Performing Syntactic Aggregation using Discourse Structures

Feikje Hielkema
Student Kunstmatige Intelligentie
student nr. 1196499

Graduation committee:

-Dr. Petra Hendriks
-Dr. Mariët Theune

May 2005, University of Groningen

Abstract

This thesis describes an effort to create an NLG-system capable of syntactic aggregation. The NLG-system is part of a Java based story generation system, capable of producing and narrating fairy tales. This thesis focuses on one of the components of the NLG-systems, a Surface Realizer capable of performing syntactic aggregation, and producing varied and concise sentences because of it.
To enable the Surface Realizer to use different cue words when combining two separate clauses, and to choose only appropriate cue words, discourse structures were used to denote connections between clauses, and a cue word taxonomy has been constructed. The Rhetorical Relation between two clauses determines which cue words can be used to aggregate the two clauses, and the selected cue word determines the structure of the generated sentence.
Repetitive parts are removed if allowed by syntactic and semantic constraints, resulting in ellipsis. The result of these two processes is that two clauses, with the same relation between them, can sometimes be realized in as much as five different surface forms, each more concise than the original clauses.

This project expands on earlier work by Faas (2002) and Rensen (2004), who created the Virtual Storyteller, a multi agent NLG-system. The addition of a Surface Realizer, one capable of syntactic aggregation, ensures greater variety in the generated text. However, much work remains to be done, not in the least in implementing two other modules in the Narrator, the Content and the Clause Planner.

Contents

Preface

For nine months I have worked on the problem of generating syntactically aggregated, elliptic sentences. It is still as fascinating to me as it was when I started, if not more. I have enjoyed this graduation project very much, and in a way it seems the hardest thing now is to quit, and to admit that attaining perfection would take another nine months at least.

Many people deserve my gratitude here. Mariët Theune, for meeting me every week, prepared with feedback on things I sent her only the day before, for her patience and for her support; Petra Hendriks, for valuable advice on the nature of ellipsis and what to concentrate on, not losing patience when I failed to stay in contact, and for convincing me to submit an article to a Workshop when I thought nothing would ever come of it; Rieks op den Akker en Dennis Reitsma, for helping me design the Surface Realizer when I didn't have a clue how to go about it.

I also want to thank my boyfriend, Marten Tjaart, for staying with me in Groningen waiting for me to finish college, for always supporting me and telling me, yes, I am sure you can do this. My parents, for giving me the loan of the car every week and sometimes even bringing me to Twente, an hour and a half drive. And, of course, my family and friends, for asking me regularly how it was going, and for not inquiring any further when the answer was a noncommittal 'uh, fine…'. Thank you very much, everyone.

1. Introduction

1.1 Goals

Ellipsis and co-ordination are key features of natural language. For a Natural Language Generation system to produce fluent, coherent texts, it must be able to generate co-ordinated and elliptic sentences. In this project, we set out to answer the question:

How can a grammar for a Natural Language Generation be realized in Java? Specifically, how to realize ellipsis and syntactic aggregation?

The goal is to create a grammar that produced grammatically correct utterances that are varied and compact as a result of syntactic aggregation and ellipsis. This thesis describes an effort to implement syntactic aggregation in the Virtual Storyteller, a Java-based story generation system. As the Virtual Storyteller is java-based, I want to create the grammar in Java as well, to improve compatibility. The focus lies on the generation of co-ordinated and elliptic structures for the Dutch language.
In this thesis, syntactic aggregation is defined as the process of combining two clauses at the surface level using any kind of syntactic structure, for instance co-ordination or a subordinate clause. Two important claims are made:

- The process of syntactic aggregation belongs in the Surface Realizer module of a natural language generation system, because it is a grammatical process and belongs with the other grammatical processes
- The combination of dependency trees and discourse structures constitutes excellent input to the Surface Realizer, because dependency trees are easy to manipulate, and rhetorical relations are necessary to select a correct syntactic structure in the process of syntactic aggregation

Cue phrases are one of the resources of natural language to signal different rhetorical relations, and as such are a vital part of syntactic aggregation. They have great influence on the syntactic structure of an aggregated sentence. For this reason a taxonomy of the most common Dutch cue words has been designed to use in the aggregation process.

The context for our work is the Virtual Storyteller (see section 1.2), a Java-based story generation system. At the moment, the system can create simple fairy tales featuring two characters (a princess, Diana, and a villain, Brutus), with different events and endings.
The Narrator is the module responsible for the generation of the actual text. The initial version of the Narrator only presented the bare facts of the story, mapping the events from the plot to simple, fixed sentences. This made for a very monotone, uninteresting narrative. (*Diana entered the desert. Diana saw Brutus. Diana was afraid of Brutus. Brutus left the desert*. Etc.) Syntactic aggregation should help enormously to improve the liveliness of the generated narratives. Therefore, the goal of the project is to make the Narrator produce at least the following structures:

- Paratactic constructions: *Diana verliet de woestijn en Brutus betrad het bos (Diana left the desert and Brutus entered the forest)*
- Hypotactic constructions*: Diana verliet de woestijn, omdat ze Brutus zag (Diana left the desert, because she saw Brutus)*

- Conjunction Reduction: *Diana ging de woestijn binnen en zag Brutus (Diana entered the desert and saw Brutus)*
- Right Node Raising: *Diana betrad en Brutus verliet de woestijn (Diana entered and Brutus left the desert)*
- Gapping: *Diana verliet de woestijn en Brutus het bos (Diana left the desert and Brutus the forest)*
- Stripping: *Diana verliet de woestijn en Brutus ook (Diana left the desert and Brutus too)*
- Coördinating one constituent: *Amalia and Brutus entered the desert*

I did not include VP-Ellipsis (Diana left the desert and Brutus did, too) as this structure is not allowed in the Dutch language. The structures are explained in section 2.1.7. For the paratactic and hypotactic constructions, different cue words should be available, to improve the variety and to be able to signal different rhetorical relations. Although the work aims in the first place at improving the texts produced by the story generation system, I believe that the approach to syntactic aggregation and ellipsis is sufficiently general to be relevant for all kinds of language generation systems. In addition, it will be argued that the approach is largely language-independent.

This thesis is structured as follows. First the architecture of the Virtual Storyteller is described. Then the related research is discussed in chapter 2. This includes a definition of ellipsis, a study of aggregation and the way it is defined and handled by other NLG-systems and some words on existing Surface Realizers. Meaning-Text Theory and Rhetorical Structure Theory are treated as well, as some of their features were used in the project. Also research in which a cue word taxonomy was created.
Chapter 3 describes the design of the Narrator and presents the cue word taxonomy that we developed for use in the aggregation process in the Virtual Storyteller. It also discusses how we perform aggregation in our system, using this taxonomy.
In chapter 4 the implementation of the Surface Realizer is explained. Chapter 5 shows the results and discusses the problems. Chapter 6 and 7 give a general discussion and a conclusion, and the thesis ends with suggestions about future work.

1.2 The Virtual Storyteller

The Virtual Storyteller is a Java based multi-agent Natural Language Generation system, that produces fairy tales. At the time of writing, these fairy tales are fairly simple, featuring only two characters, a princess and a villain. However, the agents responsible for the plot production and the ontology are under heavy construction, to enable the production of more complicated stories.

The architecture of the Virtual Storyteller is depicted in figure 1.1. The plot is created by so-called Actor agents and a Director agent. The Actors are autonomous computer programs that represent the characters in the story, each with its own personality and goals. They are able to reason logically and are affected by emotions. These emotions, when felt strongly, may cause the agent to adopt a new goal that overrides its original goal. For instance, Diana's original (episodic) goal may be to kill Brutus, but she may flee instead if she is too scared (emotional goal). The Director controls the flow of the story and makes sure it has a decent plot. The

Actor agents will reason which action is best for them to take, but they have to ask the Director for permission to perform it, to prevent too much repetition.
The Virtual Storyteller also contains a Narrator agent and a Speech agent. The Narrator transforms the plot into text. Before this project, the Narrator used simple templates to map the actions and events to text. As an action always mapped to the same line, the resulting text was rather monotone. As said above, this project set out to introduce more diversity in the generated text, and more fluent output.
The Speech agent transforms the text into speech. At the time of writing, it does not use prosodic information, as the Narrator does not generate any. In the future, this would be a useful addition, especially as the generated sentences have become more complicated by the use of aggregation. An evaluation of a storytelling speech generator by Meijs (2004) indicated that listeners showed a preference for speech that included climaxes at key points, over the 'flat' text of a neutral speech generator.

The agents in the framework continuously exchange information. For instance, an Actor will ask permission from the Director for every action he/she wants to perform, and tell the Narrator that the action has taken place, its reason for taking it and its emotions.
Figure 1.2 shows two stories that were generated by the Virtual Storyteller before this project started. For more detailed information on the Virtual Storyteller, read (Faas, 2002) and (Rensen, 2004).

