the question types and should therefore be reflected in the probability problem:

- If multiple questions are embedded in one exercise, the system should ensure that a student can not use the answer of a previous question to answer other questions.
- All counts that are used in the statements section of the probability problem should be different. Otherwise, it is possible that the student provides the correct answer by performing a calculation that is different than the intended one.

4.4.1 An overview of the generation process

The researcher specifies one or more question types and the context of the exercise, the system will select attributes and attribute values based on this information and it will format statements. In the next subtask the system will create a random probability distribution and it will ensure the correct dependencies between attribute values. The probability distributions will be converted in counts, that are added to the statements. The last task is to order the statements and format the probability problem.

An overview of all subtasks is shown in Figure 4.2. We will discuss the generation process by discussing each subtask of this module.

4.4.2 Input: context and question types

The input for the generation of a probability problem specifies the types of questions and a context. Via the graphical user interface, the user of Genpex specifies this information. The user gives the name of every question type that he wants to be included in the probability problem, using the naming convention we discussed in Section 4.3.3. Genpex will list the available contexts, and the user selects the one that should be used for this new probability problem. The context and question types can be chosen independently, it is also possible that the user first selects a context and then specifies the question types.

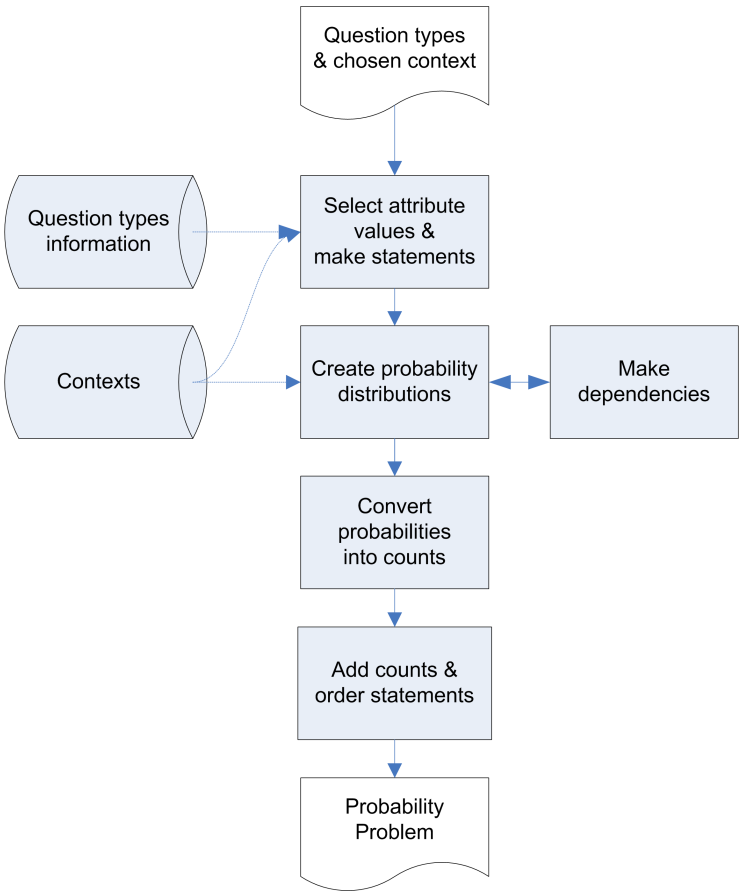


Figure 4.2: An overview of the process that creates a new probability problem, based on one or more question types and a context.

In this section, we will show an example of the creation of a new probability problem. For this example, we assume that the user wants to create a new probability problem and specifies the following information:

- The context that should be used is called 'bikes'
- There should be three questions embedded in this probability problem, of the question types: '10', '5h2' and '10n0'.

4.4.3 Select attribute values and make statements

Based on the given question types, the system knows the structure of the questions that should be embedded in the probability problem. The table shown in Table 4.1 is also embedded in the system. In this table, a template for this structure is given, but the attributes and attribute values that should be used in this question have to be chosen. In this structure, capital letters refer to attributes and lowercase letters to attribute values, where x_i is an attribute

value for attribute X. Based on the information in the question types table and the available context, this process chooses appropriate attributes and attribute values to be used in each question. To ensure that the student cannot use the answer of a previously answered question, we will choose different attribute values for each question. The chosen attribute values should also fulfill the given dependencies.

In this step, the program uses the 'Template' and 'Dependency'-column of Table 4.1. The example probability problem that we use to illustrate the generation process includes 3 question types:

- Question type '10': $P(X = x_i|Y = y_j)$, where X and Y are dependent attributes
- Question type '5h2'[4]: $P(X = x_i \wedge Y = y_j)$, where X and Y are independent attributes
- Question type '10n0': $P(\neg(X = x_i)|Y = y_j)$, where X and Y are dependent attributes

For our example probability problem, we use the 'bikes' context. Assume that the following attributes and attribute values are available in this context:

- gears: 3, 5, >5
- type: hollandbike, mountainbike, sportsbike, citybike
- colour: black, silver, white, green, blue, orange

Furthermore, in this context the number of gears and the type of bike are dependent, other couples of attributes are independent.

The attribute value selection process will perform the following algorithm to choose attribute values for the empty slots in the question templates:

1. Select a dependent or independent attribute couple, based on the dependency restriction of the question type
2. Determine the number of different attribute values that is necessary, by counting the number of 'x'-s and 'y'-s in the question template
3. Select enough attribute values that have not been used in a previous question.
    - If no unused attribute values are available, go to step 1 and select a different attribute couple
    - If there are no unused attribute values available in any of the suitable attribute couples, the user is notified

For our first question, we need a dependent attribute couple, and from both attributes one attribute value. In the context the attributes 'gear' and 'type' are dependent, so the system will select from both attributes one value: for example 'gear[3]' and 'type[mountainbike]'. For the second question, two attribute values will be chosen from two independent attributes. If the system selects

[4] As we have discussed in Section 4.3.3, we can use the same information as is being used for question type 5

```
Q: P(gear[3] | type[mountainbike])
Q: P(gear[5] /\ colour[white])
Q: P((~gear[>5]) | type[sportsbike])
```

Figure 4.3: Questions, based on the templates derived from Table 4.1.

the attribute couple 'gear' and 'colour', the attribute value 'gear[3]' will not be chosen, because it has been used in the first question.

Now we replace the blanks in the questions by using the chosen attribute values. The questions that are created are shown in Figure 4.3.

Once we have selected attribute values and designed the questions that are embedded in the probability problem, we should create the statements of the probability problem. For every attribute value that is used, we should create a statement that gives the count for this attribute value. Because this count is still unknown, we will add a statement without a count and add the count later. An overview of all created statements for our example is given in Figure 4.4.

Adding more attribute values

If for an attribute less than three attribute values are used, the system will add more attribute values from this attribute to the statements. As we will show in the language generation process in Chapter 5, we have more possibilities to express three attribute values from the same attribute than one or two. In our example, an extra statement is added for the attribute 'type' (attribute value 'hollandbike') and two extra statements for the attribute colour ('black' and 'silver').

Extra information and hidden counts

Normally, for all attribute values used in a question, a statement is given that defines the count for this attribute value. Some question types require extra information to be added to the statements section of the probability problem. We should add extra information for the first and last question of our example (question type '10' and '10n0'). This information is required to be able to calculate the correct answer to the question. Both combined statements are added to the statements without count, as is shown in Figure 4.4.

The count of one of the attribute values used in the second question should be hidden, as is indicated by the name of the question type. The addition of 'h2' indicates that the second attribute value has to be hidden. As we can see in Figure 4.3 this attribute value is 'colour[white]'. In Figure 4.4, this statement is indicated by a question mark.

```
gear[3] = ...
type[mountainbike] = ...
gear[5] = ...
colour[white] = ?
gear[>5] = ...
type[sportsbike] = ...
type[hollandbike] = ...
colour[black] = ...
colour[silver] = ...
gear[3] /\ type[mountainbike] = ...
gear[>5] /\ type[sportsbike] = ...
```

Figure 4.4: The statements that should be embedded in the probability problem. The counts will be added later. The count for 'colour[white]' will be hidden.

4.4.4 Generation of counts

In this subsection, we sketch the algorithm for generating counts for the different statements. Globally, this algorithm is split into the following tasks:

1. Create probability distributions for the attribute values per attribute
2. Create matrices per attribute couple, with the probability for every combination of attribute value
3. Ensure dependency for dependent attribute couples
4. Convert probabilities into counts

We will discuss every task in more detail and provide an example for each step. In the first step, probability distributions are chosen randomly (given some restrictions), and also in the third step dependencies are created randomly. Multiple runs of this algorithm with the same input parameters will therefore create different counts.

Task 1: Creating a distribution of probabilities for all attribute values

For each attribute, the system should create a distribution of probabilities for all attribute values that have been used in the probability problem. This probability distribution specifies the probability that one entity has a specific value for the attribute.

There are some restrictions on these distributions:

1. The sum of all probabilities in the distribution is 1
2. None of the probabilities is zero
3. All probabilities are different

If we have to create probabilities for K attribute values, we draw N times from a multinomial distribution. The number K is an integer. N specifies the

total number of objects that are put into K boxes in the typical multinomial experiment.

While K is known (the number of different attribute values), N is chosen arbitrarily. We want to ensure that the smallest possible probability that is created is $\frac{1}{N}$, because the probability of a combination of two attribute values is the muliplication of both probabilities. The smallest probability for a combination of two attribute values is therefore $\frac{1}{N} \times \frac{1}{N} = \frac{1}{N^2}$. To ensure that the count for all combinations will be integers, we define that the total number of entities is N^2. Because every fraction is multiplied by the total number of entities to calculate the count, and the smallest value is $\frac{1}{N^2}$, every multiplication will result in an integer. When we define N^2 to be the total number of entities, N is defined as the root of the total number of entities. We round N for aesthetical reasons so that it can be divided by 5, so that the counts of statements will be rounded too.[5]

We define N as a random number between the root of the minimum number of entities that is allowed in this context and the root of the maximum number of entities that is allowed. This number is rounded to the nearest number that can be divided by 5.

We give an example of this algorithm for generating a probability distribution for the attribute colour:

- We have the three values for the attribute 'colour' : 'black', 'silver' and 'white', so $K = 3$
- In the context, we can find that there should be at least 50 and maximum 500 bikes. The program chooses $N = 20$.[6]
- If we draw from the multinomial distribution [7] we can get the probabilities: $\frac{3}{20}$ for 'black', $\frac{8}{20} = \frac{2}{5}$ for 'silver' and $\frac{9}{20}$ for 'white'.

By using the same algorithm, we can make a distribution for the type of the bike (for example $\frac{7}{40}$ for 'mountainbike', $\frac{1}{8}$ for 'sportsbike' and $\frac{7}{10}$ for 'hollandbike') and for the number of gears (for example $\frac{1}{2}$ for '3', $\frac{3}{10}$ for '5' and $\frac{1}{5}$ for '>5').

Task 2: Matrices with probabilities for all combinations of attributes

For every couple of attributes we make a matrix with the probability distributions of the attribute values. Each cell in this matrix represents the probability of the joint occurrence of two attribute values. If the attribute values are independent, this is calculated by multiplying the probabilities of both attribute values. We perform this multiplication also for dependent attribute values, in the next step the dependency is created.

We show two matrices: for the attributes 'gear' and 'colour' in Table 4.2 and for 'gear' and 'type' in Table 4.3. The matrix for the combination of 'colour' and 'type' has to be created similarly, but we will not show it here.

[5] Found by experience.

[6] N should have a value between $\sqrt{50} \approx 7.07$ and $\sqrt{500} \approx 22.36$, randomly chosen, rounded to the nearest number that can be devided by 5.

[7] A multinomial distribution with parameters: $n = 1$, $size = 20$, $prob = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$

As we discussed, each cell is filled by multiplying the probabilities of two attribute values: for the first cell in Table 4.2, we multiply the probability for 'colour[black]' with the probability for 'gear[3]', $\frac{3}{20} \times \frac{1}{2} = \frac{3}{40}$, for the second cell $\frac{2}{5} \times \frac{1}{2} = \frac{1}{5}$ etc.

	colour[black]	colour[silver]	colour[white]	distr.
gear[3]	3/40	1/5	9/40	1/2
gear[5]	9/200	3/25	27/200	3/10
gear[>5]	3/100	2/25	9/100	1/5
distr.	3/20	2/5	9/20	1

Table 4.2: The distribution of the attributes 'gear' and 'colour'

	type[mountainbike]	type[sportsbike]	type[hollandbike]	distr.
gear[3]	7/80	1/16	7/20	1/2
gear[5]	21/400	3/80	21/100	3/10
gear[>5]	7/200	1/40	7/50	1/5
distr.	7/40	1/8	7/10	1

Table 4.3: The independent distribution of the attributes 'gear' and 'type'

Task 3: Make dependencies

By multiplying the probabilities of the attribute values, we created matrices where the attribute values are independent from each other. To make two attributes dependent, we should modify the values of some cells in the created matrix. We cannot randomly change values, because the distributions for the attribute values themselves should not change. We should therefore not change the sum of any row or column.

The following algorithm will change some values, without affecting the sum of rows and columns:

- We choose for every attribute two values and select the matching cells: we select 4 cells
- We randomly choose a probability that is smaller than the smallest probability in those 4 cells
- We subtract this probability from the values in one diagonal, and add it to the values in the other diagonal

This algorithm should be applied to every matrix of attribute couples that should be dependent. In our example, the attributes 'gears' and 'type' (given in Table 4.3) should be dependent. We have made the attributes 'gears' and 'type' dependent using the algorithm described above, which we show in Table 4.4.

Task 4: Convert probabilities into counts

Now that we have the probability of the occurrence of every attribute value and attribute value, we can fill the statements with this information. But we

	type[mountainbike]	type[sportsbike]	type[hollandbike]	distr.
gears[3]	7/80 (21/400+1/100=)	1/16 (3/80-1/100=)	7/20	1/2
gears[5]	1/16 (7/200-1/100=)	11/400 (1/40+1/100=)	21/100	3/10
gears[>5]	1/40	7/200	7/50	1/5
distr.	7/40	1/8	7/10	1

Table 4.4: The distribution of the attributes 'gears' and 'type', the changes to the probabilities given in Table 4.3 make those attributes dependent.

	colour[black]	colour[silver]	colour[white]	distr.
gears[3]	30	80	90	200
gears[5]	18	48	54	120
gears[>5]	12	32	36	80
distr.	60	160	180	400

Table 4.5: The distribution of the attributes 'gears' and 'colour'

should first convert the probabilities into counts. We can simply make a count by multiplying the probability by the total number of entities available in this world. A count is always an integer. Therefore, we should choose the number of entities in the world carefully to ensure this. We ensure that all numbers are integers by using the Least Common Denominator of all fractions in all matrices. In our example the Least Common Denominator of all fractions is 400, and therefore the total number of entities is 400.