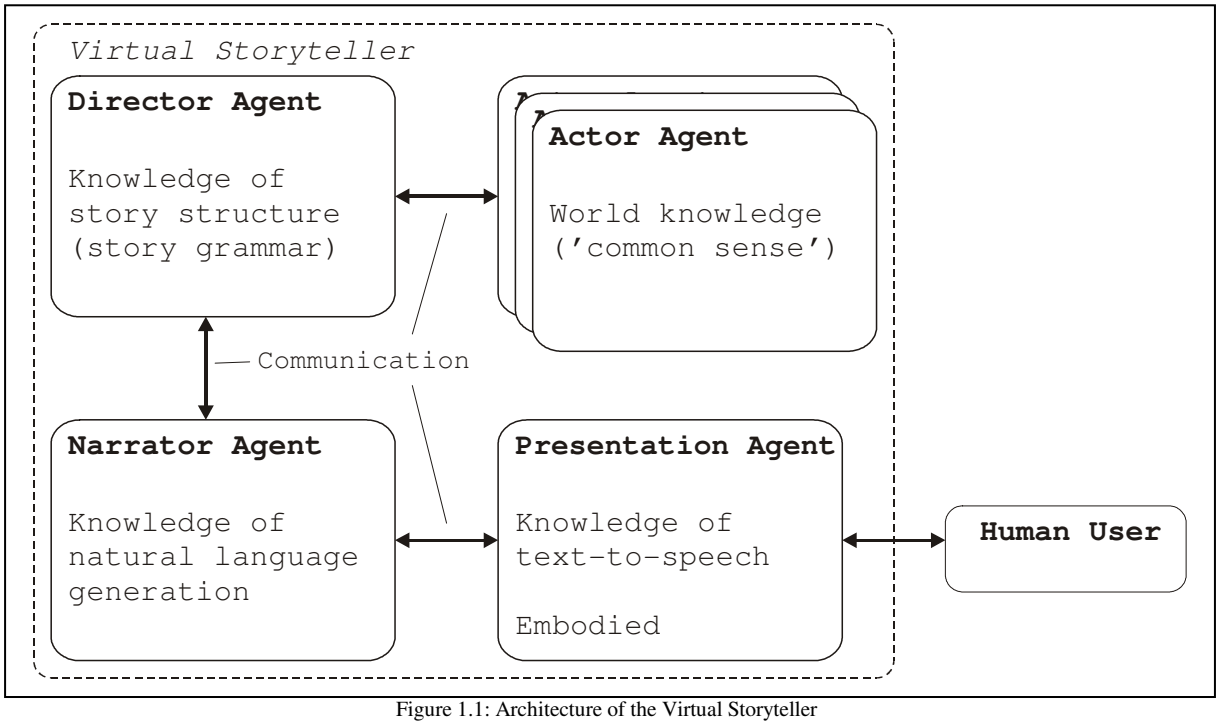


Figure 1.1: Architecture of the Virtual Storyteller

Verhaal 1
Er was eens een prinses. Ze heette Amalia. Ze bevond zich in Het kleine bos.
Er was eens een schurk. Zijn naam was Brutus. De schurk bevond zich in het moeras.
Er ligt een Zwaard. in De bergen.
Er ligt een Zwaard. in Het grote bos.
Amalia loopt naar de woestijn.
Brutus loopt naar de woestijn.
Amalia ervaart angst ten opzichte van Brutus vanwege de volgende actie :
Amalia ziet Brutus
Amalia loopt naar de kale vlakte.
Brutus loopt naar de kale vlakte.
Amalia ervaart angst ten opzichte van Brutus vanwege de volgende actie :
Amalia ziet Brutus
Amalia loopt naar de bergen.
Brutus loopt naar de bergen.
Amalia pakt zwaard op.
Brutus ervaart angst ten opzichte van Amalia vanwege de volgende actie :
Amalia pakt zwaard op.
Brutus schopt de mens.
Amalia steekt de mens neer.
en ze leefde nog lang en gelukkig!!!

Verhaal 2
Er was eens een prinses. Ze heette Amalia. Ze bevond zich in Het kleine bos.
Er was eens een schurk. Zijn naam was Brutus. De schurk bevond zich in het moeras.
Er ligt een Zwaard. in De bergen.
Er ligt een Zwaard. in Het grote bos.
Amalia loopt naar de woestijn.
Brutus loopt naar de woestijn.
Amalia ervaart angst ten opzichte van Brutus vanwege de volgende actie :
Amalia ziet Brutus
Amalia gaat het kasteel binnen.
Brutus gaat het kasteel binnen.
Amalia ervaart angst ten opzichte van Brutus vanwege de volgende actie :
Amalia ziet Brutus
Amalia slaat de mens.
Brutus schopt de mens.
Amalia schreeuwt.
Brutus slaat de mens.
Amalia schreeuwt.
Brutus pakt de mens op.
Amalia schreeuwt.
Brutus ervaart hoop ten opzichte van Amalia vanwege de volgende actie :
Brutus pakt de mens op.
Brutus neemt in het kasteel de mens gevangen.
en de mensen spraken jaren later nog over deze treurnis

Figure 1.2: stories generated by the VS

2. Literature

This chapter is concerned with relevant research in the field of Natural Language Generation. The first section deals with syntactic aggregation and discusses different definitions and the existing approaches to implement it. It also contains a section treating the elliptic structures we want to generate.
In the beginning of this project, I considered three Surface Realizers to see if they could be used in this project, and if their approach would be useful with an eye to syntactic aggregation. These Surface Realizers are discussed in section 2.2.
The chosen input of the Surface Realizer of this project consists of Dependency Trees connected by Rhetorical Relations. Dependency Trees have sprung from Meaning-Text Theory, which is treated in section 2.3. Rhetorical Relations are a creation of Rhetorical Structure Theory, discussed in section 2.4.
The fifth section discusses a Cue Phrase Taxonomy and its possible uses for Natural Language Generation. The last section gives some preliminary conclusions about the nature of aggregation and the decisions that were made about the design of the Surface Realizer, based on the literature.

2.1 Aggregation

To formulate a suitable approach to performing syntactic aggregation, I first investigated how other NLG-systems handle this process. This section describes several approaches toward and definitions of aggregation. First the three-stage pipeline structure of Reiter & Dale (1995) is discussed, a standard architecture for NLG-systems that dictates a place for several processes, aggregation among them. Section 2.1.2 describes an attempt to define the different kinds of aggregation. The following sections show how definitions of the aggregation process vary in the NLG-field, by taking a closer look at several approaches, and by discussing the RAGS project, in which over twenty NLG-systems were analyzed. Finally, the differences are discussed and it is decided what definition will be used throughout this thesis.

2.1.1 Standard pipe-line NLG-architecture

Reiter & Dale (1995) have devised a schema, portraying the common structure of many Natural Language Generation systems. This is a pipe-line structure, a chain of several modules. In the most common architecture, there are three modules: *Document Planning*, *Microplanning* and *Surface Realisation*.
The Document Planning module performs the tasks of Content Determination, deciding what information should be communicated, and Document Structuring, ordering the text. Its output is a Document plan, a tree whose internal nodes specify structural information and whose leaf nodes specify content.
The Microplanner takes the Document plan and performs Lexicalisation (the choosing of appropriate words and phrases), Referring Expression Generation (pronominalization etc) and Aggregation on it. Aggregation entails mapping the output of the Document Planner onto linguistic structures such as sentences and paragraphs. It output is a Text specification, a tree whose internal nodes specify the structure of a text and whose leaf nodes specify the sentences of a text.

Finally, the Surface Realisation module maps Text specification onto an actual text, using syntactic, morphological and orthographical rules.

2.1.2 Reape & Mellish: Just what is aggregation anyway?

Reape & Mellish (1999) have written an article about what aggregation is and why it should be performed. There appeared to be no consensus about the definition of aggregation. As a solution, they propose two definitions of aggregation, a broad and a narrow sense, and claim the confusion is due to these different uses of the same term.

They claim aggregation, in the broad sense, is the combination of two or more linguistic structures into one linguistic structure. This improves sentence structure and construction. In the narrow sense, aggregation is any process which maps one or more structures into another structure which gives rise to text which is more x-aggregated than would otherwise be the case. They define x-aggregated text as text which contains no multiple nonpronominal overt realisations of any propositional content and no overt realisations of content readily inferable or recoverable from the reader's knowledge or the context which are not required to avoid referential ambiguity or to ensure grammaticality.

Reape & Mellish distinguish several sorts of aggregation:

- conceptual: reducing the number of propositions in the message
  For instance, the mapping of dove(x) and sparrow(y) into bird({x,y})
- discourse: reducing the complexity of the text plan, mapping a rhetorical structure to a 'better' structure. For example, mapping two nucleus-satellite relations with the same nucleus to one 'nucleus - &(satellite1, satellite2)' relation
- semantic: semantic grouping and logical transformations
  For instance, mapping 'Chris is Jamie's brother' and 'Jamie is Chris' sister' to 'Jamie and Chris are brother and sister'
- syntactic: subject grouping, predicate grouping, etc.
  For example, mapping 'John is here' and 'Jane is here' to 'John and Jane are here'
- lexical: mapping more lexical predicates to fewer lexemes and to fewer lexical predicates, and mapping more lexemes to fewer lexemes
  For instance, mapping 'monday(x1), ..., friday(x5)' to the lexeme 'weekdays'
- referential: pronominalization
  For example, mapping 'John is here' and 'Jane is here' to 'They are here'

Reape & Mellish point out the lack of linguistic theories in many NLG-systems. Comparing a NLG-system built without linguistic knowledge to a power plant built without knowledge of physics, they state that any successful system which achieves 'aggregated text' will have to incorporate linguistic knowledge about co-ordination, ellipsis etc.

2.1.3 Shaw's approach to aggregation

According to Shaw (2002) aggregation improves the cohesion, conciseness and fluency of the produced text. He distinguishes four types of clause aggregation: interpretive, referential, lexical and syntactic aggregation. Interpretive aggregation uses domain-specific knowledge and common sense knowledge. Referential aggregation uses a reader's knowledge of discourse and ontology. An example is quantification, which replaces a set of entities in the propositions with a reference to their type as restricted by a quantifier. For example, 'the left arm' and 'the right arm' could be replaced with 'each arm'. Lexical aggregation combines multiple lexical items to express them more concisely.

Syntactic aggregation is divided in two constructions, paratactic and hypotactic. In a paratactic construction, two clauses of equal status are linked. In a hypotactic structure, two clauses have a subordinate relationship (nucleus – satellite).
Shaw also discusses some constraints on aggregation. For instance, the aggregated entities will have to be ordered. 'A happy old man' is more fluent than 'An old happy man' (Malouf, 2000).
Shaw, like Reiter & Dale (1995) uses the pipe-line architecture. He focuses on syntactic aggregation, e.g. co-ordination, ellipsis and quantification. In an article on co-ordination and ellipsis in text generation (Shaw, 1998), Shaw describes a Co-ordination algorithm, designed to handle co-ordination and ellipsis. It is divided in four stages; the first three take place in the sentence planner, the last one in the lexical chooser.
First, the propositions are grouped and ordered according to their similarities while satisfying pragmatic and contextual constraints. For instance, days of the week are put in the right order by comparison operators. Second, elements are subjected to the sense and equivalence tests, which test whether two elements have the same surface realization ánd refer to the same element. If they pass both tests, they are marked as recurrent, but not deleted right-away. Third, a sentence boundary is created when the combined clause reaches pre-determined thresholds. The algorithm keeps combining propositions until the result exceeds the parameters for the complexity of a sentence. And last, it is determined which recurrent elements are redundant and should be deleted. At this stage the algorithm uses directional constraints to determine which occurrence of the marked element is truly redundant. For instance, if a slot is realized at the front or medial of a clause, the recurring elements in the slot generally delete forward.

This algorithm manages to produce co-ordinated and elliptic sentences, and gapping. The process partly takes place in the linguistic realizer and uses some syntactic knowledge. In my view however, ellipsis can entail more than just the deletion of recurrent elements. In Stripping, for instance, the elliptic constituents are not so much deleted as replaced (see section 2.1.7). With subject deletion, the main verb sometimes has to be inflected differently. Moreover, not all forms of ellipsis are always allowed (see section 2.5 on Rhetorical Structure Theory).
Another interesting feature of Shaw's work is his use of rhetorical relations. Shaw points out the relation between rhetorical relations and hypotactic constructions. Using rhetorical relations, one can decide which hypotactic structure is appropriate to use. According to Hendriks (2004) the same is true about elliptic structures (see section 2.5.1), but in Shaw's algorithm the generation of ellipsis seems to take place at a later stage, in which the rhetorical relations are no longer used.

2.1.4 Dalianis' approach

Dalianis (1999) equals aggregation with the removal of redundant information in a text. He discerns four types of aggregation. The first is syntactic aggregation, which removes redundant information but leaves at least one item in the text to explicitly carry the meaning. The result can be seen as co-ordination. Syntactic aggregation is carried out by a set of aggregation rules that perform syntactic operations on the text representation.
Second, elision removes information that can be inferred, the meaning remains implicitly. Third, lexical aggregation is divided in two: bounded and unbounded lexical aggregation. Bounded lexical aggregation keeps the meaning of the sentence intact, the aggregated information is retrievable. In unbounded lexical aggregation, there is loss of information. Bounded lexical aggregation chooses a common name for a set with a bounded number of

elements, instead of exhaustively listing them (for example, the weekdays). If there are any residual elements, they are generated using the cue word ‘except’. Unbounded lexical aggregation is performed over an unbounded set of elements and does not employ cue words. An example is the mapping of 'Mary has a cat and Mary has a dog' to 'Mary has pets'. Some information is lost here, as with the conceptual aggregation of Reape & Mellish.
Finally, referential aggregation replaces redundant information with for example a pronoun.
Dalianis takes a four-stage pipeline view of the text generation process. The four stages are Content determination, Text planning, Sentence planning and surface form realization (in that order). He places the aggregation process after text planning, but before sentence planning. Exactly how this fits in with his pipeline view (as there is nothing between the text planning and the sentence planning modules) is not clear. There is also a small aggregation program by his hand available (Astrogen). This program has only two modules, a deep and a surface component. The aggregation is all done in the deep module, but that does not tell us anything about how to place the process between the text and the sentence planning, as both modules are not realized. Dalianis has also looked into discourse structure theory, and using cue words to prevent ambiguity.

2.1.5 Aggregation as an emergent phenomenon
Wilkinson (1995) contests Dalianis’ view of aggregation as the elimination of redundancy. They are not one and the same, because redundancy elimination can also take the form of, for example, pronominalization (I am not sure whether this argument would be recognised by Dalianis, as he considers pronominalization a part of aggregation). In addition, aggregation does not always eliminate redundancy. It may instead affect sentence complexity or the expression of rhetorical relations.
Wilkinson also states that it is difficult to isolate aggregation as a unitary process, as the decision about how to express the aggregated ideas is just a special case of the more general question of how to structure a sentence.
Therefore, the concept of aggregation needs to be revised. According to Wilkinson, aggregation should be seen as an emergent phenomenon. It is not a process, but a characteristic of texts that have been generated properly. So instead of concentrating on aggregation, we should focus on the processes of Semantic grouping, Sentence structuring and Sentence Delimitation, al of which happen at sentence planning level.

Wilkinson’s article is interesting and refreshing, as it gives a completely different view on aggregation, but I do not think I agree with him. These processes from which aggregation emerges, may not happen in every kind of text. Let us look for example at semantic grouping in storytelling. If we let semantic grouping take place in a story plot, we would have to be careful about it. If we let semantic grouping go on unbounded, a probable result would be that first all statements pertaining one character would be grouped together, and after that the statements of another, etc. This could be very confusing for the listener. Some semantic grouping would be desirable, for instance for two characters that are doing completely separate things, but it would have to be strictly controlled. Thus semantic grouping seems to play but a small role in storytelling, and a dangerous one at that.
However, aggregation *is* an integral part of it. Without aggregation, the story would become monotonous and boring. But if I read Wilkinson's article correctly, semantic grouping plays a very important role in the emerging of aggregation, so important aggregation might not be possible without it. This makes me suspect that aggregation is a process in its own right, and more complex than Wilkinson would have us believe.

2.1.6 RAGS: A classification of NLG-systems

Under the RAGS-project (Cahill & Reape, 1999), over twenty NLG-systems were investigated and compared to Reiter & Dale's three-stage pipe-line (Reiter & Dale, 1995). For each system, it was specified at what stage which task occurred. One of the tasks investigated was aggregation.

Cahill & Reape define aggregation as 'any process of putting together more than one piece of information, which at some level are separate'.

For some systems, the authors had trouble localising the aggregation process, but overall there is obviously no consensus about where aggregation should take place. There are several systems where aggregation occurs in the surface realizer, several where it occurs in the sentence planner, or both, and some that place aggregation in the content planner. This discrepancy is closely connected with the different goals and input of the systems, as some text just needs to be more heavily aggregated than other text. For instance, written text can be more aggregated than spoken language. The only conclusion Cahill & Reape draw about aggregation and where this process should be placed is that aggregation always occurs in interchangeable or adjacent order with segmentation. Segmentation involves the dividing up of information or text into sentences and paragraphs, and is thus connected with aggregation, a seeming opposite.

I have tried to look up the literature on those systems where aggregation is performed in the surface realizer, in particular EXCLASS (Caldwell & Korelsky, 1994) and KPML (Teich &Bateman, 1994) (Bateman & Teich, 1995). I was unable to discover how aggregation was handled in KPML (it was only mentioned twice, and then it referred to semantic aggregation). EXCLASS makes use of the lambda calculus. Ellipsis is realized by two basic rules:

$(A * B) \& (A * C) \rightarrow A * (B \& C)$
$(A * C) \& (B * C) \rightarrow (A \& B) * C$

The first rule states that if the first wordgroup in both clauses is the same, then the last wordgroup in the clauses should be combined. The second rule treats an identical second wordgroup similarly. This implementation of ellipsis is not unlike Shaw's co-ordination algorithm (see section 2.1.3).

As to the right place for ellipsis and gapping, the literature seems to agree that it is a part of syntactic aggregation, even if there is no agreement on the right place for syntactic aggregation.

2.1.7 Ellipsis

This section describes the linguistic phenomenon ellipsis. The first part deals with a definition of ellipsis, the second with the frequencies of different elliptic structures in natural language.

2.1.7.1 An analysis of ellipsis

Quirk et al. (1985) gives an extensive analysis of the linguistic phenomenon ellipsis. Ellipsis is a form of omission: superfluous words, whose meaning is understood or implied, are omitted. What distinguishes ellipsis from other kinds of omission, according to Quirk, is the principle of *verbatim recoverability*. The actual words whose meaning is understood or implied must be recoverable. However, the boundaries are unclear and there are several degrees of ellipsis. Quirk distinguishes five criteria which the strictest form of ellipsis must satisfy:

1. The elliptic words are precisely recoverable

2. The elliptical construction is grammatically 'defective'
3. The insertion of missing words results in a grammatical sentence
4. The missing words are textually recoverable
5. The missing words are present in the text in exactly the same form

An example of the strictest form of ellipsis is 'I'm happy if you are [happy]'. Further, Quirk distinguishes several other, less strict forms of ellipsis which satisfy a few of the above criteria. One such is standard ellipsis, which satisfies only the first four criteria. For example: 'She sings better than I can [sing]'. Here, the elliptic word does not have exactly the same form as its antecedent.
Another category is quasi-ellipsis, which is closely linked to pro-form substitution. For example: 'Our house is different from theirs [their house]'. Here the substitute form is a grammatical variant of the word or construction which appears in the replaced expression.

2.1.7.2 Desired forms of ellipsis
This work will focus on a few elliptic structures that are very prevalent in the Dutch (and English) language. They are:

- Conjunction Reduction: *Amalia betrad de woestijn en zag Brutus (Amalia entered the desert and saw Brutus)*

In Conjunction Reduction, the subject of the second clause is deleted. In the example above, that is 'Amalia'.

- Right Node Raising: *Amalia betrad en Brutus verliet de woestijn (Amalia entered and Brutus left the desert)*

In Right Node Raising, the rightmost string of the first clause is deleted. In the example, the ellipted string is a locative (the desert), but it could also be a direct object, as in 'Amalia slaat en de Prins schopt Brutus' (Amalia hits and the Prince kicks Brutus), or any string, as long as it is in the rightmost position of the first and second clause. A sentence with this construction cannot have a causal reading.

- Gapping: *Amalia verliet de woestijn en Brutus het bos (Amalia left the desert and Brutus the forest)*

In Gapping, the main verb is deleted (in the example above, 'verliet' or 'left'). A gapped sentence cannot have a causal reading either.

- Stripping: *Amalia verliet de woestijn en Brutus ook (Amalia entered the desert and Brutus did too)*

In Stripping, all constituents but one are deleted, and replaced by the word 'ook' (too). This can happen with any constituent, but again only if the sentence does not have a causal meaning.

- Coördinating one constituent: *Amalia and Brutus entered the desert*

Here, one constituent (the subject in this example) is co-ordinated. This can happen with any constituent, but again not in a causal relation.

Any combinations of the first three structures should be generated as well (such as *Amalia gave Brutus a kick and the prince a kiss,* which is both gapped and conjunction-reduced, as both verb and subject are deleted in the second conjunct), as well as co-ordinating single constituents (*Amalia kicked and cursed Brutus*).

2.1.8 Discussion

The above clearly indicates that the literature on aggregation does not agree on a single definition. Though some attempts have been made to sum up the work done in the field (RAGS) and to posit a definition of aggregation (Reape & Mellish, 1999), in practice most projects use their own definition.

The terminology is confusing, as sometimes the same process has different names, or different processes go by the same name. Sometimes the aggregation process is very extensive, encompassing pronominalization and some content determination (as for instance in Reape & Mellish's discourse aggregation), while according to Wilkinson it is not a process at all.

In Dalianis' types of aggregation with respect to the analysis of Reape & Mellish there is no straight mapping from one to the other. His syntactic aggregation and his lexical aggregation seem to match theirs, and his unbounded lexical aggregation matches the conceptual aggregation. Elision seems to be subsumed by semantic aggregation, matching the logical transformations, and bounded lexical aggregation seems to be subsumed by lexical aggregation. It is not clear how Reape & Mellish's discourse aggregation comes into it.