We multiply all fractions in the matrices created before by 400, which results in Tables 4.5 and 4.6. The integers in these tables can be used in the statements section of the probability problem.

4.4.5 Add counts and order statements

Now that we have calculated all possible counts, we can use those counts in the constructed statements. The count for every statement (that does not have a hidden count) is derived from the correct matrix.

The statements will be ordered, using the following algorithm:

- For each attribute, we group all statements having a value from that attribute

	type[mountainbike]	type[sportsbike]	type[hollandbike]	distr.
gears[3]	35	25	140	200
gears[5]	25	11	84	120
gears[>5]	10	14	56	80
distr.	70	50	280	400

Table 4.6: The distribution of the dependent attributes 'gears' and 'type'

```
Context: bikes

numEntities: 400

gear[3] = 200
gear[5] = 120
gear[>5] = 80
type[mountainbike] = 70
type[sportsbike] = 50
type[hollandbike] = 280
colour[black] = 60
colour[silver] = 160
colour[white] = ?

gear[3] /\ type[mountainbike] = 35
gear[>5] /\ type[sportsbike] = 14

dependentAttributes: type&gear

Q: P(gear[3] | type[mountainbike])
Q: P(gear[5] /\ colour[white])
Q: P((~gear[>5]) | type[sportsbike])
```

Figure 4.5: The created probability problem, based on the context 'bikes' and question types '10', '5h2' and '10n0'.

  - The groups are randomly ordered
  - If the count of one of the statements in a group is hidden, it will be used as last, the other statements are randomly ordered

- The statements that give the count for a combination of two attribute values are placed after those discussing a single attribute value.
- The statements that give the chosen context and the total number of entities are placed at the top of the statements.
- A statement that specifies the attributes that are dependent is placed at the end

We have now determined and ordered all statements. The questions have also been defined, as we have seen in Figure 4.3. Together they form the probability problem. The textual representation of the probability problem that we created is shown in Figure 4.5.

4.5 Calculating the answer to a question

During the creation of a probability problem, the complete world is defined by matrices. The answer to the question in the probability problem can be found in one of these matrices. However, we also want to be able to calculate

the answers to probability problems that were not created by the program but specified by hand by the user. Therefore, we should not use the information from the creation process. Only the information that is embedded in the probability problem should be used to answer the questions.

4.5.1 Terminology and basic calculations

All questions embedded in the probability problem require the calculation of a probability for an event. As we discussed in Section 4.1.3, the range of possible events in our exercises is limited. A question $P(A)$ asks for the probability of **event** A. This event can be described as "Someone draws one entity out of a (sub)set of entities and this entity has property A.". In general, this set of entities is the entire set of entities in the defined world. It can be limited by a conditional statement. The event $A|B$ can be described as "Someone draws one entity with the property A out of a subset of entities that have B". As we discussed in Section 4.1.3 all events are limited to a single draw of exactly one entity.

If our entities are bikes and A is the property that the colour is black, the question is to calculate the probability that a bike is black if we take one random bike from this world. We assume that we know the total number of bikes and the number of bikes that is black. Counts are used in the statements, but the questions ask for a probability. We can calculate the probability by

$$P_E(A) = \frac{\#E_A}{\#E}$$

where $\#E_A$ is the number of entities for which property A holds and $\#E$ is the total number of entities. In our example: $P(colour[black])$ is calculated by dividing the number of bikes that are black by the total number of bikes. In our example that probability is $310/840 = 1/4$.

In the statement $A \vee B$, A and B are **disjoint** if A and B can never apply simultaneously to one entity. A and B are disjoint if

- A and B are both attribute values from the same attribute
  For example: $A = colour[white]$ *and* $B = colour[black]$

- A is a single attribute value from attribute X, and B is a combination of two attribute values where one of those attribute values is also from type X
  For example $A = colour[white]$ *and* $B = (colour[black] \wedge gears[5])$*, because* $colour[white]$ *and* $colour[black]$ *exclude each other*

- Notice that two attributes do not need to be disjoint in the previous rule if there is a negation
  For example $A = colour[white]$ *and* $B = \neg(colour[black] \wedge gears[5])$ *are not disjoint, because both statements could be applied to an entity that is white and has 5 gears*)

A:	$P(x_i) = 1 - \sum_{\forall j, j \neq i} P(x_j)$	*(if x_i is hidden)*
B1:	$P(\neg A) = 1 - P(A)$	
B2:	$P(\neg x_i) = \sum_{\forall j, j \neq i} P(x_j)$	*(if x_i is hidden)*
C1:	$P(A \vee B) = P(A) + P(B)$	*(if A and B are disjoint, XOR (exclusive or))*
C2:	$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$	*(else, A and B not disjoint, OR)*
D:	$P(A \wedge B) = P(A) * P(B)$	*(if A and B are independent)*
E1:	$P(A\|B) = P(A \wedge B)/P(B)$	*(if A and B are dependent)*
E2:	$P(A\|B) = P(A)$	*(else, A and B independent)*
F:	$P(A \wedge (B \vee C)) = P((A \wedge B) \vee (A \wedge C))$	
G:	$P(\neg\neg A) = P(A)$	*(Involution)*
H:	$P(A \wedge \neg B) = P(A) - P(A \wedge B)$	
I1:	$P(\neg A \wedge \neg B) = P(\neg(A \vee B))$	
I2:	$P(\neg(A \vee B)) = P(\neg A \wedge \neg B)$	*(De Morgan)*
J1:	$P(\neg A \vee \neg B) = P(\neg(A \wedge B))$	
J2:	$P(\neg(A \wedge B)) = P(\neg A \vee \neg B)$	*(De Morgan)*
K:	$P(A\|B) = (P(B\|A) * P(A))/P(B)$	*(Bayes' theorem)*

Table 4.7: A list of all calculations that Genpex can perform. We can refer to a calculation rule by a letter, some rules have two similar variations, indicated by the addition of number 1 or 2.

4.5.2 Calculation rules

To answer the questions using the information in a probability problem, our program will perform the same actions as a student will perform to answer the questions. If the number of entities that is described by the event is given, the basic calculation described in the previous section is used. Otherwise, we have to apply (a combination) of rules to be able to answer the question. We have added rules for the calculation of probabilities for different types of questions. General rules that can be used to calculate probabilities are discussed (among others) by Russell and Norvig [RN03] (p. 470-480). We have not implemented all rules, but only those listed in Table 4.7. In this table we refer to events by capital letters. We use small letters with an index (for example x_i) to refer to an event that describes an entity of which the attribute value for a single attribute is specified.

By using (a combination of) the rules in this table, it is possible to answer at least all question types that we showed in Table 4.1.

Genpex will calculate an answer for all questions using the following algorithm:

- Check whether the probability for this event has been calculated before, if so: return this saved value
- If not, check whether a count is specified for this event, if so: divide this count through the total number of entities in the world to derive the probability of this event

- Iterate through all rules listed in Table 4.7, if the left side of the statement in a rule matches to the current event, use this rule
  - The execution of a rule will result in one or more new events
  - Perform this algorithm recursively for each of these new events
  - If for every event a probability is found, apply the calculation defined in the rule
  - If the algorithm did not found a probability for every event, continue with the iteration of all rules and try the next rule that matches
- If all rules were tried but no answer is found, throw an exception

If the answer to a question cannot be answered by this algorithm, an exception is thrown. It can be used to notify the user. This algorithm does not garantuee that all questions can be answered, but it is possible to answer the questions that are generated using the question types.

As soon as Genpex calculates an answer, it does not only provide the user with the answer, but it provides the user with the calculations that were performed to calculate the answer too. In Figure 4.6 we show the calculations that were necessary to answer the questions in the probability problem from Figure 4.5. The first question was answered by performing a single calculation of type 'E1'. The second and third questions required two calculations. For the second question, first a calculation of type 'A' had to be performed, the answer of that calculation was used in the calculation of type 'D'.

As we have shown in Section 4.2.6, both the answer and the names of the calculations that need to be performed to calculate this answer can be added to the probability problem. This information can be used in the analysis of field trials performed with the exercises that were created using this probability problem.

```
Q: P(gear[3] | type[mountainbike]) = 1/2

   P(gear[3] /\ type[mountainbike]) / P(type[mountainbike]) =
   7/80 / 7/40 = 1/2 (calculation type: E1)

Q: P(gear[5] /\ colour[white]) = 27/200

         P(colour[white]) =
   1-(P(colour[black])+P(colour[silver])) =
   1-(3/20+2/5) = 9/20 (calculation type: A)

   P(gear[5] /\ colour[white]) =
   P(gear[5]) * P(colour[white]) =
   3/10 * 9/20 = 27/200 (calculation type: D)

Q: P((~gear[>5]) | type[sportsbike]) = 18/25

   P((~gear[>5]) /\ type[sportsbike]) =
   P(type[sportsbike]) - P(type[sportsbike] /\ gear[>5]) =
   9/100 (calculation type: H)

   P((~gear[>5]) | type[sportsbike]) =
   P((~gear[>5]) /\ type[sportsbike]) / P(type[sportsbike]) =
   9/100 / 1/8 = 18/25 (calculation type: E1)
```

Figure 4.6: An overview of the calculations that are performed to answer the questions embedded in the probability problem shown in Figure 4.5.

Chapter 5

Natural Language Generation in Genpex

The most important output of Genpex is the text of an exercise. The mathematical contents of an exercise are completely based on the probability problem and therefore the text of the exercise is a representation in natural language of this probability problem. In this chapter we will discuss the Natural Language Generation (NLG) process in Genpex that results in this story.

During this process, variation is added to the text of the exercise. Multiple runs of the language generation process on the same probability problem result in different texts. While these texts are different, mathematical meaning of the exercises is equal. If someone attempts to determine the underlying probability problem of a text, the result will be the same for all texts that are based on the same probability problem.

The student that makes the exercise is trying to derive the probability problem (at least the part that is necessary to answer the questions) from the text. It is very important that the generated language is not ambiguous. Variations between two texts that are based on the same probability problem, should not influence the difficulty of understanding the mathematical meaning of the text.

The NLG-process of Genpex is responsible for two tasks:

1. Generating a correct textual representation of the probability problems
2. Adding variation to texts, so that multiple runs of the NLG-process will result in different texts

As we will see, those two tasks are closely related to each other. Many other NLG-systems such as the Narrator (see Section 2.2), the ModelCreator software (Section 2.1.1) and WebALT (Section WebALT) try to create the best textual representation, by determining in each situation the best possible set of sentences. In the same situation, systems will always produce the same text. In Genpex, we consider several options and randomly choose one of the options (this will be discussed in Section 5.4). Every run of the NLG-process, the system can 'choose' a different representation, resulting in different texts.

As said, the text of the exercise is based on a probability problem. Besides the mathematical contents of the exercise, this probability problem guides some other language specific aspects: in the probability problem, not only the context is chosen, also the number and ordering of the attributes and attribute values that are used is defined. Also, a statement can include an attribute value and a count. This count has influence on the morphology of the text, because the inflections of words depend on this count. This count determines whether the words are singular or plural.

For the generation of the language, we make use of a pipeline-architecture, as many other NLG-systems. In this chapter we will first give a global overview of this pipeline, and discuss all three subtasks in this pipeline. We will also discuss the internal representation of sentences that are used to exchange information between the subtasks. Finally we discuss the resources that are used in the NLG subtasks.

5.1 Natural Language Generation pipeline

As we have discussed in Section 2.2.1, the architecture of many NLG-systems divides the text generation into three modules: Document Planning, Microplanning and Surface Realization [RD97]. These modules are connected in a pipeline structure: the output of the document planning module is used as input for the micro planning, the output of the micro planning as input for the surface realization. Due to the pipeline architecture of the system, there are some limitations. It is not possible for the micro planning part to influence the behavior of the document planner, and for the surface realizer it is not possible to change the behavior of the micro planning. This architecture simplifies the process and is therefore used in many NLG systems, such as the Narrator discussed in Section 2.2.

Generally speaking, the three modules are responsible for the following tasks:

- Document planning: determining the content and structure of the text.
- Micro planning: determining the words and syntactic structures that will be used to express the structure chosen in the document planner. This module is generally also responsible for the aggregation of sentences.
- Surface realization: mapping the abstract representation of objects into actual text. The inflection of words is performed here, based on the grammatical structure.

Different NLG systems vary in the exact definition of the tasks of each module and there are also differences in the location of these tasks. Some tasks, such as the process of lexicalization (choosing the correct word form and inflection of the word), are performed in one system in the micro planning module, where other systems perform that task in the surface realization module.

5.1.1 The NLG pipeline in Genpex

A global overview of the NLG-process in Genpex is given in Figure 5.1. Information between the modules in the NLG process is exchanged by (a list of) **sentence trees**. Every sentence tree represents a sentence and defines the contents and grammatical structure of it. We will discuss sentence trees in more detail in the next section, Section 5.2. In the **document planner**, basic sentence trees are constructed, based on the order of statements in the probability problem.

The generated sentence trees will be passed on to the **microplanner** that is responsible for making more sophisticated sentences. In this module, variation can arise: we distinguish a couple of techniques that can be applied to sentence trees, to change the basic sentence. A technique is applied with a certain probability. The whole microplanner can be skipped, but that will result in very monotonous sentences. The output of the microplanner - the changed sentence trees - is the input for the **surface realizer**. This last module applies the correct morphology and is responsible for the correct layout of the text.

For further analysis of the generated exercises, it is important to know what language techniques have been applied on the text. This information is saved together with the generated exercise. During the generation process, the techniques that could have been applied and the techniques that have been applied are saved in a log file. This log is saved together with the input and output of the system.

5.2 Sentence trees

A sentence tree is a structural representation of the information in a sentence. This information includes both the meaning and the linguistic structure of the sentence. All words and phrases in a sentence are represented by tree nodes, all tree nodes together represent a sentence in a tree structure. The information embedded in the sentence tree is used to generate correct sentences, but can also be used to add variation to the sentences.