With Shaw, interpretive aggregation seems to accord with Reape & Mellish's conceptual aggregation. It is the reducing of propositions using language-independent knowledge, such as common sense. Referential aggregation is more difficult to define in the terms of Reape & Mellish. It seems to encompass their referential aggregation (Shaw calls it referring expression generation, after Reiter & Dale (2000)), but also parts of discourse and semantic aggregation. For Shaw, referential aggregation seems to be the reducing of propositions using language-dependent information, such as semantics. Lexical and syntactic aggregation, finally, seem to match well again with their namesakes in Reape & Mellish's analysis.

What definition, then, should I work with? The emphasis of my work lies on the phenomenon of ellipsis, which (at least there is consensus on *that*) is a part of syntactic aggregation. I will therefore at least require a definition of syntactic aggregation.

For this project, I will define syntactic aggregation as the process of combining two clauses at the surface level using any kind of syntactic structure, for instance ellipsis, co-ordination or a subordinate clause.

I'll further recognise conceptual, discourse, referential and lexical aggregation as defined by Reape & Mellish. As mentioned above in the paragraph on Wilkinson, I do not think semantic grouping a important process in storytelling, which leaves semantic aggregation only with logical transformations.

As a whole, I define aggregation as the process that removes information that can be found in, or inferred from, the residual text, the ontology or common knowledge. I will work solely on syntactic aggregation.

As syntactic aggregation deals solely with grammatical processes, the most logical place for it is the Surface Realizer. This is the module that concerns itself with all grammatical processes, such as linearization (ordering the words of the Surface Form). To generate a 'Right Node Raising' structure, it may be necessary to know the surface order of the constituents, as only the last string can be deleted. So it was decided to perform syntactic aggregation in the Surface Realizer. The next section describes several existing Surface Realizers.

2.2 Surface Realizers

The conclusion of the last section was that syntactic aggregation should be placed in the Surface Realizer. Much work has already been done in the field of Natural Language Generation and surface realisation. Some Surface Realizers are freely available for academic purposes and as there is no point in reinventing the wheel, I tried to see whether I could use one of them for my purposes. Below is a list of those I considered.

2.2.1 RealPro

RealPro (Lavoie & Rambow, 1997) works in a Java-environment, is freely available to all scientific purposes, and is supposed to be relatively easy to work with. It is based on Mel'cuk's Meaning-Text Theory (Mel'cuk, 1988 – see also section 2.3). In MTT, the sentence at the syntactic level is represented by a DSyntS (Deep Syntactic Structure). This is a dependency tree with linearly un-ordered nodes, meaning that the nodes are not ordered by word order or argument order, but randomly. The nodes, labeled with lexemes, represent content words and their arcs represent the deep-syntactic relations that hold between the content words (for more information on MTT and Dependency trees, see the section below).
RealPro takes such a DSyntS as input. Its output is a fully grammatical sentence at surface level. The transition between input and output is accomplished in several stages:

1. Pre-processing
   Default features are filled out. For instance, if the tense of a verb has not been specified, its default feature is present
2. Deep Syntactic Component
   Transforms the DSyntS to a SSyntS (Surface Syntactic Structure). The SSyntS has function words (e.g. some) and specialized labels
3. Surface Syntactic Component
   Linearizes the SSyntS, determines the order of the words. Its output is an DMorphS (Deep Morphological Structure)
4. Deep Morphological Component
   Transforms the DMorphS to a SMorphS (Surface Morphological Structure) by inflecting the words using morphological processing
5. Graphical Component
   Applies orthographic processing, its output is a DGraphS (Deep Graphical Structure), complete representation of the sentence
6. Post-processing
   Converts the DGraphS to standard output like HTML

The components consist of rules, which are also linearly un-ordered. An example from the Deep Syntactic Component is:

DSynt-Rule
[(X I Y)] | [(X [class: verb])] <=> [(X predicative Y)]

This rule replaces the I-relation between X and Y at DSyntS level with the predicative (that is, subject) relation at SSyntS level, if X is a verb.

Because its input is language-independent, RealPro has been used for Machine-Translation purposes (Lavoie et al., 1999). This bodes well for any endeavours to implement a Dutch grammar.
Of course, there is also a downside to RealPro. First, contrary to KPML (see below) it does not have a Dutch grammar. That would have to be written from scratch. However, this is still preferable to writing the whole program.
Second, the DSyntS representation is not very abstract. According to Reiter & Dale (Reiter & Dale, 2000), this makes RealPro easy to understand and to work with. As a consequence though, the microplanner, which generates the DSyntS, has a lot of work on its plate.

2.2.2 KPML

Another freely available surface realizer is KPML (Teich &Bateman, 1994; Bateman & Teich, 1995). KPML is based on Systemic Functional Grammar, and has a Dutch grammar. It accepts several types of input, with both semantic and syntactic information. According to Reiter & Dale (2000) one of KPML's great strengths is its ability to accept a mixture of meaning specifications, lexicalised case frames and abstract syntactic structures in its input.
KPML is written in Lisp. Unfortunately, this clashed with my intention to build the system in Java, like the other agents of the Virtual Storyteller. Otherwise, KPML might have been very useful.

2.2.3 Exemplars

Exemplars (White & Caldwell, 1998) is an object-oriented, rule-based framework designed to support practical, dynamic text generation. Its emphasis is on ease-of-use, programmatic extensibility and run-time efficiency. It is a hybrid system, that uses both NLG-techniques and templates. It also has some novel features in comparison to other hybrid systems, such as a Java-based definition language, advanced HTML/SGML support and an extensible classification-based text planning mechanism. It makes use of exemplars, schema-like text planning rules meant to capture an exemplary way of achieving a communicative goal in a given communicative context, as determined by the system designer. Each exemplar contains a specification of the designer's intended method for achieving the communicative goal. These specifications can be given at any level.
Exemplars has been used with success in at least three projects at CoGenTex and the makers suggest it is well suited for mono-lingual generation systems. Though I have no intention of working with templates, Exemplars is a good example of the uses they can be put to, in combination with other NLG-techniques.

I do think RealPro would have been very suitable for the purposes of this project. Unfortunately, no permission was given to use RealPro, so in the end I decided to develop a new Surface Realizer.

2.3 Meaning-Text Theory

Although it was impossible to use RealPro, it was decided to use similar input as RealPro does, i.e. Dependency Trees. Dependency Trees seemed very promising with respect to the generation of ellipsis, because of their dependency labels and because they are easy to

manipulate. In this section we discuss Dependency Trees and the theory that produced them, Meaning-Text Theory.

2.3.1 Meaning-Text Theory

In contrast to popular linguistic theories based on constituency grammar, Mel'cuk (Mel'cuk, 1988) defends an approach based on Dependency Syntax. Dependency syntax, he says, is not a theory but a tool. Dependencies are much better suited to the description of syntactic structures than constituency is. They concentrate on the binary relationships between syntactic units. Because linear order of symbols is not supposed to carry any semantic or syntactic information in MTT, the nodes in dependencies are not linearly ordered. The type of the syntactic relations is specified in detail, and so all information is preserved through labeled dependencies.

Mel'cuk designed a Meaning-Text Model, a model that maps the set of Semantic Representations onto the set of its surface structures and vice versa. A MTM includes six major components: the semantic, deep-syntactic, surface-syntactic, deep-morphological, surface-morphological and deep-phonetic components. The middle four of these match the middle four of those components of RealPro, described above. Each component transforms a representation at one level to a representation at a lower level, thus introducing more and more language-specific information. The Semantic structure, the highest level, incorporates no language-specific information at all.

2.3.2 Dependency trees

According to Skut et al. (1997) NLP-applications should not depend on word order. Free word order languages, such as Russian, have a lot of discontinuous constituency types and word order variation. The local and non-local dependencies, the discontinuous constituency types, cause a sentence of a free word order language to be much harder to annotate.

An alternative solution is to focus on argument structure. In Meaning-Text theory (Mel'cuk, 1988), dependency trees are constructed on the basis of predicates and arguments. There is no dependency on linear word order and no limit to the amount of children a node can have. Thus the trees are able to handle word variation easily. Also, as they incorporate no language-specific grammar, they translate well over different languages. They are not only suitable for NLP-applications, but for NLG-systems as well. RealPro, a surface realizer based on Meaning-Text Theory, has succesfully been used for Machine Translation (Lavoie et al, 1999).

This project focuses on ellipsis and gapping, features which are closely connected to word displacement and the like. Dependency trees seem an eminently suitable medium to realize these features.

Dependency trees are based on predicate-argument relations. A node is labeled with a lexeme, and has relations with one or more arguments. These relations are language independent, and take several forms. For instance, the verb 'to like' will have a Subject and an Object relation (not agent-patient, as those are semantic categories), and optionally an Attribute, Coordinative or Appendency relation. Sentence (1) and figure 2.1 give an example.

(1) Paul likes Cathy a lot

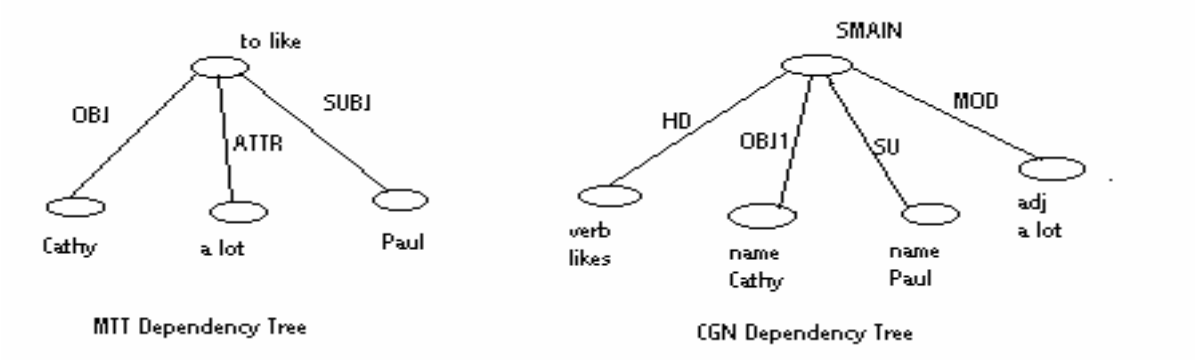


Figure 2.1

Note that, as linear word order is irrelevant to the dependency tree, the relations are not ordered in any way.

For examples on dependency trees for Dutch, the Spoken Dutch Corpus (CGN: Wouden et al., 2002) has thousands. They are structured a little differently (see figure 2.1), but based on the same principles.