We use sentence trees to exchange information between the different modules in the NLG-process of Genpex. The document planner creates sentence trees based on the probability problem, the micro planner makes changes to those sentences trees and finally the surface realizer converts the sentence trees to natural language.

In this section we will discuss the structure and the use of sentence trees.

5.2.1 Analysis of example sentences

We have analyzed the structure of existing sentences from the exercises that were created by the University of Münster. To investigate the structure of the sentences that we are going to generate, it is useful to parse existing sentences by using a syntactic parser. We have used two parsers that can parse a sentence

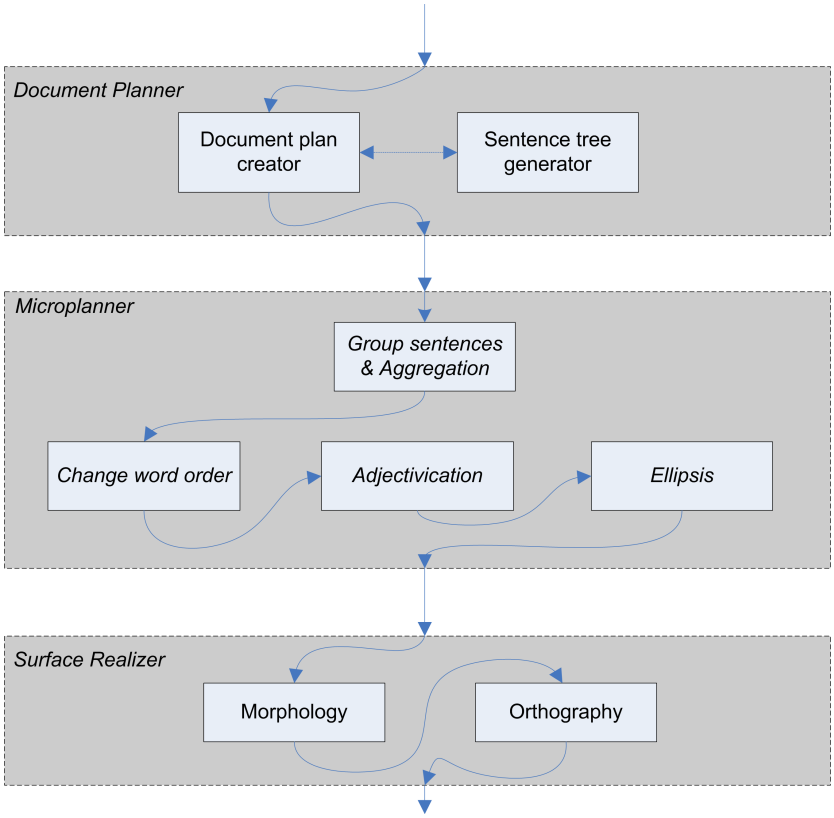


Figure 5.1: The connection between the modules in the NLG part of the system.

and show its structure: the Stanford parser[1] [KM02] and the Alpino parser[2] [BNM01]. The Alpino parser can only parse Dutch sentences, so we have used the literal translation of the German sentences. The Stanford parser makes use of the NEGRA corpus and can analyze German sentences.

Although the Alpino parser can only parse sentences translated into Dutch, it gave us useful information about the structure of sentences. The danger of parsing translated sentences is that those sentences can have a different structure compared to the original German sentences. We still use the Alpino parser, because it has been used in the research for the Narrator of the Virtual Storyteller that we discussed in Section 2.2, and we are already familiar with Alpino. During our analysis, we carefully considered potential errors due to this translation, and verified our findings with the German Stanford parser.

The sentence trees that we discuss in this chapter are based on our analysis of existing sentences.

5.2.2 Information embedded in a sentence tree

As we have discussed in Chapter 2, in many NLG systems it is a common approach to embed structured information in a sentence tree. The structure, contents and the use of sentence trees is different in all NLG systems and it is therefore important to carefully describe the sentence trees that we use in Genpex in detail.

A sentence tree is a tree with so-called **tree nodes** containing information about a sentence. Every tree node can have zero or more child-nodes. A complete sentence is encoded as a sentence tree, every tree node contains information about a word or phrase in that sentence. The structure of the tree describes the structure of the sentence. A graphical representation of a sentence tree is shown in Figure 5.2.

Each node in a sentence tree might contain several types of information. We list all types of information, and the possible values for each type in Figure 5.3. Every node has a syntactical category that defines its syntactical type such as noun or verb. The top-node (the root) of a sentences tree has the syntactical category 'sentence', all other tree nodes represent the phrases and words of the sentence. A word or phrase also has a grammatical role, indicating the role it has in a sentence, for example the subject or direct object.

Nodes are either lexical nodes or have one or more child nodes. The top-node with the syntactical category 'sentence' has multiple child nodes, each child node can have child nodes too. Only the leaf nodes in the tree have words (or lexical values), indicating the basic form of the words that should be used in the sentence. Information about the specific inflections is defined by the other properties of the tree node. Words are inflected in the surface realizer and will be discussed in more detail in Section 5.5.

A sentence tree that represents a phrase has child-nodes, representing the different words in that phrase. The order of the nodes in the sentence tree defines

[1] http://nlp.stanford.edu/software/lex-parser.shtml
[2] http://www.let.rug.nl/vannoord/alp/Alpino/

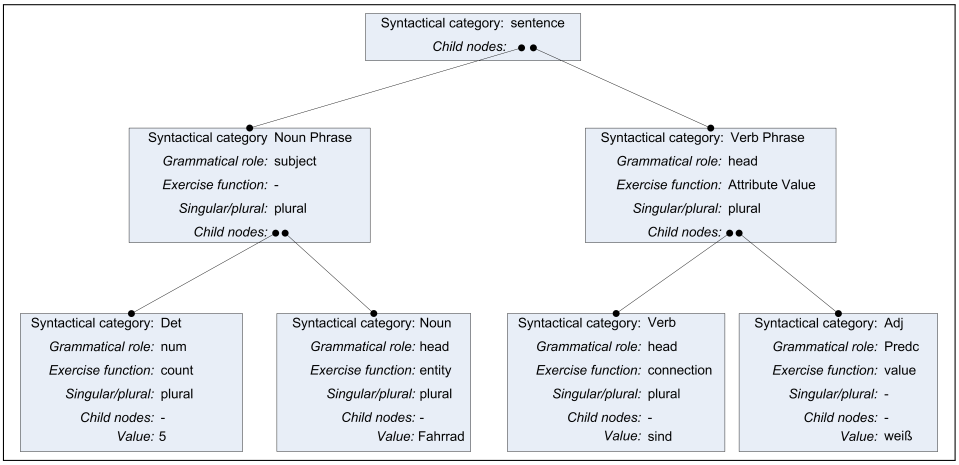


Figure 5.2: An example of a graphical representation of a sentence tree representing the sentence "5 Fahrräder sind weiß". Translation: "5 bikes are white".

- Syntactical category
  - Sentence
  - Noun Phrase (np)
  - Verb Phrase (vp)
  - Prepositional Phrase (pp)
  - Noun (noun)
  - Pronoun (pron)
  - Verb (verb)
  - Adjective (adj)
  - Determiner (det)
- Grammatical role
  - Subject (su)
  - Head (hd)
  - Direct Object (obj1)
  - Modifier (mod)
  - Determiner (det)
  - Prepositional Complement(pc)
  - Verbal Complement (vc)
- Function within the exercise
  - Entity
  - Count
  - Attribute Value
  - Connection
- Singular/plural
- Child nodes
- A word, the lexical value of the node (i.e. the singular form of a noun, the stem for a verb)

Figure 5.3: An overview of all types of information that can be embedded in a tree node. The syntactical category is obligatory, the other types of information are optional. We give the possible options for every type of information and the abbriviation that is used in the textual representation of the nodes.

the order of the words in the sentence that is represented by this tree. If we make changes to the order in the tree, this will change the word order in the corresponding sentence. This is a contrast to other NLG systems, suchs as the Narrator in the Virtual Story Teller discussed in Section 2.2. The word order is in those system determined in the surface realizer. In our system the micro planner, discussed in Section 5.4, can add variation by changing the word order in the sentence.

Sentence trees and tree nodes can be graphically represented as is shown in Figure 5.2, but internally a Java-object is defined that represents the structure of a sentence tree. This object can also be represented in a textual form, as is shown in Figure 5.4. This textual representation can easily be parsed and is therefore used to save and reopen generated sentence trees. This textual representation of sentence trees will be used to give examples of sentence trees in the remainder of this document.

```
[s]
        [np grammaticalRole=su]
                [det grammaticalRole=num]5[/det]
                [noun grammaticalRole=hd]Fahrrad[/noun]
        [/np]
        [vp]
                [verb grammaticalRole=hd]sind[/verb]
                [adj grammaticalRole=predc]weiß[/adj]
        [/vp]
[/s]
```

Figure 5.4: The textual representation of a sentence tree representing the sentence “5 Fahrräder sind weiß”. Translation: “5 bikes are white”.

5.2.3 Canned Text: simple representation of a sentence in a single node

It is possible to represent a whole sentence or phrase with a single tree node. This sentence or phrase is specified in the ‘word’ part of the Tree Node, and except for the syntactical category, all other properties are undefined. Now we can use ‘canned text’, text that is not processed by the NLG-subsystems, but is copied in the resulting exercise text.

An example of canned text is the introduction in the exercise that is not generated but specified in the context information. It does not have to be processed and therefore it is not necessary to be represented as a correct grammatical structure. This prevents a lot of unnecessary work. By representing a whole sentence as a sentence tree with a single tree node, it could be passed through the entire system just as generated sentences. This tree node is not changed by the microplanner, because of its structure.

5.3 Document Planner

The most important tasks of the document planner in most language generation systems are the content determination and structuring of the text. However, in our system this task is relatively easy, because the input for this module, the probability problem, already contains this information. The contents of the exercise are defined by the contents of the statements and the structure of the exercise is defined by the structure of the probability problem: the order of the statements and questions in the probability problem. The context as been used in the probability problem defines the words that should be used in the sentences that express the statements.

As we have discussed, the information between the modules in the natural language pipeline share their information in sentence trees. The output of the document planner is a document plan, which is a structured list of sentence trees. This document plan has a introduction, a list of statements expressed as sentence trees, and one or more questions, also represented as a list of sentence trees. The document planner is responsible for converting the statements and questions embedded in the probability problem into sentence trees. We will discuss this process for the statements and the questions separately. We will not discuss the creation of the introduction of the exercise: it is a simple canned-text specified in the context-file. If more than one introduction is specified in the context-information one is randomly chosen, but this is a very simple process.

The sentences included in this document plan are all very simple, and have a basic structure. Various techniques in the micro planner change these basic structures in more sophisticated sentence structures. This basic structures as we introduce in the following sections are derived from our research of existing exercises, as discussed in Section 5.2.1.

5.3.1 The construction of sentences trees for statements

We take for example the statement '*colour[white] = 10*'. The first step in the construction of sentence trees is the selection of the nodes that have to be included in the sentence tree. For the previous statement, we can find those nodes in the 'bikes' context:

- *[noun]Fahrrad[/noun]*, the representation of the entity
- *[adj]weiß[/adj]*, the representation of the attribute value *colour[white]*
- *[verb]sind[/verb]*, the connection between the entity and the attribute value
- *[det grammaticalRole=num]10[/det]*, the count

A new noun phrase node is constructed, by using the noun that represents the entity and the count. The grammatical role of this noun is subject.

As we have discussed, our sentence trees also define the word order of the sentence. This is different compared to other NLG-systems, but allows us to change the word order by manipulating the order of the nodes in the tree. Now that

we are constructing a new sentence tree, we should add the nodes in the right order to this sentence tree.

There is a default word order that is based on the grammatical role and syntactical category of the nodes. All sentences start with a subject, followed by a verb or verb phrase. The default word order of the rest of the sentence in the German language is: pronoun, noun phrase, noun, adjectives, prepositional phrase followed by a clause[3]. If we add the previous selected nodes in this order to a new sentence tree, we get the following tree:

```
[s]
[np grammaticalRole=su]
[det grammaticalRole=num exerciseFunction=count]10[/det]
[noun grammaticalRole=hd exerciseFunction=entity plural]Fahrrad[/noun]
[/np]
[verb grammaticalRole=hd exerciseFunction=connection plural]sind[/verb]
[adj grammaticalRole=predc exerciseFunction=value]weiß[/adj]
[/s]
```

The words in this sentence tree have a certain order that can be changed in the microplanner.

5.3.2 The construction of sentences trees for questions

If we construct a sentence tree for a question, the first two child nodes are the same for every question. All questions start with the same phrase: 'Wie groß ist die Wahrscheinlichkeit dass'. This phrase is added as 'canned text', as a node with syntactical category 'clause'. The child node that follows this clause is the subject of the sentence, and a reference to the entities discussed in the exercise. This reference is made by a noun phrase, that is a combination of the words 'ein' and the noun that expresses the entity. Because the information is embedded in a sentence tree, the words will be correctly inflected in the surface realizer.

While the first part of the sentence is equal for every question, the second part is based on the structure of the question. The reader of the sentence should be able to understand the structure. As we have discussed, the sentence may not be ambiguous.

To ensure that the structure of the question is represented in the sentence, we use templates for every possible structure. Templates can be used recursively: the capital letters in those templates can be filled in with the basic structure for an attribute value, or with one of the other templates. In Figure 5.5 we show the construction of a sentence tree for a question, using multiple question templates.

If an attribute value has to be added to the sentence tree, it is added as a verb phrase. This representation is similar to the representation of an attribute value in the statements: in the verb phrase, the value of the attribute is added and the verb connecting the value to the entity. The word order in this verb phrase

[3]Reference: http://www.canoo.net

depends on the location of this verb phrase in the sentence: if this verb phrase is used before the word 'vorausgesetzt[4]' (or in a question without that word), the verb is place at the end. Is the verb phrase placed after 'vorausgesetzt', the verb is the first word in the sentence.

A representation of the attribute *colour*[*white*] is

[vp][adj]weiß[/adj][verb]ist[/verb][/vp]

when it is placed before 'vorausgesetzt' and

[vp][verb]ist[/verb][adj]weiß[/adj][/vp]

when it is placed after it.

There is one special case: If the attribute value is represented as a noun, it should be used as a noun phrase with the addition of 'ein'. For example, the attribute value *type*[*mountainbike*] results in the representation:

[vp][np][det]ein[/det][noun]Mountainbike[/noun][/np][verb]ist[/verb][/vp]

We distinguish the following templates for the different question types:

1. $\neg A$: ...nicht A? (example: ...nicht weiß ist?)
2. $A \vee B$: ...entweder A oder B? (example: ...weiß (ist) oder schwarz ist?)
3. $A \wedge B$: ...sowohl A als auch B? (example: ...sowohl weiß ist als auch ein mountainbike (ist)?)
4. $A|B$: ...A, vorausgesetzt, dieses (inflected) entity (singular) B? (example: eine 3-Gang-Schaltung hat, vorausgesetzt, dieses Fahrrad ein Mountainbike ist?)

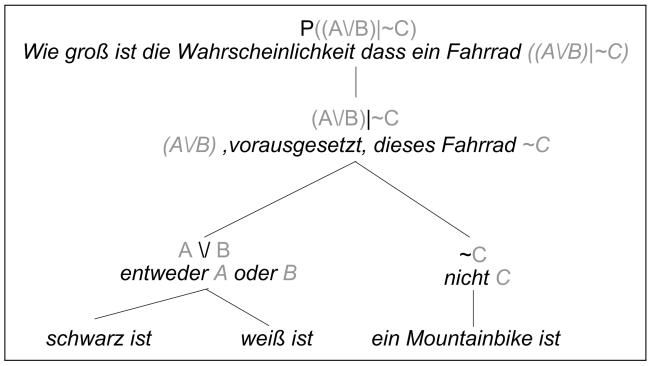


Figure 5.5: Construction of a question, a combination of multiple question templates.

[4]Translation: 'assuming that'

5.4 Micro Planner

The micro planner uses the document plan as input. This document plan with sentence trees is used in all steps of the micro planner. In each step, the micro planner can make changes to those trees. The output of this module is the document plan with modified sentence trees and will be converted into real text by the surface realizer. We will call the techniques that can be applied in the module the **text variation techniques**, because they can be used to vary the output of the text. It is possible to skip the whole micro planner, but that will result in very monotonous sentences, because all variation in language is added by this module. The techniques used in the micro planner have specific requirements for the sentences to which they can be applied, and therefore not every technique can be applied to all sentences.

In this section, we will discuss every technique in more detail. Then we show that there is a preferred order in which we should apply those techniques, to prevent unnecessary work. We will also discuss that we can use those techniques to introduce variation in the exercise text.

As we have seen in Section 2.1.1, Enright et al. [EMS02] argue that wording of a sentence could vary, without the meaning of it. Also Reiter and Dale [RD97] argue that for example the aggregation of a sentence does not change the information embedded in it, but aggregation contributes to the readability and fluency of the text. This is what we want to achieve: add variation in the language without affecting the information that it contains. We are therefore using aggregation, and some other techniques, as text variation techniques.

We discuss each technique that can be applied in the micro planner in the following subsections.

5.4.1 Aggregation

In the **aggregation** process of the micro planner, multiple sentences are grouped together and aggregated in one new sentence. If we look at the structure of the sentence, then the syntactical category of the original sentences will be clauses in the new sentence, and they will be connected to each other by a conjunction. The order of the sentences will remain intact.

Because the aggregation process is the first process in the micro planner, the input consists of simple sentence trees with a basic structure. Only the sentences reasoning about the value of the same attribute are grouped together. This is important, because otherwise it is possible that the resulting sentence is ambiguous:

For example:

1. *5 bikes are white. 6 bikes are mountain bikes.*
2. *5 bikes are white and 6 bikes are mountain bikes.*

The meaning of both sentences is equal: there are 5 bikes that are white and 6 bikes that are mountain bikes, but it could be that there are bikes both white

and mountain bike. The '5 bikes' and '6 bikes' are not by definition different bikes, but the second sentence could suggest this.

Notice that the order of the sentences is defined by the order of the statements in the probability problem. As we have discussed in Section 4.4.5, the default order of the statements ensure that statements that discuss the same attribute value follow each other.

However, it is therefore possible that simple statements that give the counts for different values of the same attribute do not follow each other: if so, those sentences are not aggregated, because of this order.

If a lot of sentences can be grouped, aggregation is performed on a maximum of three sentences. A resulting text with one large conjunction is prevented. One exception is made: if 4 sentences are grouped, they will be aggregated into two new sentences, where both are the aggregation of two of the original sentences. This will result in two sentences that are similar in both length and complexity.

If four sentences can be aggregated, the system won't generate:

- 42 Fahrräder sind Mountainbikes, 168 sind Rennräder, 200 sind Hollandräder und 10 sind Seniorenräder.[5]
- 42 Fahrräder sind Mountainbikes, 168 sind Rennräder und 200 sind Hollandräder.
  10 Fahrräder sind Seniorenräder.
- 42 Fahrräder sind Mountainbikes.
  168 Fahrräder sind Rennräder, 200 sind Hollandräder und 10 Fahrräder sind Seniorenräder.

In the first example, only one sentence is given, with 4 attribute values. In the second example, the first sentence discusses 3 attribute values and the second sentence only one. The first sentence is much longer than the second one.

Genpex will aggregate the sentences into two sentences, both discussing two attribute values:

- 42 Fahrräder sind Mountainbikes und 168 sind Rennräder.
  200 Fahrräder sind Hollandräder und 10 sind Seniorenräder.

5.4.2 Adjectivication

For the technique that we will call **adjectivication**, we change the position and grammatical role of the adjective in the sentence. The standard grammatical role that represents the value of an attribute in a simple sentence is 'predicative complement'. If we apply adjectivication to a sentence, the adjective will be placed in front of the noun that represents the entity, and the function of this adjective will be 'modifier', because it modifies the meaning of the noun. Furthermore, the verb connecting the noun phrase and the adjective is removed.

[5] Translation: '42 bikes are mountain bikes, 168 are sport bikes, 200 are holland bikes and 10 are senior bikes.'

To make the sentence complete again, the words 'Es gibt'[6] should be added as a verb phrase at the beginning of the sentence.

Of course, adjectivication could only be applied to sentences in which the complement is an adjective.

For example, the basic sentence "10 Fahrräder sind weiß." with structure

```
[np grammaticalRole=su]
[det grammaticalRole=num exerciseFunction=number]10[/det]

[noun grammaticalRole=hd exerciseFunction=entity]Fahrräder[/noun]
[/np]
[vp]
[verb grammaticalRole=head exerciseFunction=connection]sind[/verb]
[adj grammaticalRole=predc exerciseFunction=value]weiß[/adj]
[/vp]
```

is changed to "Es gibt 10 weiße Fahrräder."[7] with the sentence tree structure

```
[vp]
[adv]Es[/adv]
[verb]gibt[/verb]
[/vp]
[np grammaticalRole=su]
[det grammaticalRole=num exerciseFunction=number]10[/det]
[adj grammaticalRole=mod exerciseFunction=value]weiß[/adj]
[noun grammaticalRole=hd exerciseFunction=entity]Fahrräder[/noun]
[/np]
```

The inflection of the adjective will be changed, based on the different grammatical role. This change is made in surface realizer.

5.4.3 Entity substitution

We replace or substitute the noun that represents the entity (and has exercise function entity) with the noun that represents the value (exercise function value). The value should be expressed as a noun.

For example, the basic sentence "10 Fahrräder sind Mountainbikes." with structure

```
[np grammaticalRole=su]
[det grammaticalRole=num exerciseFunction=number]
10[/det]
[noun grammaticalRole=hd exerciseFunction=entity]
Fahrräder[/noun]
[/np]
[vp]
[verb grammaticalRole=head exerciseFunction=connection]
```

[6] Translation: 'There are'

[7] Translation: 'There are 10 white bikes.'

```
sind[/verb]
[noun grammaticalRole=predc exerciseFunction=value]
Mountainbikes[/noun]
[/vp]
```

is changed to "Es gibt 10 Mountainbikes." with the sentence tree structure

```
[vp]
[adv]Es[/adv]
[verb]gibt[/verb]
[/vp]
[np grammaticalRole=su]
[det grammaticalRole=num exerciseFunction=number]10[/det]
[noun grammaticalRole=hd]Mountainbikes[/noun]
[/np]
```

5.4.4 Change word order to marked word order

For this technique, we change the default word order of the sentence, so that a part of the sentence is emphasized. The words before the main verb of the sentence are swapped with the words after this verb.

For example, the basic sentence "10 Fahrräder sind Mountainbikes." with the default word order and the structure

```
[np grammaticalRole=su]
[det grammaticalRole=num exerciseFunction=number]10[/det]
[noun grammaticalRole=hd exerciseFunction=entity]
Fahrräder[/noun]
[/np]
[vp]
[verb grammaticalRole=head exerciseFunction=connection]
sind[/verb]
[noun grammaticalRole=predc exerciseFunction=value]
Mountainbikes[/noun]
[/vp]
```

is changed to "Mountainbikes sind 10 Fahrräder."[8] resulting in the sentence tree structure

```
[vp]
[noun grammaticalRole=predc exerciseFunction=value]
Mountainbikes[/noun]
[verb grammaticalRole=head exerciseFunction=connection]
sind[/verb]
[/vp]
[np grammaticalRole=su]
[det grammaticalRole=num exerciseFunction=number]10[/det]
[noun grammaticalRole=hd exerciseFunction=entity]Fahrräder[/noun]
[/np]
```

[8] Translation: 'Mountain bikes are 10 bikes.'

It is not always possible to use a marked word order in a sentence. Whether marked word order is suitable or not depends on the context of the sentence.

5.4.5 Techniques in combined statements and questions

If a technique is used in the sentences that represent the statements that give the count for a single attribute value, the same technique is used in the sentences for the statements that combine the count of two attribute values. If for example adjectivication is applied in a sentence that describes the statement that the bike is white, then in a sentence that combines the attribute value that a bike is white with another attribute value, adjectivication is used for this attribute value in the combined sentence. Also, if entity substitution is used in a sentence for a simple statement, it should be used in the sentence for the combined statement too.

For example, if in the sentences discussing single attribute values include the following sentences

"Es gibt 10 weiße Fahrräder. Es gibt 20 Mountainbikes."

then the combined statement discussing bikes that are both white and mountain bike will look like

"Es gibt 5 weiße Mountainbikes."[9]

It is possible to apply the text variation techniques to some of the textual representations of questions. However, because we cannot apply the text variation techniques on every question, we will not use any text variation technique on the sentences with a question. This will prevent any confusion for the student.

5.4.6 Ellipsis

The **ellipsis** of words is applied after all other techniques are applied. Ellipsis is the removal of words from the sentences, without affecting the meaning of the sentence. Only duplications of words are removed. Duplications of words only occur in aggregated sentences, so ellipsis is only applied on those sentences.

We can remove all except the first verb in a combined sentence, if they are equal in the following clauses of the sentence. In the sentence "10 bikes are white and 5 bikes are black." the second 'are' can be removed without affecting the meaning of the sentence: "10 bikes are white and 5 bikes black.".

Also the noun that represents the entity can be removed in a similar way: In the sentence "10 bikes are white and 5 bikes are black." the second reference to 'bike' can be removed: "10 bikes are white and 5 are black.". A combination of the ellipsis of a verb and the noun is also possible: "10 bikes are white and 5 black.".

Ellipsis is also possible in sentences with marked word order: "White are 10 bikes and black are 5 bikes." results in "White are 10 bikes and black are 5.". However, in this sentence, the verb cannot be removed from the last clause.

[9] Translation: 'There are 5 white mountain bikes.'

In sentences in which only some clauses have marked word order, we should separately consider the clauses with marked word order and the ones without. If clauses with marked word order and without marked word order follow each other, no words can be removed without affecting the meaning of the sentence. Ellipsis will most likely result in grammatically incorrect sentences. For example, in the sentence "5 bikes are white and black are 10 bikes." the system will not apply ellipsis.

5.4.7 Techniques used in the Micro Planner

Because all techniques that we discussed change the same sentence trees, and because they are only applied on sentence trees that have a certain structure, the techniques can be applied in a random order.

There is however a preferred order for the techniques. As we said, ellipsis is only applied on aggregated sentences. We should therefore first apply aggregation, before we apply ellipsis. The following order of techniques is used in Genpex:

1. Aggregation
2. Apply marked word order, adjectivication and entity substitution techniques
3. Perform ellipsis of verbs and nouns

5.4.8 Variation

The techniques described in this section are designed to change the structure of the sentence, without affecting its meaning. We can therefore use those techniques to vary the output of the NLG-process. Variation is introduced by randomly applying a technique with a certain chance that is defined by both the context and the user in the GUI.

For every sentence, the system will check for each technique whether it can be applied by analyzing the structure of the sentence. The chance whether this technique is applied or not, is specified by a probability in both the contents and in the GUI of the program. In the context, the user can prevent the system from using a specific technique (by giving a probability of 0), if a certain technique is never suitable for the sentences that represent the attribute values. The probability that is specified in the GUI influences all attribute values. With this probability, a certain technique can be skipped for all attribute values, in every context. This can be used to exclude a technique, if further research shows that this technique has unintended influence on the difficulty of the exercise.

If in both the context and GUI a probability is specified, the chance that the technique is used is the product of both probabilities. If for example in the context a probability of 50% is given, and 80% in the GUI, the chance that this technique is applied (if it is possible to apply this technique) is 40%. While this is a flexible design choice, it is suggested to only use 0% or 100% in the context file. The user can further adjust the probability via the GUI and does not need to know the value that is specified in the context file.

5.5 Surface Realizer

In the Surface Realizer, we will apply a correct morphology (the inflection of words) and orthography (the layout of the sentences, and correct use of punctuation marks). Both morphology and orthograpy depend on the language of the text. Because our exercises are German, we should apply a correct German grammar.

5.5.1 Morphology

The main task of this module is to convert the basic sentence trees to actual text. Information about the words that should be used in this sentence is already available as **word value** in the leaf nodes of the tree. Some leaf nodes do not have a word value such as nodes that represent an article; other word values have to be correctly inflected. After the morphology has been applied, all leaf nodes will have a correctly inflected word value.

The inflection is changed, due to the gender, number and case of the word. The gender of a word can be masculine, feminine or neuter, the number indicates whether a word is singular or plural. A case is the inflectional form of a noun, adjective and pronoun. In German four cases exist: nominative, accusative, dative and genitive. Due to the nature of the exercises for which we generate text, all words are in the nominative case.