2.3.3 Alpino

Alpino is a computational analyser of Dutch which aims at accurate, full parsing of unrestricted text (Bouma, Van Noord & Malouf, 2001). The grammar produces dependency structures, which its creators feel provides a reasonably abstract and theory-neutral level of linguistic representation. Alpino's aim is to provide computational analysis of Dutch with coverage and accuracy comparable to state-of-the-art parsers for English.
Parsing is the very reverse of generation, and has its own very different set of problems. Still, Alpino's dependency trees are capable of representing most Dutch utterances and so would provide an excellent input to the Surface Realizer of the Virtual Storyteller.

2.4 Rhetorical Structure Theory

An important part of aggregation is the combination of two clauses to one sentence. There are different ways to combine two clauses, and different phrases that can be used to connect them, but not all phrases and structures are appropriate in each situation. We need a way to signal what kind of connection exists between clauses, and let that connection then determine which phrases are suitable. Rhetorical Structure Theory provides us with a way to signal such connections. This section deals with Rhetorical Structure Theory and its uses in Natural Language Generation.

2.4.1 Rhetorical Structure Theory

According to Mann & Thompson (1987) Rhetorical Structure Theory is a descriptive framework for text, identifying hierarchic structure in text and describing the relations between text parts in functional terms. It provides a general way to describe the relations among clauses in a text, whether or not they are grammatically or lexically signalled. RST is often used in linguistic issues, and sometimes in Natural Language Processing as well.
In RST, relations are defined to hold between two non-overlapping text spans (an uninterrupted linear interval of text), called *nucleus (N)* and *satellite (S)*. A relation definition

consists of constraints on N, S and the combination of both, and of the effect of the relation. Examples of relations are Circumstance, Elaboration, Evidence and Justify.
Schema's define the structural constituency arrangements of texts. They are abstract patterns consisting of a small number of constituent text spans, relations between them and specification how certain nuclei are related to the whole collection. There are five kinds of schema's: contrast, sequence, joint, circumstance and motivation.
A structural analysis of a text is a set of schema applications that is complete, connected, unique and adjacent. This causes RST-analyses to be trees. An example of an RST-analyses is given in figure 2.2, taken from (Mann & Thompson, 1987) on the following text:

1. The next music day is scheduled for July 21 (Saturday),
   noon-midnight.
2. I'll post more details later,
3. but this is a good time to reserve the place on your calendar.

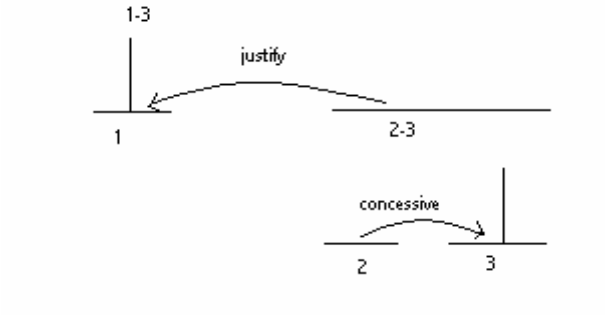


Figure 2.2

The last lines justify the first line, because they make clear why the date of the next music day was mentioned. The second line concedes something to the third, namely that details will have to follow.

2.4.2 RST in NLG
In Natural Language Generation, RST has often been used in Document Planning (Reiter & Dale, 2000). Shaw (2002) uses RST for the aggregation process. He states that propositions in themselves often cannot carry all of the relevant meaning. Look for instance at the two propositions:
-John abused the duck.
-The duck buzzed John.
The two clauses do not make clear who started the unpleasantness, but that information is very important for the decision of how to aggregate the propositions. 'John abused the duck that had buzzed him' means something different than 'The duck buzzed John who had abused him'.
Therefore, according to Shaw, we should specify the rhetorical relationships between propositions, so that different aggregation operators can be selected to realize them.
Hendriks (2004) showed that rhetorical relations hold as well for certain elliptic structures, such as gapping: a gapped sentence cannot have a causal relation between its clauses, but only a Resemblance relation, such as Additive or Contrast. The following sentences were taken from Levin & Prince (1986):

1) Sue became upset and Nan became downright angry
2) Sue became upset and Nan downright angry

The first sentence has two readings, a symmetric and asymmetric reading. In the symmetric reading, both conjuncts are independent events. In the asymmetric reading, the first event has caused the second event. The second sentence, which is gapped, has only the symmetric reading. A gapped sentence can only communicate a Resemblance relation. So if we want to communicate a Causal relation, we cannot use a gapped sentence. This means that we can use rhetorical relations to determine which elliptical structure is suitable as well.

It should be possible to implement some rhetorical relations in the input to the Surface Realizer of the Virtual Storyteller as well. A Dependency Tree has labeled relations between its nodes. All that needs to be done is to specify certain labeled relations that can link the top nodes of two Dependency Trees, which correspond to the rhetorical relations we want to implement.
In the next section I will describe some research into the nature of cue phrases or discourse markers, and their role in signalling rhetorical relations.

2.5 Cue phrases

There are several words and phrases we can choose to connect two clauses or sentences. Most of these phrases communicate some meaning themselves (such as 'because' or 'that's why'), which means that not every cue phrase is appropriate in every situation. How do cue phrases influence the human processsing of natural language, and can we use this to facilitate syntactic aggregation in a NLG-system? This section deals with some research on this topic.

2.5.1 Cue phrases in text processing

Do rhetorical relations play a part in human text processing? Sanders & Noordman (2000) investigated how some aspects of rhetorical, or coherence, relations influence text processing. First, they found that the type of coherence relation that a segment has with the preceding text, has some influence. In an experiment it appeared that causal problem-solution relations were processed faster and recalled better than additive list relations. Sanders & Noordman offer as a possible explanation the view that readers tend to try to relate events to their causes. Thus, causal relations would be preferable to additive relations.
The second conclusion they reached was that linguistic markers facilitate the encoding of the coherence relations between two text segments. A segment is processed faster if it is connected to the preceding text by a linguistic marker (also called discourse marker or cue phrase). However, linguistic marking was not found to influence recall. Sanders & Noordman conclude that the influence of linguistic marking decreases over time, and so differs from the influence of coherence relations, which stays strong. They add that the effect they found of relational markers on reading times is consistent with other literature, such as Britton et al. (1982).

Apparently, linguistic markers or cue phrases are a useful linguistic feature in the processing of text, and should therefore be important in the generation of language.

2.5.2 A Coherence Relation Taxonomy

In 'Toward a Taxonomy of Coherence Relations', Sanders, Spooren & Noordman (1992) propose a method to create a taxonomy of coherence relations. Rhetorical Structure Theory is

a descriptive framework for the organization of text, but lacks psychological plausibility. Other research has suggested that coherence relations are more than analytic tools, that they are psychological entities. But the choice for RST's particular set of coherence relations has no theoretical basis.
Sanders et al. argue that the set of coherence relations is ordered and that readers use a few cognitively basic concepts to infer the coherence relations. They derive a coherence relation out of a composite of such basic concepts. An argument in favour of this view is that one linguistic marker can express only a limited set of coherence relations. Therefore there must be similarities between coherence relations and so they must be decomposed into more basic elements.
Sanders et al. try to categorize coherence relations using the *relational criterion*. A property of a coherence relation satisfies the relational criterion if it concerns the extra information that the coherence relation adds to the interpretation of the isolated discourse segments. It focuses on the meaning of the relation, not on the meaning of each specific segment. They selected four primitives that satisfy the criterion:

- Basic operation: the primary distinction in the taxonomy is between causality and addition. An additive operation exists if P & Q can be deduced. Deducing $P \rightarrow Q$ is necessary, but not sufficient for a causal relation; the antecedent needs to be *relevant* to the conclusion. If both relations hold, the most specific should be selected (i.e. causal)
- Source of Coherence: a relation is semantic if the discourse segments are related because of their propositional content. The state of affairs referred to in P is the cause of the state of affairs referred to in Q. As example is given: 'The unicorn died because it was ill'.
  A relation is pragmatic if the discourse segments are related because of the illocutionary meaning of one or both segments. The coherence relation concerns the speech act status of the segments. Example: 'John is not coming to school, because he just called me'.
- Order of the Segments: In the basic operation '$P \rightarrow Q$', if S_1 expresses P the order is basic, if S_2 expresses P the order is non-basic. As additive relations are symmetric, this primitive does not discriminate between additive relations.
- Polarity: if S_1 and S_2 express respectively P and Q, a relation is positive. Otherwise, if P or Q is expressed by $\neg S_1$ or $\neg S_2$, the polarity is negative. For example: 'Although he did not have any political experience, he was elected president.' The basic operation here is '$P \rightarrow Q$' (if he has no political experience, he will not be elected), but Q is represented by $\neg S_2$.

By combining these primitives, a taxonomy of twelve classes of coherence relations was constructed (figure 2.3). Two experiments were performed to test the taxonomy. In the first, a group of discourse analysts had to choose coherence relations (out of a list) for sentence pairs. It was found that the subjects' classification agreed considerably with the classification in the taxonomy. When there was disagreement, subjects chose a related class. There was, however, a lot of confusion between pragmatic and semantic relations over the whole range of classes.

Basic Operation	Source of Coherence	Order	Polarity	Class	Relation
Causal	Semantic	Basic	Positive	1	Cause-consequence
Causal	Semantic	Basic	Negative	2	Contrastive cause-consequence
Causal	Semantic	Nonbasic	Positive	3	Consequence-cause
Causal	Semantic	Nonbasic	Negative	4	Contrastive consequence-cause
Causal	Pragmatic	Basic	Positive	5a	Argument-Claim
				5b	Instrument-Goal
				5c	Condition-consequence

Causal	Pragmatic	Basic	Negative	6	Contrastive argument-claim
Causal	Pragmatic	Nonbasic	Positive	7a	Claim-argument
				7b	Goal-instrument
				7c	Consequence-condition
Causal	Pragmatic	Nonbasic	Negative	8	Contrastive claim-argument
Additive	Semantic	--	Positive	9	List
Additive	Semantic	--	Negative	10a	Exception
				10b	Opposition
Additive	Pragmatic	--	Positive	11	Enumeration
Additive	Pragmatic	--	Negative	12	Concession

Figure 2.3: taxonomy of coherence relations from Sanders et al.

A second experiment was made investigating whether people are able to infer the coherence relations between sentences and to express them by appropriate linguistic devices, such as connectives. This time the subjects were students, who were told to choose a connective (again, out of a given list) for a sentence-pair. The chosen connective was compared to the original connective. Here, too, there was considerable agreement between the chosen connective and the original. Again, there was least agreement concerning connectives that differ only in the Source of Coherence.
Thus, according to Sanders et al., the taxonomy and its primitives are supported by the experimental results and therefore the psychological plausibility of the primitives is supported.

I have some reservations with these results. First, both experiments indicate some confusion concerning the 'Source of Coherence' primitive. Sanders et al. admit that this primitive is the most dubious, but argue that the confusion might be due to the lack of context in the experiment material. I am inclined to think that 'Source of Coherence' should not be a primitive; while the distinctions made by 'Polarity' and 'Basic Operation' are intuitively clear, this is not the case with 'Source of Coherence', at least not to me. It may be different for practiced discourse analysts. Which brings me to my second point: the first experiment was conducted with discourse analysts as its subjects. It seems to me that a practiced discourse analyst may be inclined to think along the same lines, and use the same distinctions, as the authors of this article. When deciding on the proper relation, the analysts may have not so much used their intuition but their training, and so would tend to agree with a taxonomy constructed using similar training.
Still, the second experiment indicates that when students (who hopefully had not received any training in discourse analysis) were used, the taxonomy was still largely supported. Also, the distinctions made by the other primitives are intuitively clear. I think that, with more research, some more primitives but without Source of Coherence, a taxonomy of coherence relations could be created that encompasses all rhetorical relations and has great psychological plausibility.

2.5.3 A Cue Phrase Taxonomy

Knott & Dale (1993) remark that Rhetorical Structure Theory lacks a uniform and commonly used set of relations. This has resulted in a proliferation of rhetorical relations, as every researcher creates a set suitable to his needs. Unfortunately, if any rhetorical relation one takes a fancy to may be added to the set, this reduces any explanational power of RST considerably, and makes it impossible to falsify. Thus, Knott & Dale argue, the constraints on relationhood need to be tightened.

Like Sanders, Spooren & Noordman (1992), Knott & Dale view coherence relations as psychological constructs, which people use when creating and interpreting text. Knott & Dale argue that, if people use a certain set of relations in constructing and interpreting text, then it is quite likely that the language should have the resources to signal these relations explicitly. The most obvious means to signal relationships are cue phrases (sometimes referred to as discourse markers). These cue phrases can be used as an objective measure to determine "the" set of rhetorical relations. They gathered a corpus of cue phrases and classified them according to their function in discourse, using a substitutability test. Put simply, this test is used to determine whether two cue phrases signal (partly) the same features, by checking whether one can be substituted by the other in a particular context. Look at the example in figure 2.4 (taken from Knott & Dale, 1996). In the first example, 'Whereas' can be substituted by 'On the other hand', but not by 'Then again'. In the second example it is the other way around. 'Then again' thus signals different features than 'Whereas'. However, 'On the other hand' can figure as a substitute in both example. Apparently 'On the other hand' signals only those features that 'Whereas' and 'Then again' have in common.

Kate and Sam are like chalk and cheese. Sam lives for his books;	whereas +on the other hand *then again	Kate is only interested in martial arts.
I don't know where to eat tonight. The Star of India is always good;	then again, + on the other hand * whereas	we had curry just the other night.

Figure 2.4: substitution test (Knott & Dale, 1996)

On this basis a taxonomy of cue phrases was created. This taxonomy was hierarchal, as some cue phrases are more specific than others, that is, signal more features. The cue phrases that are used most in language, seem to be the phrases that are least specific. This makes sense if you consider that a very specific cue phrase can always be substituted by a more general one, but not the other way around. Also, it may be convenient for the speaker not to explicitly signal features that can be easily deduced by or are already known to the speaker.

In 'The Classification of Coherence Relations and their Linguistic Markers: An Exploration of Two Languages' (Knott & Sanders, 1998), the similarities between the work of Knott & Dale and Sanders et al. are discussed. An attempt is made to create a similar taxonomy for Dutch cue phrases, using the cognitive primitives that were proposed by Sanders et al. Some new features are added in the process, such as Volitionality. Volitionality can distinguish non-volitional cue-words (daardoor, doordat) from volitional (daarom, om die reden).
The taxonomy is hierarchically structured. In terms of features, the following definitions are given:

- X is synonymous with Y if they signal identical features
- X is exclusive with Y if they signal different values of some feature
- X is a hyponym of Y (and Y a hypernym of X) if X signals all features that Y signals, and some other feature as well, for which Y is undefined
- X and Y are contingently intersubstitutable if X and Y signal some of the same features, but in addition X is defined for a feature for which Y is undefined, and Y is defined for a feature for which X is undefined.

The following notation is used:

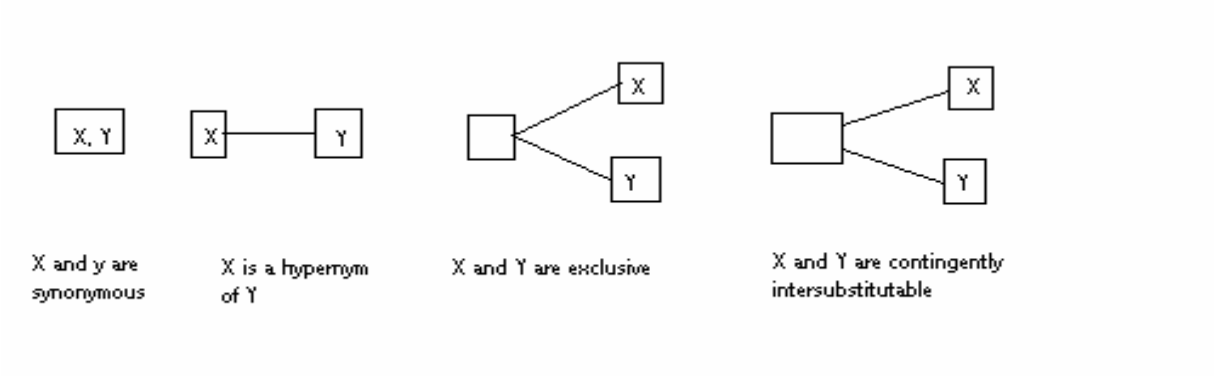


Figure 2.5

The taxonomy for Dutch cue-words is given in figure 2.6.

The idea of cue phrases as an objective measure to identify different rhetorical relations is very attractive. However, the taxonomy that was created for a (small) part of the Dutch cue phrases, already looks very convoluted and complicated. Moreover, Knott & Sanders readily admit that this taxonomy was created using the cue phrases that were easiest to classify; other cue words will be even harder to classify and cause the taxonomy to be even more of a labyrinth. Though it may be possible to create a less convoluted taxonomy, I suspect that the principles used must be changed, as I have argued earlier in the section on the coherence relation taxonomy.
I conclude that, though I agree that cue phrases could be an objective measure with which to define rhetorical relations, the preliminary taxonomy of Knott & Sanders is too convoluted to be easily implemented. Using different principles to distinguish different categories might cause the taxonomy to be clearer. For the purpose of the Virtual Storyteller, a small taxonomy charting only the most prevalent cue words in Dutch, might be constructed. The principles used to distinguish different categories should be principles that are easy detectable by the Storyteller algorithm.

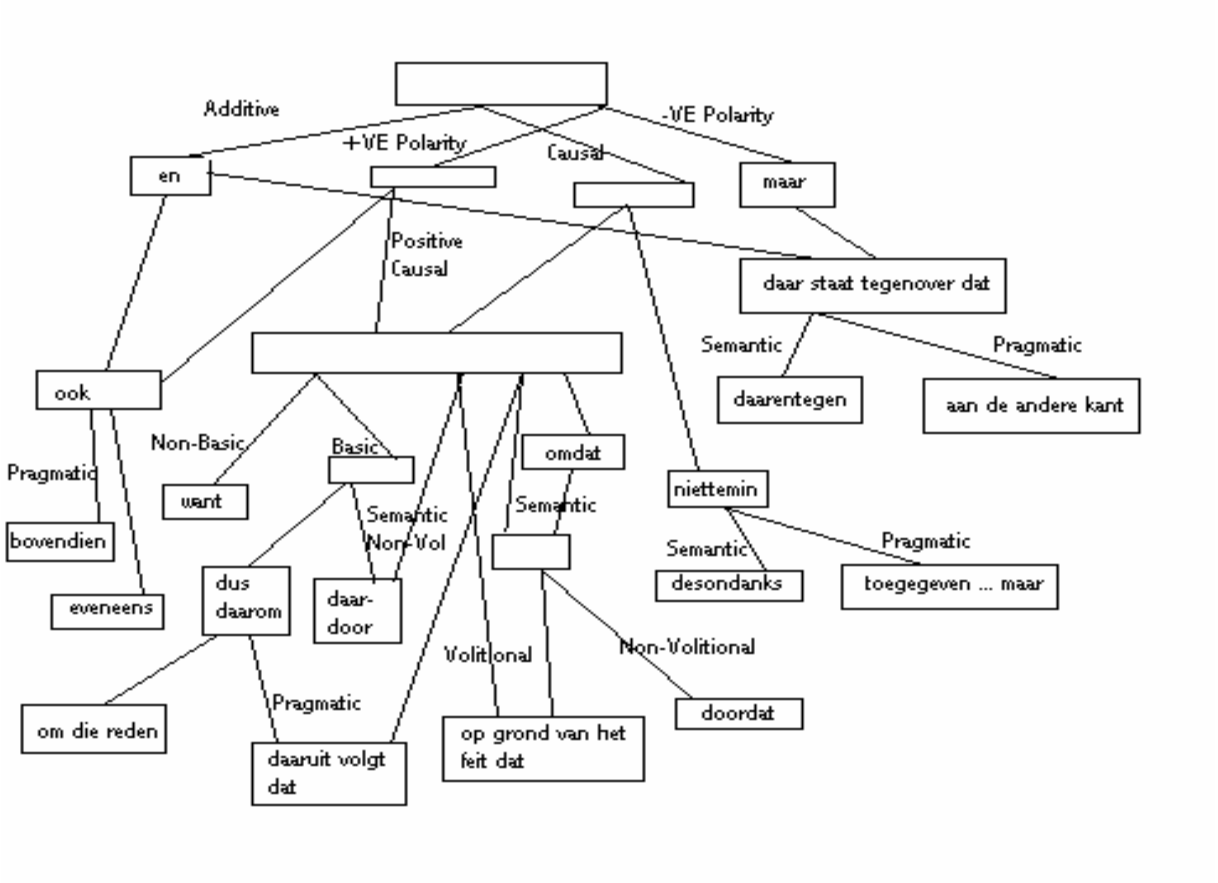


Figure 2.6: Taxonomy of Dutch cue phrases (Knott & Sanders, 1997)

2.6 Conclusions

Reiter & Dale (2000) see aggregation as a Microplanning task. The RealPro architecture presupposes a microplanner that performs all aggregation and ellipsis tasks as well. The RAGS-project (Cahill & Reape, 1999) showed a lack of consensus on the location of the aggregation process in the NLG pipe-line; instead the situation varied widely over different NLG-systems. Shaw (1998) places aggregation partly on the level of the Surface Realizer (called Lexical Realizer in his algorithm) and partly on the level of the Sentence Planner. Dalianis (1999) places it somewhere between the Document Planner and the Sentence Planner. Of course, this divergence is partly caused by the fact that many, quite different processes are gathered under 'aggregation' (see Mellish & Reape, 1999).
This project only deals with syntactic aggregation. For this project, syntactic aggregation is defined as the process of combining two clauses at the surface level using any kind of syntactic structure, for instance ellipsis, co-ordination or subordinate clause. As syntactic aggregation is concerned with grammatical processes (co-ordinating sentences and deciding which elements can be deleted without rendering the sentence ungrammatical), in our view it should be situated with the other grammatical processes, in the Surface Realizer.