Articles

An article always accompanies a noun. The word value of an article is based on the noun. Articles are inflected and agree in gender, number and case with the noun they accompany. In German, we distinguish the definite singular articles 'der' (masculine), 'die' (feminine) and 'das' (neuter), and the indefinite singular article 'ein' (masculine and neuter) and 'eine' (feminine). The plural definite article is for all genders 'die'.

The morphology process looks for all articles in the sentence tree, and finds for every article the noun it accompanies. Whether an article is definite or indefinite is specified in the word value of the article: if its value is 'ein', it is indefinite, otherwise it is definite. Together with the gender and number of this noun, the correct inflection for the article is chosen. The word value of the article node will now represent the correct inflection of the article.

Conjunctions

In the micro planner sentences were grouped and combined in a new sentence tree. These clauses in the new sentence are separated by a conjunction-node, that has not got a word value yet. To express the junction of multiple clauses, in German the following structure is used:

[clause 1], [clause 2] und [clause 3]

All clauses are connected to each other by a comma, except for the last connection where the word 'und' is used. The morphology process will therefore replace the word value of all conjunction-nodes with a comma, except for the last conjunction-node that gets the word 'und' as word value.

Adjectives

An adjective also accompanies a noun and the case and gender are defined by the case and number of this noun. To be able to correctly inflect an adjective we also have to consider the determiner that is used in front of the adjective. This determiner is either a determiner node or an article node representing a definite article ('der', 'die', 'das') or indefinite article ('ein' or 'kein'). The word value of a determiner node is a cardinal number ('zwei', 'drei' or '2', '3' etc.). In German, 'strong inflection' has to be used after a cardinal number, 'weak inflection' after definite articles and 'mixed inflection' after 'ein' or 'kein'.

The inflections for 'strong inflection', used after a cardinal number in the nominative case are:

- Singular masculine: stem + er (e.g. großer)
- Singular feminine: stem + e (e.g. große)
- Singular neuter: stem + es (e.g. großes)
- Plural: stem + e (e.g. große)

The inflections for 'weak inflection', used after a definite article in the nominative case are:

- Singular masculine: stem + e (e.g. große)
- Singular feminine: stem + e (e.g. große)
- Singular neuter: stem + e (e.g. große)
- Plural: stem + en (e.g. großen)

The inflections for 'mixed inflection', used after the indefinite article 'ein' or after 'kein' in the nominative case are:

- Singular masculine: stem + er (e.g. großer)
- Singular feminine: stem + e (e.g. große)
- Singular neuter: stem + es (e.g. großes)
- Plural: stem + en (e.g. großen)

While there is a grammatical difference between the cardinal 'ein', that needs to be inflected using 'strong inflection', and the indefinite article 'ein', that needs to be inflected using 'mixed inflection', in practice this difference can be ignored. If 'ein' is used, the noun is always singular, and the singular inflections for 'strong inflection' and 'mixed inflection' are equal for every gender.

Nouns

In spite of the fact that all nouns that we use are in the nominative case, there is no default inflection for most nouns. German nouns are divided into many different inflection classes and each class of nouns is inflected differently. The inflection class can be found in a dictionary.

Based on the inflection class and the stem of the word, the right inflection of a noun is defined. However, there are numerous exceptions to the basic rules set by the inflection class: sometimes an e is deleted, an 's' or 'n' has to be doubled, or the plural sometimes needs an umlaut. In the surface realizer, we can implement all those exceptions; however this will be hard and there will always remain exceptions that could not be covered by simple rules. As we have seen, the nouns in our sentences are all in nominative case. We therefore do not have to know all inflections for every inflection class of the noun, as long as we know the singular and plural form in the nominative case for every noun that we use. We choose to make our own dictionary that contains the inflections for the singular and plural case of each noun that is used.

Whether a noun is singular or plural is determined by the determiner. If this determiner is a cardinal number that is bigger than one, the noun is plural. This is already determined in the document planner, and in the node representing the noun in the sentence tree it is indicated whether the noun is singular or plural. In the surface realizer, we will look up the correct inflection in our own dictionary.

Verbs

For the verbs, we have to look at only one tense: the present, indicative tense. Where every verb has a different inflection for different persons, in our sentences the only inflection that is needed is for verbs in the third person singular (er/sie/es) and third person plural (sie).

The forms of the present indicative tense for regular verbs are formed according to the following simple rules:

- third person singular: stem + t (e.g. bringt)
- third person plural: stem + en (e.g. bringen)

For verbs, there are also many exceptions to this basic rule. We also use irregular verbs in our sentences. Because we are only interested in the two tenses, we choose to add those two word forms for all the verbs that we use to our dictionary.

The process of determining whether a verb is singular or plural is similar to the dermination whether a noun is singular or plural.

5.5.2 Orthography

Orthography is the process that converts the sentence trees to text and that adds markup to it. Converting sentence trees into text is very easy: The word

order is already defined by the structure, so all values of the nodes in the tree can be joined together in a sentence in that order, separated by whitespaces. The first letter of the first word in every sentence should be capitalized. In German, the first letter of every noun has to be capitalized too. Every sentence should be ended by a dot, except for questions that should end with a question mark instead. Furthermore, within a sentence a punctuation mark should not be preceded by a whitespace.

Finally, the orthography process will layout the text. The introduction and statements are both placed in a new paragraph. To separate all questions, every question starts on a new line.

Chapter 6

Evaluation of Genpex

To check whether Genpex works as discussed, we performed several tests that we will discuss in this chapter. Because not only the program itself is important, but also the correctness of the output, we extensively evaluated the generated exercises.

Each test that we performed, tested Genpex on one or more of the following four aspects:

1. Global functionality of the program and GUI
2. Correctness and quality of the generated language
3. Correctness of the mathematical exercises embedded in the text
4. Ambiguity of the created exercises

We first give an overview of all tests that were performed during the evaluation of Genpex, and the purpose of those tests. We then continue with the discussion of the four different aspects in separate sections. Every test is discussed in one or more sections. The functionality is discussed in Section 6.2, the language in Section 6.4, the correctness of the mathematical exercises in Section 6.3 and the ambiguity of exercises is discussed in Section 6.5.

6.1 Overview of the tests

In total, we performed 7 tests where the first two tests were performed by a couple of two test subjects and the other tests were performed by single test subjects. We give a short description of each test and the relation between test and test subjects in Table 6.1. In Table 6.2, we give an overview of all tests and indicate whether the test subject was a German native speaker.

We specify the focus of each test by one or more stars, for the first three aspects that we discussed above. The degree of focus on the aspects varied from * if this aspect was only globally evaluated, ** if we specifically tested this aspect to *** if every option of this aspect was tested. If an aspect is not tested at all,

Test nr.	Description
Test #1	Extensive review of the functionality of Genpex v1, performed by two test subjects simultaneously. *(Section 6.2.1)*
Test #2	A first review of the mathematical exercises generated by Genpex v1, performed by the same test subjects as in test #1 *(Section 6.5.1)*
Test #3	Extensive review of the language generated by Genpex v1, test was performed by a native speaker *(Section 6.4.1, 6.4.2, 6.5.1)*
Test #4	Global evaluation of Genpex v2 *(Section 6.2.2)*
Test #5	Extensive review of the mathematical exercises, created with Genpex v2 *(Section 6.3, 6.5.2)*
Test #6	Test in which we tried to find suggestions from a native speaker for the improvement of the language generated with Genpex v2 *(Section 6.4.2, 6.3)*
Test #7	Confirmation of correctly applied changes in Genpex v2 to the language. The test subject was the same subject as in test #3, a German native speaker. *(Section 6.4.1, 6.5.2)*

Table 6.1: Overview of all tests, with a reference for each test to the sections in which we discuss the results of that test.

Test	Native sp.	1: Function	2: Language	3: Math	Genpex version
#1	No	***	*	-	Genpex v1
#2	No	*	*	**	Genpex v1
#3	Yes	-	***	*	Genpex v1
#4	Yes	**	**	*	Genpex v2
#5	No	-	*	***	Genpex v2
#6	Yes	-	**	**	Genpex v2
#7	Yes	-	***	**	Genpex v2

Table 6.2: Overview of all tests. The degree of focus for each aspect is indicated by the number of stars.

this is indicated with a -. Ambiguity was not tested separately. If an exercise was ambiguous, this became clear during the test of the mathematical aspect of the exercise.

6.2 Functionality of Genpex

The first part of the evaluation was to test the functionality of Genpex. As we have discussed, the main purpose of the program is to create several exercises that are different in wording, but of the same difficult level. We tested all functions of the program to determine whether they worked as expected. After we preformed tests with 'Genpex v1', we made some improvements and performed a test of the functionality with this improved version.

6.2.1 An extensive test of all functionality

As we have discussed in the introduction (Chapter 1), one of the goals of this project is that it can be used in the research performed at the University of Twente and the University of Münster. In Twente, the test subjects that performed the extensive test of the functionality had knowledge of this project. It was possible to perform a complete test of the functionality of the program, because they have followed the development of the program closely and have extensive knowledge on the research project.

During Test #1, both test subjects were present. Genpex was used on one computer and all tasks had to be performed by both subjects. Because they had to work together, it was possible to overhear their reasoning: as soon as something was unclear, they discussed it and tried to solve the problem together. I was present as an observer and only made notes; this was to prevent any influence on the test results. A program was used to capture the screen of the computer, so that it was possible to replay the interesting parts of the evaluation and analyze them in greater detail. The discussion between the test subjects was also recorded.

In this test we used 'Genpex v1' which was the first complete version of Genpex. All functionality discussed in the previous chapters was already present, but only a very limited help-function was available. Due to the test subjects' extensive knowledge of this project, this was not expected to be a problem.

The first assignment that they had to perform was to play with the program: we asked to make exercises and try all options of the program. During the test, we ensured that all functionality of the system was verified by keeping track of all functionality that was tested. We explicitly asked the test subjects to test the functionality of the program that they had not tested.

First impression

Both test subjects had seen an early version of Genpex during its development and also the previously developed demonstrator version. Compared to these early versions, a major change was made to the interface of the program. In the new layout of the program, much effort was put in making clear which information can and which information cannot be changed. As a result, all information displayed in the program is connected to each other: if someone changes the input, this is directly reflected in the output. Also, as soon as you generate a probability problem, the exercise text is generated. This was not immediately clear to the subjects, because they expected to find a 'Generate Text' button, as there was in the development version of Genpex and the Demonstrator.

They found out that this text was generated automatically eventually, but this functionality was still confusing. In Genpex, there are two buttons available that make changes to the probability problem. One button randomizes the order of the statements, the other button changes the counts that are used. Both buttons are on the 'Probability Problem tab', so they were able to see that the probability problem changed directly. Because the generated exercise text is not visible when you press one of the buttons on the 'Probability Problem

tab', at first it remained unclear that this text was updated too. After they had carefully examined the changes, it became clear that the text was updated too.

Most functionality worked as expected, however some functionality needed further explanation. First of all, the list of question types was unclear. As we have discussed in Section 4.3.3, in the description was specified that the letter 'h' and a number could be added to a question type, in order to hide the count of an attribute value from the statements section of the probability problem. The exact meaning of the number after the 'h' was a bit unclear: they thought that this number indicates the number of attribute values without a count, but the number indicates which attribute value's count is hidden. It is not possible to hide more than one count per attribute, but that was not clear from the naming of these question types.

During the test, the test subjects used all options of the program without mentioning them explicitly. The only exception was that they did not edit the probability problem by hand. At first, it was unclear that the generated probability problem that is shown could also be edited. I had to ask the test subjects to make changes to this probability problem, before it was clear that it was possible to edit the probability problem by hand.

Creation of correct probability problems

It was difficult for the test subjects to edit the probability problem by hand and ensure that the probability problem was still correct. There are many restrictions on the probability problem; one of them is that the sum of the counts of all attribute values has to be equal to the total number of entities. If someone changed the total number of entities in the world, at least one count per attribute had to be changed too. As soon as one attribute was forgotten, an error was given, but it gave not much information on how to resolve this problem.

It was also difficult to change the attribute value into another attribute value, because it was unknown what attribute values were available in the context that was used. This information is available in Genpex and it will be helpful to show it to the user. Genpex v1 has a status box that shows whether the probability problem is correct (both in structure and contents), but this functionality could be improved.

Slides for text variation

In 'Genpex v1', a new text was generated as soon as one of the values changed. For example, if the probability that one technique was used was 50%, the program applied the technique in every possible situation with a probability of 50%. When the text was updated because the probability for one of the slides for the text variation was changed, this process was performed again. It was very likely that the technique now was applied to a different part of the sentences. When one variation technique was slightly changed, the whole text looked different and it was therefore difficult to see what the result was of the change that was made.

We can best explain this by the following example that was generated using the settings 'adjectivication' 30% and 'entity substitution' 75%:

10 Fahrräder sind weiß, 15 Fahrräder sind schwarz und es gibt 25 grüne Fahrräder. Es gibt 15 Mountainbikes, es gibt 25 Rennräder und 10 Fahrräder sind Hollandräder.[1]

If we change the setting of 'entity substitution' to 50%, and make no changes to 'adjectivication', the following text could be generated:

10 Fahrräder sind weiß, es gibt 15 schwarze Fahrräder und 25 Fahrräder sind grün. Es gibt 15 Mountainbikes, 25 Fahrräder sind Rennräder und 10 Fahrräder sind Hollandräder.[2]

The technique 'adjectivication' was applied differently, but that was not expected behaviour because we had only changed the probability for 'entity substitution'. For every sentence, once more was determined whether 'adjectivication' should be applied. This time, the technique was applied to different sentences than the first time. In an exercise, not only the applied text variation techniques were different, but also the introduction was again randomly chosen.

To be able to test the seperate text variation techniques, a workaround was used by the test subjects. They set all text variation techniques to 0% or 100%, so that techniques were never or always applied. It was then possible to vary one of the techniques and see its influence on the resulting text.