The Surface Realizer receives as input a 'Rhetorical Dependency Graph': a graph specifying the rhetorical relations between sentences, represented by Alpino Dependency Trees. The fact that these trees do not depend on word order means the trees are able to handle variation in word order easily, so that they translate well over different languages. In fact, Dependency

Trees have been used with success in Machine Translation (Lavoie et al., 1999). Their independency of language means that a generation system using Dependency Trees can be adjusted to another language quite easily; only the rules that are specific to the generated language will have to be replaced, but the generation algorithm can remain intact.
The independency of word order, and the labels that specify which role a node performs in its parent syntactic category, make Dependency Trees easy to manipulate, especially for the purpose of generating ellipsis. An exception to this is Right Node Raising, as this structure depends on word order instead of not dependency labels. In this project, the Alpino format for Dependency Trees (Bouma et al., 2001) will be used, with some minor changes. A tag for morphology must be added, to enable the Surface Realizer to perform morphology. The tags that indicate the position of a word or syntactic category in the sentence are left out, because these positions are determined at a later stage, within the Surface Realizer.
The discourse structure graphs, of which the Dependency Trees form the nodes, were inspired by Rhetorical Structure Theory (Mann & Thompson, 1987). This theory was originally developed as a descriptive framework for the analysis of text structure, but it is also used in several NLG applications (for an overview, see Hovy, 1993). Among other things, rhetorical relations influence the syntactic structure of a coordinated and elliptic sentence. Shaw (2002) uses discourse structures to determine the syntactic structure that should be used to combine two propositions. Not every structure is suitable to express all relations – some structures imply certain relations, and cannot be used for others. When, in Shaw's algorithm, a syntactic structure is selected to combine two propositions, this structure is determined by the rhetorical relation that connects the propositions. However, the generation of ellipsis takes place at a later stage, and it seems that the rhetorical relation is no longer used at that stage. Hendriks (2004) showed that rhetorical relations do hold as well for certain elliptic structures, such as gapping: a gapped sentence cannot have a causal relation between its clauses, but only an Additive or Contrast relation. This indicates that we will need rhetorical relations at the level where ellipsis is realized, to ensure that a sentence is not elliptic when ellipsis would give it a different reading.
Dependency Trees were constructed to be language independent. It seems that rhetorical relations are language independent as well, as is illustrated by the fact that Rhetorical Structure Theory has been applied successfully to various languages including English, Dutch, German, and Portuguese. This means that, like Dependency Trees, rhetorical structures can be passed on to the Surface Realizer, to be used for the selection of a correct and fitting coordinating syntactic and/or elliptic structure, when combining two Dependency Trees. And so the input to the Surface Realizer consists of Dependency Trees which are linked by rhetorical relations.
The rhetorical relations that we currently distinguish are Cause, Contrast, Purpose, Temporal and Additive. These were judged to be most important to storytelling. However, the number of relations can be easily expanded.
Because the Narrator is closely connected to the rest of the multi-agent system, it can (or should be able to) access the reasoning, plans and emotions of the Actor agents, and use this information to determine which rhetorical relations to include in the Rhetorical Dependency Graph. For instance, an agent makes a plan to reach a certain goal (Purpose), and may adopt a goal because of a certain emotion (Cause). Opposing goals, or actions resulting from these goals, indicate a Contrast relation. Temporal relations should be easily deducible as well, and certainly feature large in stories. Additive relations were necessary to be able to produce all the desired forms of ellipsis.

The decision that two clauses are suitable to be combined, and the decision which rhetorical relation connects them, will be made in the Content Planner. But the actual combining of the

two clauses, and the decision of how, with what linguistic structure, to do it, will take place in the Surface Realizer.
As an extra argument, some aspects of ellipsis may be language-dependent. In particular the directional constraints that Shaw mentions (Shaw, 2002) are based on syntactic information that is language-specific (for instance, word order). The Dutch language has no VP-Ellipsis. In Right Node Raising, the rightmost string is deleted, so this structure depends on word order. As dependency trees are supposed to be language-independent, and as the word order is only known at the Surface Realizer stage, this suggests that the grammar in the surface realizer is the proper place to realize ellipsis.

The construction of a Dutch cue word taxonomy by Knott & Sanders (1998) is an interesting and very promising work, which could be of vast importance to the field of Natural Language Generation. However, their taxonomy is rather complex and will presumably be hard to implement. Moreover, Knott & Sanders admit that their taxonomy was created using only those cue phrases that were easiest to classify; other cue words will be even harder to classify and cause the taxonomy to be even more of a labyrinth. For these reasons it will be better to create a less convoluted taxonomy for our own purposes. This taxonomy should be adapted to the needs of the Virtual Storyteller and to the information the Narrator has access to, to facilitate a clear implementation.

3. Design

In this section I will describe the design of the Narrator. First the input that the Narrator expects is defined. Then the process of transforming the input to the Surface Form is described, and finally the cue word taxonomy that was constructed specifically for the Virtual Storyteller is shown.

3.1 Input to the Narrator

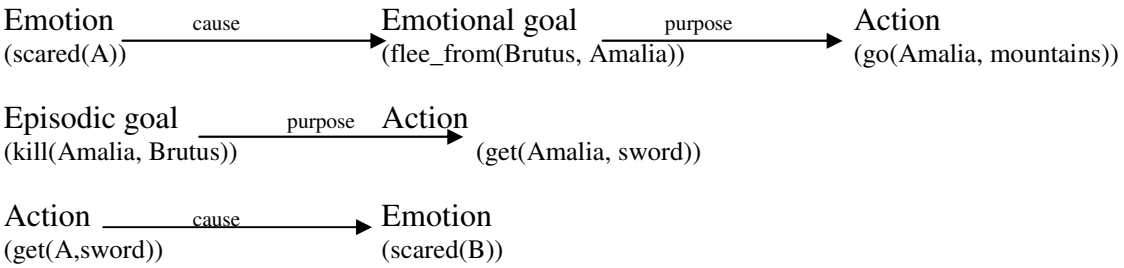
This section specifies the input of the Narrator and its different modules. The precise format of the input is at the moment not quite clear, as the Virtual Storyteller is being revised, but the input will probably be represented in the OWL Web Ontology Language (http://www.w3.org/2001/sw/WebOnt/). Logical propositions, representing actions, goals, emotions and plans, are connected by relations such as motivation. The plot is probably passed to the Narrator as a whole, but because it was not certain throughout the project, it is also possible to pass the input in chunks to the Narrator. All actions are set at a certain time so the story can be told chronologically, but if the plot is given as a whole the story can also be narrated from a certain perspective or in a cyclic manner. The following examples give an idea of expected input propositions:

- Actions, for instance: go(Amalia, mountains), scream(Amalia)
- Emotions (that is, heavily felt emotions), for instance: scared(Amalia)
- Plans, reasoning
  For instance: get(Amalia, sword) $\xrightarrow{\text{Purpose - plan}}$ go(Amalia, mountains)
- Goals
  - Episodic goal (goal to be achieved at the end of the story)
    For instance: kill(Amalia, Brutus), imprison(Brutus, Amalia, castle)
  - Emotional goal (goal caused by a strong emotion)
    For instance: flee_from(Brutus, Amalia)

There is also information that is valid throughout or at the beginning of the story. This is probably passed to the Narrator at the initialisation of the story, unless it is included in the plot, but may be vital to understand certain events. In such cases it should be included in the proper place by the Content Planner (see the section on processing).

- Attitudes (the emotional goals a character develops given certain emotions, e.g. the way he/she probably react)
  For instance: scared(Amalia) → flee_from(Brutus, Amalia)
- Background information
  For instance: isa(Amalia, princess), isin(sword, mountains)

These different types of information are often connected. For example, an emotional goal is caused by a strong emotion, which can lead to a certain action (see figure 3.1).



Figuur 3.1: Connections between propositions

3.2 Processing

This section describes the design of the Narrator, and the functions of its different modules. In the future, the Narrator should consist of three modules (see figure 3.7), the Content Planner, the Clause Planner and the Surface Realizer. At the time of writing only the Surface Realizer is implemented.

3.2.1 Content Planner

The Content Planner is the first module of the Narrator and has three tasks:

- Selecting relevant information from the input and the background- and attitude information
- Setting the paragraph borders
- Adding rhetorical relations, for instance adding a purpose-relation between a goal and an action it has caused

As the Narrator receives the entire plot at once, it is necessary to divide the text into paragraphs. Divided in paragraphs, the generated narrative will seem more coherent, less confusing.

The selection of relevant information is a very important task. Unnecessary information that might be passed in the input must be removed. For example, when the emotion 'fear' is very strong in the Amalia-character, this information is passed to the Narrator. The Content Planner might however discover that Amalia has been very scared for some time, and that this has already been narrated. It would not be necessary, it would even be undesirable, to tell it again, as the listener is already aware of Amalia's fear and the story would become monotone. In this case, the Content Planner should remove this piece of information.

Conversely, there are situations when some background information or character attitudes should be added to the information, because it would explain the motives of the character actions. For instance, the input might state Amalia's plan to get at the sword, and her action of going to the mountains. In this case the information that there is a sword in the mountains, should be added, to make the story more coherent and easier to understand. So we see that the task of selecting relevant information helps to create a coherent and not-repetitive story.