Improvement of Genpex v1

Based on the results of this evaluation, a couple of new functions were added and some functionality was changed. This resulted in a new version of Genpex, called 'Genpex v2':

- The help in the status box was improved. If someone incorrectly edits the probability problem, Genpex will specify what is wrong and give a suggestion for improvement. The most important improvement is that Genpex now shows what attributes and attribute values are available in the chosen context. This will help the user if he wants to change the attribute values, or add more attribute values to the probability problem.
- As soon as one edits the Exercise Text, this is noticed in the status window. The program does not check the correctness of the exercise, it only indicates that the underlying probability problem may not match the exercise text anymore.
- If one of the text variation options is changed, the system only updates the sentences in which the application of this technique is changed. If the probability of a text variation option has not changed, this text variation option is applied to the same sentences as before this change.

[1] Translation: '10 bikes are white, 15 bikes are black and there are 25 green bikes. There are 15 mountain bikes, there are 25 sport bikes and 10 bikes are holland bikes.'

[2] Translation: '10 bikes are white, there are 15 bikes and 25 bikes are green. There are 15 mountain bikes, 25 bikes are sport bikes and 10 bikes are holland bikes.'

- The introduction of the text can now be changed manually by selecting one of the available introductions.

- Instead of only saving the whole project with all log files, it is now possible to export the generated text to a separate HTML-file.

- A help-functionality was added. The help is split into a couple of help files, each discussing one of the functionalities of the program. These texts can be reached via the 'help menu', but also via small help-icons in the interface of the program that directly show the appropriate help file.

- Finally, we corrected some small errors, such as spelling errors and incorrect descriptions of certain variation techniques. Also, we included the sentence that expresses the attributes that are dependent in the exercise text, since we had omitted to do so before.

6.2.2 Global evaluation of Genpex v2

Another test focused on the functionality of Genpex, but in this test, Test #4, the improved 'Genpex v2' was used. We were not present during this evaluation: the program was sent to one of the test subjects together with a description and two tasks. In those tasks, several exercises had to be created and the results had to be saved. Because the settings that define the input of the program are embedded in the saved files, we were able to analyze the files and make conclusions based on these files. In the saved files was also a log included that we used in this evaluation too.

The first task was to generate 5 different versions of the same exercise. The test subject succeeded and used several variation techniques. First of all, different contexts were used: for the first exercise the context 'bikes' was used, for the second exercise the context 'university' and for the other three exercises the context 'coffee' was used. The differences in the exercises based on the coffee context were made by either regenerating new counts for the statements, or by randomly reordering the statements. Different settings for the text variation were used in only one exercise.

The second task was to generate an exercise that was as close as possible to a given example. The resulting exercise was not as close as possible to the given example, mainly because only the generation options of Genpex were used. It is also possible to edit the probability problem by hand, but this functionality was not used.

Overall, the test subject could work with Genpex without a detailed instruction. The test subject indicated that the generator worked well and provided lots of opportunities to generate items and clones of items. The help buttons were an important feature and helped to understand most of the features of the program.

6.3 Correctness of mathematical exercise

The test whether the mathematical exercise embedded in the test is correct is simple: we have used Genpex to generate exercises and asked a test subject in Test #5 to answer the questions in those exercises without knowledge of the probability problems on which the exercises are based. We have compared the calculations and results of the test subjects with those generated by Genpex. If the results and calculations matched, we assumed that Genpex has correctly calculates the answers.

For Test #5, four different exercises were generated. Each exercise used a different context. We used all question types, some with a negation, divided over the exercises. From the answers that were given, we conclude that the questions were answered correctly. We also compared the calculation performed by the test subject with the calculations performed by Genpex, and found a small difference. It is possible for the student to answer a question by using different calculations than those used by Genpex to answer the same question. We can give an example for the question $P(\neg A[1])$, where attribute 'A' could have one of the values '1', '2' or '3'. The answer can be calculated by using either of the following calculations:

- $1 - P(A[1])$
- $P(A[2]) + P(A[3])$

While both calculations are correct, Genpex will only use the first calculation. This is the result of the internal working of Genpex: as soon as the answer to the question is found, it stops trying other possible calculations. It is important that a teacher is aware of this. The final answer that a student gives to a question made by Genpex can be checked, but it is not possible to check whether the calculations performed by the student are correct.

6.4 Generated language

To check the generated language, we created a set of exercises by using 'Genpex v1'. We systematically made exercise stories that included all types of variations possible within an exercise. We used every available attribute and attribute value and generated exercises for every available context. We also made a list of all possible questions: for one context, we created a question for every possible question type.

During the tests that we discuss in this section, the test subjects made some remarks on whether the text was ambiguous or not. This will be discussed in Section 6.5. In this section we focus on the correctness of the applied grammar and the conjugation of words.

6.4.1 Grammar correctly applied

In Test #3, all texts were checked by a German native speaker. We did not show Genpex to prevent any distraction from the language. Furthermore, we did not tell the exact purpose of my research, but only asked the subject to read and evaluate the text of the exercises.

A couple of small mistakes were found: "5 Fahrräder haben mehr als fünf Gänge" was used instead of "fünf Gängen". There was also a mistake in one context in the reference to someone's age. More important was a remark on the questions: it is very important to add a comma before the word 'dass' in the questions, otherwise the sentence is not a correct German sentence. For example: "Wie groß ist die Wahrscheinlichkeit, dass ein Fahrrad weiß ist?"[3]. The comma in this sentence is not optional as I had expected. We will also add a comma before and after the word 'vorausgesetzt'. The word order of the clause that follows this word is changed too: to sound more natural, this has to be equal to the word order used in the statements.

All of those remarks were changed in 'Genpex v2', and tested again in Test #7 to ensure that we had correctly applied the changes.

6.4.2 Readability of the exercises

Two test subjects, in Test #3 and # 6, focused on the language used in the exercises. For both the statements and questions, some improvements were found.

Amount of ellipsis

One of techniques that caused unnatural sounding sentences in the statements section was the applied ellipsis. The test subjects indicated that for every sentence there is a preferred (amount of) ellipsis, which is different for each sentence. Because we have used ellipsis to add variation to the sentences, many sentences have a different amount of ellipsis applied than the preferred amount. The native speakers agreed that sentences in which a different amount of ellipsis was applied are not incorrect. Ellipsis can therefore be used to vary the output, but this technique should be handled with care. We will discuss this topic in more detail in future work, Chapter 8.

Rephrase 'nicht eine' as 'keine'

This small change was suggested to increase the readability of a sentence. If we use 'keine' (none) as a contraction of 'nicht eine' (not a), those sentences will be easier to understand. Both sentences have the same meaning, but the first option is more commonly used. Therefore, there is no reason to use 'nicht eine' in favor of 'keine'. In 'Genpex v2' we changed the surface realizer, so that 'keine' is used.

[3]Translation: 'What is the chance that a bike white is?'

Improvements of the language used in the questions

The test subjects from Test #3 and #6, both native speakers, indicated that while it was possible to extract the information from all the questions, some sentences had to be studied carefully before it was possible to get the information needed to make the exercise. Despite being correctly formulated, the sentence structure sometimes sounded a bit forced. Some sentences were correct, but sounded 'unnatural' to the native speaker. It is plausible that the readability of the exercise is decreased by those sentences, which is something we want to prevent.

These kinds of questions are not likely to be asked in a real-life situation, except during a math examination. It is probably impossible to ask those mathematical questions using sentences that sound completely normal, but during tests #3 and #6 we asked the test subjects to give suggestions for possible improvements.

The language used in some of the questions can be improved too. In daily use, if you try to express the following question:

Wie groß ist die Wahrscheinlichkeit, dass ein Fahrrad sowohl weiß ist als auch eine 3-Gang-Schaltung hat?[4]

it sounds more natural to use 'und' instead of 'sowohl ... als auch', which results in the following question:

Wie groß ist die Wahrscheinlichkeit, dass ein Fahrrad weiß ist und eine 3-Gang-Schaltung hat?

In this specific situation, the meaning of the question is not changed by this different use of words. However, the use of 'und' is not suitable to be applied in every situation, especially not in combinations with a negation. The mathematical meaning of the question might be unclear or even change.

To illustrate this problem, we consider the following sentence:

Wie groß ist die Wahrscheinlichkeit, dass ein Fahrrad nicht weiß ist und eine 3-Gang-Schaltung hat?[5]

This problem is that we have made this sentence ambiguous. This situation is similar to the one that we wanted to exclude as discussed in 4.3.1. The previous sentence can be interpreted as follows (where we have emphasized the negation):

1. *Wie groß ist die Wahrscheinlichkeit, dass ein Fahrrad* ***nicht sowohl weiß ist als auch eine 3-Gang-Schaltung hat****?*[6]
2. *Wie groß ist die Wahrscheinlichkeit, dass ein Fahrrad sowohl* ***nicht weiß ist*** *als auch eine 3-Gang-Schaltung hat?*[7]

We cannot apply 'und' in every situation and we will therefore not change this in 'Genpex v2'. If we would like to apply this in a future version of Genpex, more research on the sentences regarding ambiguity should be performed.

[4]Translation: 'What is the chance that a bike is both white and has 3 gears?'
[5]Translation: 'What is the chance that a bike is not white and has 3 gears?'
[6]Literal translation: 'What is the chance, that a bike **not both white is and 3 gears has**?'
[7]Literal translation: 'What is the chance, that a bike both **not white is** and 3 gears has?'

A conditional probability in a question

Questions that ask for the calculation of a conditional probability are somewhat difficult to understand. Of course, if you are not used to making those kinds of mathematical exercises, that might influence your understanding of the exact meaning of the sentence. In this situation, the sentences are grammatically correct, but the word 'vorausgesetzt' makes the sentence more complex to understand. It is easier to understand the exact meaning of the question when the 'verb' or 'verb phrase' (the connection between the entity and attribute value) is not placed at the end of the sentence, but just before the attribute value.

To illustrate, we give two questions, where the second question is easier to understand:

1. "Wie groß ist die Wahrscheinlichkeit, dass ein Fahrrad weiß ist, vorausgesetzt, dass dieses Fahrrad billiger als 500 Euro ist?"
2. "Wie groß ist die Wahrscheinlichkeit, dass ein Fahrrad weiß ist, vorausgesetzt, dieses Fahrrad ist billiger als 500 Euro?"[8]

In 'Genpex v1' the first sentence is used. This is changed in 'Genpex v2', which uses the second sentence to express the question for a conditional probability.

6.5 Ambiguity

As we have discussed in the introduction and our requirements, the system should avoid ambiguous sentences. The language used in the sentences must not influence the students' capability to solve the exercise. During the previously discussed tests, we found two problems in which ambiguity plays a role.

6.5.1 Ambiguity in questions with conditional probabilities

In Test #2, exercises were generated with 'Genpex v1'. One test subject created the exercises, the other tried to answer the questions without knowledge of the corresponding probability problem. Not every question type was included, but a more global review of the exercises was performed. The statements and most questions were not ambiguous, due to the structure of the sentences.

However, for one question that required the calculation of a conditional probability, the meaning was unclear and was interpreted in two ways. The sentence is shown here, where we have emphasized two different parts of the same sentence to illustrate the difference between the interpretations:

1. Wie groß ist die Wahrscheinlichkeit dass ein Zeitschrift entweder einmal vierteljährlich erscheint oder **einmal monatlich erscheint**

[8] Translation: 'What is the chance, that a bike white is, given that this bike is cheaper than 500 euro?'

vorausgesetzt dieses Zeitschrift zwischen 3 und 7 Euro ist?[9]

2. Wie groß ist die Wahrscheinlichkeit dass ein Zeitschrift **entweder einmal vierteljährlich erscheint oder einmal monatlich erscheint** vorausgesetzt dieses Zeitschrift zwischen 3 und 7 Euro ist?[10]

Just by emphasizing another part of the question, the scope of the conditional statement (indicated by 'vorausgesetzt') changed. The interpretation of the question is changed too. This difference in interpretation is best shown by the formal representation of the questions that belong to the two different interpretations:

1. $P(X = x_i \vee (X = x_j | Y = y_k))$
2. $P((X = x_i \vee X = x_j) | Y = y_k)$

If someone can derive both formal representations, the sentence is ambiguous. The calculation and result of those formal representations is different. This question should be rephrased to ensure that it is always correctly interpreted as the second interpretation.

We should note that the test subjects in Test #2 were not native speakers, but this problem was later confirmed by a native speaker in Test #3.

Preventing this ambiguity

As we have discussed before, we have already made some changes to 'Genpex v1' so that sentences with a conditional probability sound more natural. First of all, we should check whether those changes are sufficient for the problem stated above or whether extra changes are required.

If we generate the sentence for the question with the intended formal representation with 'Genpex v2', we will get:

Wie groß ist die Wahrscheinlichkeit, dass eine Zeitschrift entweder vierteljährlich oder monatlich erscheint, vorausgesetzt, diese Zeitschrift ist zwischen 3 und 7 Euro?

As we have discussed in Section 6.4, we added extra commas before and after the word 'vorausgesetzt' and we changed the word order of the clause after this word. Furthermore, we have applied ellipsis for the first use of 'erscheint'. The test subject in Test #7 answered the questions without problems, and indicated that this new phrasing cannot be misinterpreted.

We can further limit ambiguity by rephrasing the question more thoroughly. The same question can be asked as follows:

Vorausgesetzt dass ein Zeitschrift zwischen 3 und 7 Euro ist, wie groß ist die Wahrscheinlichkeit dass ein Zeitschrift entweder einmal vierteljährlich or einmal monatlich erscheint?

[9] Translation: 'What is the chance, that a magazine either appears 4 times a year or **appears each month, given that this magazin costs between 3 ans 7 euros**?'

[10] Translation: 'What is the chance, that a magazine **either appears 4 times a year or appears each month**, given that this magazin costs between 3 ans 7 euros?'

This sentence avoids any problems regarding ambiguity, and it is confirmed by a native speaker that this representation of the question is easier to understand.

However, this solution is only applicable for this specific type of questions. In our program, we prefer to have generic techniques to construct questions: with generic techinques it will be much easier to add new question types to the system. Furthermore, the student will not be confused by two questions that look completely different, but are in fact mathematically very similar.

6.5.2 Exclusive or inclusive 'or'

Another source of ambiguity is found in some probability problems. During Test #5, that focused on the correctness of the mathematical exercises, the test subject was able to answer the exercises correctly, but the exact meaning of questions that included an 'or' remain somewhat unclear. The exclusive-or explicitly states "one or the other, but not both". In contrast, the normal-or means "one or the other, or both". An example of a question with this ambiguity is:

"Wie groß ist die Wahrscheinlichkeit, dass eine Zeitschrift entweder vierteljärlich erscheint oder billiger als 3 Euro ist?"[11]

In this sentence, we wanted to ask for the probability that a magazine is made 4 times a year or that it is less than 3 Euro. But we also want to include magazines that are made 4 times a year and cost less than 3 Euro. However, due to the word 'entweder', the question has another meaning: the probability that a magazine is either made 4 times a year or that it is less than 3 Euro, but not both made 4 times a year and less than 3 Euro. The difference is that we want to express a logical inclusive disjunction ('OR') and the sentence expressed an exclusive disjunction ('XOR').

To express a '$\vee$' structure in a probability problem, Genpex uses the structure "entweder ... oder ...". This expresses a 'XOR', instead of an 'OR'. If we only want to express the 'OR', we should use "... oder ...", without the word 'entweder'. However, if we just remove this word we might introduce ambiguity in the sentence. The "entweder ... oder" structure surrounds a clause with the words 'entweder' and 'oder', that indicated the scope within the sentence. Due to the structure of the questions, the formal meaning of the exercise does not change for most question types whether we use 'XOR' with the word 'entweder' or use a normal 'OR' without 'entweder'. If the two sides of the 'OR' are disjoint (conditions for the sides to be disjoint are discussed in Section 4.5.1), the calculation that has to be performed is the same as the calculation for a similar question that uses a 'XOR'. Implicitly, there is a 'XOR' used, because both sides exclude each other.

To illustrate, we give two sentences that require the same calculations and will result in the same answer:

[11] Translation: 'What is the chance, that a magazine either 4 times a year appears or cheaper than 3 euros is?'

1. *"Wie groß ist die Wahrscheinlichkeit, dass ein Fahrrad weiß oder schwarz ist?"*[12]

2. *"Wie groß ist die Wahrscheinlichkeit, dass ein Fahrrad entweder weiß oder schwarz ist?"*

As we have defined before, every attribute of an entity has exactly one value. A bike has therefore exactly one colour. In the previous questions, a bike is either white or black, so in both situations we express an exclusive or.

The sentence structure of a question is defined by the underlying representation in the probability problem. In the generation process, this structure is defined by the question type, as is shown in Table 4.1. The question types, that may result in questions in which the given 'OR' is not an exclusive OR, are numbered '3', '4', '3n0', '3n1' and '4n0'. In the questions based on question type '3' and '4' we can remove the word 'entweder' without changing the scope. In the other three question types, we get a scope problem if we remove the word 'entweder'.

We can illustrate this problem with the following example of question type '3n0':

"Wie groß ist die Wahrscheinlichkeit, dass eine Zeitschrift nicht ~~entweder~~ vierteljärlich erscheint oder billiger als 3 Euro ist?"[13]

The 'OR' in this sentence expresses a 'XOR', because of the use of the word 'entweder'. The probability problem of the intended meaning is expressed as follows:

$P(\neg(frequency[4peryear] \vee price[< 3]))$

If we remove the word 'entweder', the 'or' is changed in an 'inclusive or' and the meaning of the sentence is changed to:

$P((\neg frequency[4peryear]) \vee price[< 3])$

We can add some words to the question that contains the word 'entweder', to indicate that a normal-OR relation is meant:

"Wie groß ist die Wahrscheinlichkeit, dass eine Zeitschrift entweder vierteljärlich erscheint, billiger als 3 Euro ist oder beide?"[14]

The test subject of Test #7 indicated that while this sentence is correct, it sounds unnatural and is still difficult to understand. Furthermore, this change cannot be used in sentences in which one of the clauses is a negation. The following sentence is nonsense, and for the native speaker impossible to derive the intended meaning:

"Wie groß ist die Wahrscheinlichkeit, dass eine Zeitschrift entweder nicht vierteljärlich erscheint oder billiger als 3 Euro ist oder beide?"

Another option is to redefine the mathematical structure that belongs to these question types and let all question types only use 'XOR'. A question, in which

[12] Translation: 'What is the chance, that a bike either white or black is?'

[13] Translation: 'What is the chance, that a magazine not either 4 times a year appears or cheaper than 3 euros is?'

[14] Translation: 'What is the chance, that a magazine either 4 times a year appears, cheaper than 3 euros is or both?'

we want to express an inclusive 'OR' using the 'XOR', should be formulated as follows:

$\neg(A \vee B) \rightarrow \neg(A \vee B \vee (A \wedge B))$

A corresponding question will be expressed in natural language, which is hardly readable:

"Wie groß ist die Wahrscheinlichkeit, dass eine Zeitschrift nicht entweder vierteljärlich erscheint, billiger als 3 Euro ist oder sowohl vierteljärlich erscheint als auch billiger als 3 Euro ist?"[15]

By this change in mathematical structure, we change the difficulty of the exercise, because we give information about the calculation that has to be performed. This structure can be considered as a completely different question type that requires a totally different insight of the underlying mathematical problem. Furthermore, it is difficult to read this question and to derive the underlying mathematical problem. We should therefore avoid this construction.

A last option could be to inform the students in an introduction text that is given before the exercises. In this introduction, an explanation should be given that states that the sentences with the structure 'entweder A oder B' indicates a 'XOR' relation if A and B exclude each other, but a normal-OR relation if they are not excluding each other. Because this is in fact a contradiction to the language that is used, we will not use this option, because it could confuse the students.

Unfortunately, we have not found an option that can correctly express question types '3n0', '3n1' and '4n0'. We will therefore exclude those question types from Genpex. For the question types '3' and '4', we can use the word 'oder' without the word 'entweder'. We expect that a student might not see the difference. This may be tested in research that uses those exercise types.

[15]Translation: 'What is the chance, that a magazine not either 4 times a year appears, cheaper than 3 euros is or both 4 times a year appears and cheaper than 3 euros is?'

Chapter 7

Conclusion

In this thesis, we have discussed the development of Genpex, the Generator for Narrative Probability Exercises. In the introduction, in Section 1.3, we gave a list of requirements on the exercises that this program should be able to generate. We also gave a list of system requirements. The goal of this project was to develop a program that meets all those requirements. Based on the results of the evaluation that we performed we have shown that the current version of Genpex meets all requirements.

In Chapter 2 we have discussed two systems that are capable of generating narrative exercises. Both systems generated a specific type of exercise and could therefore not be reused: the ModelCreator software focused on distance-rate-time problems, in WebALT the exercises contain formulas embedded in a text.

Our system design is similar to the ModelCreator software. We have created a module that is responsible for the creation of a probability problem, a structured representation of the mathematical problem. During this generation process, some variation is introduced. This variation is introduced by the possible context of the exercise, the attributes and attribute values and the type of the questions that are used. Also the numbers used in the probability problem of the exercise are varied. After the generation of a probability problem, it is converted into a German text in the language generation module.

Based on the research performed for the Narrator in the Virtual Storyteller, discussed in Section 2.2, we have chosen to use 'aggregation' and 'ellipsis' as a source of variation in the language that our system generates. We also make changes to the word order in the sentence, with the techniques that we call 'adjectivication' and 'marked word order'. As we discussed in Section 2.6, literature shows that a change in the used language theoretically does not affect the contents of an exercise. By randomly applying the text variation techniques, we make a program that is able to create several versions of the same exercise. Each version looks different, but should be equally difficult to answer.

With the help of Genpex, an user is able to easily generate new probability problems and an exercise text based on this probability problem. All techniques that

introduce variation can be excluded by manually adjusting parameter values. Our system therefore keeps track of all techniques that are applied to a text. This information can be used in further research, toghether with the generated exercise, to investigate which techniques define the difficulty of an exercise. The outcome of that research can be used in Genpex. Only the techniques that do not influence the difficulty will be randomly chosen by Genpex. This results in the generation of various exercises that all look different, but are equally difficult to answer by a student.

7.1 Discussion of the requirements

In Section 3.1, we have showed the global system design of Genpex, which has a modular design. With this design we fulfill the system requirements listed in Section 1.3.2: the input is minimal, but it is possible to guide the entire generation process. It is therefore easy to create new exercises. Nevertheless, the user has still full control over the generation process. He is able to edit the output of one module, before it is processed by the next module. The system will help the user, by checking the correctness of the specified information.

Due to its modular design, the system is prepared for future extension. It is easy to replace a single module with a new one, without affecting the other modules and functionality of the program. The two most important modules are responsible for the creation of a probability problem (in the module called 'Probability Problem Generation') and for the generation of the language of the exercise based on this probability problem (in the module 'Language Generation'). Each module has its own responsibilities: the first module is language independent, the second module does not change the mathematical exercise.

Every requirement on the exercises that we introduced in Section 1.3.1 is ensured in one of these modules. All requirements on the language are ensured in the 'Language Generation' module, that ensures a correct German text. In the 'Probability Problem Generation' module, we ensure that we can generate at least the type of exercises that are used in the research of Münster. Furthermore, we ensure that a student cannot use a previously calculated answer in other questions. It also ensures that all numbers used in the probability problem are different. In the 'Language Generation' module, we ensure that every question type that can be generated in the 'Probability Problem Generation' module results in an unambiguous text.

7.2 Evaluation

We performed an evaluation to check Genpex on all system requirements and all requirements on the exercises. We focused on 4 different aspects: the global functionality, the correctness of the generated language, the correctness of the generated mathematical exercises and the ambiguity in the texts.

During the evaluation of the first version of Genpex, we found some small mistakes in the used language. We also received some suggestions to improve the

program. It was relatively easy to fix the mistakes and add some of the suggestions to Genpex. Therefore, we created a new version called 'Genpex v2', in which we improved the language and some of the functionality. In the second part of the evaluation process, we have verified the correctness of those changes.

Based on our evaluation we can conclude that 'Genpex v2' meets the requirements listed in Section 1.3: we have made a system that is able to generate various versions of the same exercise, based on a correctly mathematical problem that is also generated. The exercises can be created easily and the output of the system include information that can be used in further analysis of the generated exercises.

While we have improved the language in 'Genpex v2', the ambiguity of the text is still a problem. As we have discussed in Section 6.5.2, we cannot correctly express an 'inclusive or'. Three question types that we proposed in Section 4.3 will result in ambiguous sentences and should therefore not be created with Genpex. We have verified that all other question types do not result in ambiguous sentences.

We have shown that for all proposed question types, Genpex is able to calculate a correct answer. The output of the system includes the calculations that are performed to calculate the answer to a question. It is important to notice that this is only one possible calculation, the student might use a different calculation to answer the same question.

7.3 Future improvements

We improved the GUI and added help buttons in 'Genpex v2'. This helped an unexperienced user to understand most of the features of the program. It is thus already possible for a researcher to use Genpex to create exercises, but there are still many opportunities to improve the program. We will discuss the most important and interesting improvents as future work in Chapter 8.

Chapter 8

Future work

In this chapter, we will discuss some future work that might further improve Genpex. In Section 3.5.3 we discussed some of the implementation details that can be used as a starting point for extending the functionality of the program. We split these possible extensions in three pieces and discuss them in separate sections. The extension of the types of exercises is discussed in Section 8.1, the context information in Section 8.2 and improvents on the used language in Section 8.3.

8.1 Introduce new types of exercises

In the statements section of the probability problem, counts are used. Together with the total number of entities, they define the world. Instead of a count we can specify probabilities in the statements section of the exercise. Holling et.al. [HBKK08] show that this 'number format' influences the difficulty of the exercise. This number formatting property can therefore not be used as a new variation technique. The use of counts or probabilities cannot be chosen per question; either counts or probabilities should be used in the whole exercise. The number formatting is therefore a global option for an exercise and cannot be included in the question types.

In the current version of Genpex, relatively simple probability structures are used. We have limited the possible dependencies between attributes, and therefore excluded certain types of probability problems. For the designer of an exercise, it would be an improvement if all exercises that can be expressed by a Bayesian network are supported. An example Bayesian network taken from the user manual of JavaBayes is shown in Figure 8.1, created with JavaBayes[1]. This network is explained in the user manual: "Suppose you are going home, and you want to know what is the probability that the lights are on given that the dog is barking and the dog does not have any bowel problem. If the family is out, often the lights are on. The dog is usually out in the yard when the

[1] http://www.cs.cmu.edu/~javabayes

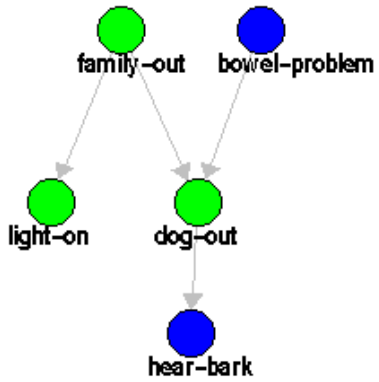


Figure 8.1: Example of a Bayesian network, created with JavaBayes.

family is out and when it has bowel troubles. And if the dog is in the yard, it probably barks."

Instead of the world that we use in Genpex, we can extend the system to let it use a Bayesian network for the creation of a probability problem. Bayesian networks are used in much research and we might benefit from previous research. Because the structure of a Bayesion network can be very complex and can include many connections, we expect that is a hard to express any random Bayesian network in natural language. This might be especially difficult if we use the type of context that we use now. The more complex dependencies that may exists in a Bayesian network should match to the context that is described. It might therefore be necessary to limit the type of Bayesion networks and the type of contexts that are supported.

If we assume that it is possible to create Bayesian networks, we probably have to extend the possible structure of a probability problem to be able to express all relations between attributes. Also, the language generation has to be extended to make it possible to express those structures in natural language. The process that calculates the result to the questions embedded in this probability problem should also be extended.

If Genpex is extended and uses Bayesian networks, it might be interesting to add a connection with JavaBayes, or at least an import/export function for JavaBayes files. It will be helpful to show and edit the Bayesian networks using the interface of JavaBayes.

8.2 Context and mathematical exercises

The generation processes in Genpex rely heavily on the available context files. For both the generation of probability problems and the text of the exercise, information is used from those context files. In Section 3.3.1 we discussed the contents and structure of the XML-files that contain the context information.

Except for the structure, there are numerous other restrictions on the content of a context file. For now, someone who wants to add a new context to Genpex is responsible for creating a correct context file. This is hard to do because it is easy to introduce errors.

8.2.1 Develop a context file editor

By using the context file tester tool discussed in Section 3.4.1, it is possible to create correct context files. However, the developer still has to edit XML-files by hand. A special editor for context files is not only easier to use, but extra features can assist the developer of new context files.

The use of a GUI will prevent the creation of XML-files of which the structure is incorrect. It is impossible to forget to specify some information, because the editor will show what information is required. Furthermore, the editor can refuse to save an incorrect context file.

The real power of a special context file editor is that it can already show the creator an example of an exercise with this context during the development of the context file. It will be easy to see whether some text variation techniques should be excluded, because they cannot be applied to the sentences created for this context. The technique 'adjectivication' can only be used if the attribute value is expressed by an adjective. The editor can show the techniques that can be applied, as soon as the attribute value is specified.

Furthermore, if other resources are connected to this editor, the system can give the creator suggestions. For example, when the creator has specified that an entity has a colour, the system could suggest all colours that are available. Because of the relations between words, it is likely to use GermaNet (discussed in Section 2.4) as resource for this information. It is likely that also other resources can provide information that is useful in the context editor.

8.2.2 Add and use more context specific information

Currently, not much information about the real world is embedded in the context files. We specify for example only the attributes that can be dependent on each other (and therefore we also exclude dependencies between other attribute combinations). However, the system does not have any knowledge on whether a combination of two attribute values occurs more frequently in the real world than other combinations. For example, we can assume that is more likely to have mountain bikes with many gears, than mountain bikes with a single gear. If this information is embedded in the context file, it can be used in the probability problem generation process and result in probability problems that describe a more realistic situation. It is also possible to use an (existing) frame database, similar to the one used in the ModelCreator software that we discussed in Section 2.1.1.

8.3 Language

Genpex already generates correct texts, but there are still many opportunities to improve the quality of the used language, and make more sophisticated texts. In this section we discuss some possible research topics and also suggest a multilingual version of Genpex.

8.3.1 Generate referring expressions in the questions

One topic that is important in many NLG systems, namely the generation of referring expressions, is not discussed in great detail during the creation of Genpex. The process of determining how to express the word or phrase that is referring to objects or persons in the text is called referring expression generation.

In the question that is embedded in the example exercise, shown in Figure 1.1, there is referred to the introduction. In the questions that Genpex generates, there is no reference to the introduction of the exercise. We can show this difference, by showing the question as have been used in the research of the University of Münster and the same question generated by Genpex:

- *"Wie groß ist die Wahrscheinlichkeit, dass ein Fahrrad ins Schaufenster gestellt wird, das nicht sowohl ein Rennrad als auch weiß ist?"*
- *"Wie groß ist die Wahrscheinlichkeit, dass ein Fahrrad nicht sowohl ein Rennrad als auch weiß ist?"*

The first sentence, used in the research of Münster, refers to the introduction that discusses a situation where a bike is chosen to be displayed in the window of the shop. This reference makes the text more coherent. It might be interesting to derive this referring expression from the introduction automatically. Alternatively, the context information can be extended so that for every introduction embedded in the context, a referring expression is specified.

8.3.2 Use synonyms and hyperonyms

There are several options to refer to the same object or person. In many NLG systems one could refer to a person by using a pronoun as soon as the person has been introduced in the previous sentence. We refer mainly to objects instead of persons due to the type of exercises. It is possible to refer in several ways to an object, for example by using a synonym or hyperonym. A synonym is another word with the same meaning, a hyperonym encloses the meaning of another word. For example, 'vehicle' is the hyperonym of 'car'. If in a sentence some information about 'a car' is given and in the next sentence information about 'the vehicle' is given, the reader knows that the same car is meant. Information about synonyms and hyperonyms can be derived from an existing resource, for example GermaNet.

Before synonyms or hyperonyms are used, it should be clear under what conditions they can be used. We should prevent any confusion: a student might

think that 'a car' and 'the vehicle' refer to different entities. Research might show that we should only choose one synonym or hyperonym and use it for the whole exercise.

8.3.3 Investigate the preferred (amount of) text variation

We have seen that it is possible to create several variations of the same sentence, in which we, for example, vary the word order. Our test subjects indicated that while all variations are correct, they prefer one variation over other variations. Some variations sound more natural, other variations will normally never be used. It might be worth to check whether certain variations created by Genpex should not be used at all.

In a research by Nenkova et al. [NCLP10], it is shown that the number of words that is used to represent the subject, influences the readability of the sentence. A sentence with a longer subject is more difficult to understand. In another research performed by Cahill and Forst [CF10], a human-based evaluation of surface realization alternatives is presented. Subjects ranked several generated sentences based on naturalness. The sentences were also ranked by the number of occurrences in corpus sentences. They showed that native speakers accept some variation in for example word order, but prefer some word orders above others.

Referring expressions are sensitive to ambiguity. For example, Kahn et al. [KDR08] discuss referring expressions that could be scopally ambiguous. They have psycholinguistic evidence that the avoidance of all ambiguity is hard. Their suggestion is an approach that avoids referring expressions that have distracting interpretations.

8.3.4 Make Genpex multilingual

The language of our system is German, because field trials are performed in Germany. Due to the cooperation between the University of Twente en Münster, it might be interesting to adapt the program so that is able to output the exercises in Dutch too.

Because the basis of our program is similar to the approach of the ModelCreator software, we expect that it is possible to add a new language in a similar way. We have designed the process so that the probability problem is language independent, and the language specific information is embedded in the language generation module. To make the system multilingual, only the NLG part of the system has to be changed. In a worst case scenario, we should redesign the whole NLG process. This might for example be necessary for language that uses a different set of characters.

For every new language that is added to Genpex, we should specify a new dictionary file for the conjugations of words in that language. There is also language specific information embedded in the context files. Both files are prepared for the extension with new languages, as we have discussed in Section 3.5.3. As we have seen in the ModelCreator software, one should consider possible cultural

differences between the current and new languages: the same exercise might be more difficult in one language than another.

8.4 Add new import and export functionality

It is also possible that the input for Genpex is created by another system. In particular, it would be interesting if JavaBayes output can be imported into Genpex (as discussed in Section 8.1). If Genpex is also extended to support Bayesian networks, those networks might be created or edited in JavaBayes and then used in Genpex.

It is possible to use the exercises generated with Genpex in field trials. As Genpex might be used in combination with existing tools or systems, it might be necessary to connect the output of Genpex to an existing test administration tool.

All information that is collected by Genpex during the generation of the exercise can be saved. This saved file could be parsed by another program that is specially designed for the conversion of Genpex-files into another format, but it is also possible to extend Genpex and make it able to export the information in a certain format.

It is possible to extend the program to generate a full test for an examinee. It is for example possible to format the output in LaTeX, and create similar tests as the examples used during the field trials performed in Münster. Those tests were made in LaTeX and started with some general instructions, then the tasks and finally a questionnaire. The LaTeX file is also used to format the exercises. Genpex is already able to export the text of the exercise to a HTML-file. This functionality can be extended to support LaTeX or any other format.

Bibliography

[ADS96] J. Abbott, A. Diaz, and R.S. Sutor. A report on OpenMath: a protocol for the exchange of mathematical information. *ACM SIGSAM Bulletin*, 30(1):21–24, 1996.

[BNM01] G. Bouma, G. van Noord, and R. Malouf. Alpino: Wide-coverage computational analysis of Dutch. *Language and Computers*, 37(1):45–59, 2001.

[BR09] A.R. Boer Rookhuiszen. Generation of German Narrative Probability Exercises. *Capita Selecta assignment*, 2009.

[CF10] A. Cahill and M. Forst. Human Evaluation of a German Surface Realisation Ranker. *Empirical Methods in NLG, LNAI 5790*, pages 201–221, 2010.

[DS03] P. Deane and K. Sheehan. Automatic item generation via frame semantics: Natural Language Generation of math word problems. *Princeton: Educational Testing Service*, 2003.

[EMS02] M.K. Enright, M. Morley, and K.M. Sheehan. Items by design: The impact of systematic feature variation on item statistical characteristics. *Applied Measurement in Education*, 15(1):49–74, 2002.

[HBKK08] H. Holling, H. Blank, K. Kuchenbäcker, and J.T. Kuhn. Rule-based item design of statistical word problems: A review and first implementation. *Psychology Science Quarterly*, 50(3):363–378, 2008.

[HBZ09] H. Holling, J.P. Bertling, and N. Zeuch. Automatic item generation of probability word problems. *Studies In Educational Evaluation*, 35(2-3):71–76, 2009.

[HFD05] D. Higgins, Y. Futagi, and P. Deane. Multilingual generalization of the ModelCreator software for math item generation. *Research report - Educational Testing Service Princeton RR*, 5, 2005.

[HK06] K. Harbusch and G. Kempen. ELLEIPO: a module that computes coordinative ellipsis for language generators that don't. In *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics: Posters & Demonstrations*, pages 115–118. Association for Computational Linguistics, 2006.

[HK09] K. Harbusch and G. Kempen. Generating clausal coordinate ellipsis multilingually: A uniform approach based on postediting. In *Proceedings of the 12th European Workshop on Natural Language Generation*, pages 138–145. Association for Computational Linguistics, 2009.

[KDR08] I.H. Khan, K. van Deemter, and G. Ritchie. Generation of Referring Expressions: Managing Structural Ambiguities. *Proceedings of the 22nd International Conference on Computational Linguistics*, pages 433–440, 2008.

[KM02] D. Klein and C.D. Manning. Fast Exact Inference with a Factored Model for Natural Language Parsing. *Advances in Neural Information Processing Systems 15*, 2002.

[NCLP10] A. Nenkova, J. Chae, A. Louis, and E. Pitler. Structural Features for Predicting the Linguistic Quality of Text. *Empirical Methods in NLG, LNAI 5790*, pages 222–241, 2010.

[Ran04] A. Ranta. Grammatical Framework, A Type-Theoretical Grammar Formalism. *Journal of Functional Programming*, 14(2):145–189, 2004.

[RD97] E. Reiter and R. Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87, 1997.

[RN03] S.J. Russell and P. Norvig. *Artificial Intelligence: a modern approach*. Prentice hall, 2003.

[SC05] A. Strotmann and O. Caprotti. Multilingual access to mathematical exercise problems. In *Electronic Proceedings of the Internet Accessible Mathematical Computation Workshop*. Citeseer, 2005.

[Sla06] N. Slabbers. Narration for Virtual Storytelling. Master's thesis, University of Twente, 2006.

[THH06] M. Theune, F. Hielkema, and P. Hendriks. Performing aggregation and ellipsis using discourse structures. *Research on Language and Computation*, 4(4):353–375, 2006.

[WC98] M. White and T. Caldwell. EXEMPLARS: A practical, extensible framework for dynamic text generation. *Proceedings of the Ninth International Workshop on Natural Language Generation*, pages 266–275, 1998.

Appendix A

Word list

- **Adjectivication**: One of the techniques that is used to add variation to the text. It changes the position of the adjective in the sentence. (Section 5.4.2, p. 74)
- **Aggregation**: One of the techniques that is used to add variation to the text. Multiple sentences can be joined together in one new sentence. (Section 5.4.1, p. 73)
- **Attribute**: A characteristic of an entity, for example: the colour of a bicycle.
- **Attribute value**: One of the possible options for an attribute, for example: white, silver or black.
- **Attribute couple**: A combination of two attributes. Every attribute couple is either dependent, or independent.
- **Context**: The theme or domain of the story of the exercise, for example 'bikes in a store' or 'rooms in a hotel'.
- **Context file**: A file containing all information about a context that is needed in Genpex. (Section 3.3.1, p. 29)
- **Count**: A number to represent the number of entities in the indicated world that have a certain attribute value for an attribute.
- **Disjoint**: In the statement $A \vee B$, the events A and B are disjoint if A and B can never be applied simultaneously to one entity. A statement that is disjoint requires a different required calculations than a statement without disjoint events.
- **Ellipsis**: One of the techniques that is used to add variation to the text. In an aggregated sentences, some duplicated words can be removed. (Section 5.4.6, p. 77)
- **Entity**: A reference to an object in the real world, that has multiple attributes. Every entity has exactly one value for every attribute.

- **Entity substitution**: One of the techniques that is used to add variation to the text. The noun that represents the entity is replaced with the noun (or noun phrase) that represents the attribute value. (Section 5.4.3, p. 75)
- **Event**: In the exercises that we create, an event describes always the situation that a random entity is picked out of a (sub)set and that it has one or more properties (a specific attribute value for an attribute). The questions ask to calculate the probability of events.
- **Exercise**, or **Exercise text**: The text or story in which a probability problem is embedded.
- **Exercise configuration**: The input for the system, designed by the user. It specifies properties that guide the output of the system. Some properties apply on the generation of the probability problem, others on the language generation. In Genpex this information has to be specified via the GUI.
- **Genpex**: The created program, abbreviation for 'Generator for Narrative Probability Exercises'. (Chapter 3, p. 23)
- **Marked word order**: One of the techniques that is used to add variation to the text. The default word order in a sentence can be changed into a different word order, without affecting the meaning of the sentence. (Section 5.4.4, p. 76)
- **NLG**: Natural Language Generation. The process that creates text to express information. (Chapter 5, p. 63)
- **Probability problem**: The formal representation of a mathematical exercise in which a student is required to calculate a probability of a certain event. It is formulated as a list of statements and one or more questions. The system is able to create a textual representation of a probability problem, that can be edited by the user. The system is also able to parse this textual representation. (Chapter 4, p. 37)
- **Question**: The question that is included in the probability problem. To be able to answer this question, a student is required to calculate the probability of a certain event.
- **Sentence tree**, **tree nodes**: A structured representation of the words and clauses in a sentences. The structure is represented by a tree, consisting of tree nodes. (Section 5.2, p. 65)
- **Statement**: An expression in the probability problem that gives the number of entities in the world that have certain attribute values. This is for example the number of entities that have a specific value for a given attribute. (Section 4.2, p. 42)
- **Student**: The person that answers the exercise.
- **Text variation techniques**: All techniques embedded in the microplanner of the NLG-process, that can change the default sentences. Because each variation is applied randomly with a certain probability, these techniques vary the textual output of the system.

- **User**: The researcher that uses Genpex to make new exercises.
- **World**: An (internal) representation of a situation, where all entities are defined, and every attribute of the entity has a value. A part of the world is reflected in the statements of the probability problem.
- **Word value**: The (basic form of) the word that represents a tree node in a sentence tree. This can be the stem of a word, or a word that is not inflected. In the NLG process the word value is changed, to represent the correct inflection, based on the structure of the sentence tree. A word value can also be empty or a punctuation mark.