Finally, rhetorical relations must be added between the propositions in the input. For now, I distinguish five broad categories of rhetorical relations: Cause, Purpose, Contrast, Additive and Temporal. These are based on, but not identical to the rhetorical relations defined by Mann & Thompson (1987). In Rhetorical Structure Theory, there are far more, and far more specific, rhetorical relations. Most of these relations are simply too detailed for the simple stories created by the Virtual Storyteller.

Some of the distinctions do not make sense when viewed from the perspective of the dependency tree level. In a text, Cause and Consequence can be two different relations. The nucleus is the important clause, which the satellite supports. The nucleus cannot be removed from the text without making it impossible to understand, while the satellite can be removed with little loss of meaning. This is how the nucleus and satellite are distinguished in classic RST. If the nucleus is the cause and the satellite the consequence, we see a Consequence-relation. Otherwise, if the satellite is the cause, we see a Cause-relation.
The Virtual Storyteller makes no such distinction. The Content Planner first determines that both clauses are important enough to mention, then that there is a cause-consequence relation between the two clauses. Because it has no way to distinguish between a Cause- and a Consequence relation, and because these two relations can be realized at the surface level in the same way, I have only defined a Cause-relation, in which the satellite is always the cause and the nucleus the consequence. This is convenient when, in the Surface Realizer, a selected cue word must be added to either the nucleus or satellite. Take for instance the cue word 'omdat' (because). This cue word must always be attached to the Cause. If the satellite is always the cause, then we know that 'omdat' must always be added to the satellite. If we had both a Cause and a Consequence relation, the category of the relation would have to determine which Dependency Tree the cue word would be added to, thus complicating matters needlessly.
There is one drawback, however. In RST, the nucleus is the text-part on which lies the emphasis, the most important bit. In the Narrator this distinction is not present, but it could be useful in the future. The problem could however be solved by adding a primitive (see section 4.3, on the cue word taxonomy) to the category of the relation, that stated if the nucleus or the satellite should be emphasized, or if they are equally important. This primitive would restrain the number of cue words that are appropriate to the relation. This makes sense because the category of the relation, and the primitives it contains, is the only variable that restricts the number of appropriate cue words.
Another drawback is that there can be no multi-nuclear relations, such as Additive and Contrast in classic RST. However, arbitrarily assigning one the status of 'nucleus' and the other 'satellite', will do no great harm. It will slightly reduce the possible Surface Forms, because the clauses can be ordered only one way instead of two, but that is all.

In the same way, I have a Purpose-relation which has a purpose or goal as satellite, and a consequence (maybe an action, maybe another goal) as nucleus. In a Contrast-relation, two clauses contradict each other, for example an emotional and an episodic goal. In a Temporal relation, two clauses are in some specified way connected in time. They may be right before or after each other, or during each other. Some kinds of Temporal relations are special in that they have no satellite, only a nucleus. For instance, only one clause is needed for a 'finally' or 'suddenly' relation.
These four relations are important for storytelling, and should be easy to use for the Virtual Storyteller. Temporal relations will help especially when the actions of two different characters are narrated (for instance, "while Amalia was searching the sword, Brutus was preparing his dungeons"), or to narrate the story chronologically. Of course one has to be careful, otherwise a childlike, monotonous narrative (and then, …, and then, …) is generated.
Cause and Purpose relations make causes and motives explicit, and thus help the listener to understand why the characters perform the actions they do, and why certain things happen. They make the story far more coherent.
A Contrast-relation will bring some nuance in the story, and help to avoid the impression that important motives are ignored. For instance, when Amalia has an emotional goal to flee from

Brutus, and an episodic goal to kill him, and is able to kill him because she has a sword, it helps to state the internal conflict in Amalia. Otherwise a listener might ask afterwards: but what about her fear? The listener would feel the story is incoherent, because it seemed the fear was ignored. Narrating that the fear is still felt, but that killing Brutus is more important, makes the story seem far more balanced.
The Additive relation was added mainly because, apart perhaps from Contrast, it is the only relation for which all forms of ellipsis are possible. As the major aim of the project was to generate ellipsis, it was necessary to implement an Additive relation.

The output of the Content Planner is a Text Plan, a graph with propositions as its units, which are connected by rhetorical relations (see figure 3.2). Only information that must be communicated is included. Every rhetorical relation that connects two units is specified, but these do not all need to be explicitly realized. That is left to the discretion of the Surface Realizer module.

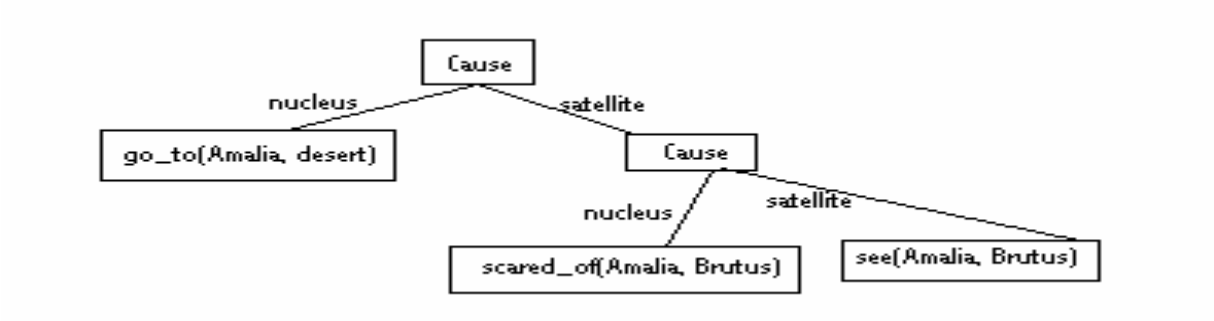


Figure 3.2: Text Plan

3.2.2 Clause Planner
The Clause Planner has two tasks: Lexicalisation and Lexical Aggregation. During lexicalisation the propositions from the Text Plan are replaced by Dependency Trees. Lexicalisation is the process of selecting words suitable to express the meaning of the proposition. These words have a root and a Part-of-Speech (pos) tag, and are given adequate morphological information and their role in the grammatical category that is their direct parent. This parent has a syntactic category and its role in its own parent category. If a node is the root of a tree, its relation is 'top'.
The necessary information is obtained from the Lexicon-module. This Lexicon has entries on all words that can be used by the system (see figure 3.3).

λx.lopen(x):
- HD=rennen, SU=x Morph=Morph(x) + tense, pos(rennen)=V, cat(rennen(x))=SMAIN
- HD=lopen, SU=x Morph=Morph(x) + tense, pos(lopen)=V, cat(lopen(x))=SMAIN
….

Figure 3.3

If, for instance, the proposition 'lopen(Amalia)' has to be lexicalised, one of the entries in figure 3.3 is selected and a node is created with pos-tag 'verb', rel 'HD'. A subject is selected from the entries concerning 'Amalia', and the Morph-tag of the resulting node is copied to the Morph-tag of the HD-node, with 'past-tense' added to it. According to the entries, these two nodes combined constitute the syntactic category 'SMAIN', so a parent node is added with cat-tag 'SMAIN', and rel-tag 'top' because it is the top-most node. The resulting dependency tree is given in figure 3.4.

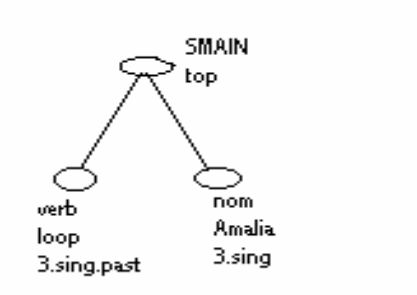


Figure 3.4

```
<?xml version="1.0"?>
<!DOCTYPE rhetdepgraph>
<rhetdepgraph>
    <rhetrelation cat="cause">
        <nucleus>
            <dt>
                <node rel="top" cat="smain">
                    <node rel="hd" pos="verb" root="vlucht" morph="3.sing.present" />
                    <node rel="su" pos="nom" root="Amalia" />
                    <node rel="mod" cat="pp">
                        <node rel="hd" pos="prep" root="naar" />
                        <node rel="obj1" cat="np">
                            <node rel="hd" pos="noun" root="woestijn" morph="sing" />
                            <node rel="det" pos="det" root="de" morph="sing" />
                        </node>
                    </node>
                </node>
            </dt>
        </nucleus>
        <satellite>
            <rhetrelation cat="cause">
                <nucleus>
                    <dt>
                        <node rel="top" cat="smain">
                            <node rel="hd" pos="verb" root="zijn" morph="3.sing.pr" />
                            <node rel="su" pos="nom" root="Amalia" />
                            <node rel="predc" pos="adv" root="bang" />
                            <node rel="pc" cat="pp">
                                <node rel="hd" pos="prep" root="voor" />
                                <node rel="obj1" pos="nom" root="Brutus" />
                            </node>
                        </node>
                    </dt>
                </nucleus>
                <satellite>
                    <dt>
                        <node rel="top" cat="smain">
                            <node rel="hd" pos="verb" root="zien" morph="3.sing.pr" />
                            <node rel="su" pos="nom" root="Amalia" />
                            <node rel="obj1" pos="nom" root="Brutus" />
                        </node>
                    </dt>
                </satellite>
            </rhetrelation>
        </satellite>
    </rhetrelation>
</rhetdepgraph>
```

Figuur 3.5: xml-output of the Clause Planner

When all propositions have been lexicalised, some lexical aggregation could be performed as well. For example, summing ups such as 'Monday, Tuesday, Wednesday, Thursday and Friday' could be reduced to 'working days'. As said above, if the same referent is present in the text plan more than once, it is marked with an index.

The output of the Clause Planner is a Rhetorical Dependency Graph. This is the same graph as the graph represented in the Text Plan, but the units, which were propositions, have been replaced by Dependency Trees, which have syntactic, lexical and morphological information. The nodes are unordered and the word roots are uninflected. Figure 3.5 gives the xml-representation of a Rhetorical Dependency Graph that could be generated on the basis of the Text Plan in figure 3.2.

3.2.3 Surface Realizer

The Surface Realizer is the last module in the Narrator. It handles four tasks: Syntactic Aggregation, Referential Expression Generation, Generating Surface Form and Orthography. Syntactic Aggregation is handled in the first phase, when the Rhetorical Dependency Graph is transformed to one or more Dependency Trees. For every rhetorical relation in the graph, a way is sought to process it. It can be processed explicitly, by combining two Dependency Trees using a cue word or a conjunctor, or implicitly by doing nothing at all. Doing nothing about some relations will be a good way to avoid too convoluted sentences and repetitive stories (and then…, and then …). When the same node is present more than once in an (aggregated) dependency tree, ellipsis can take place. A Dependency Tree is always ellipted as much as possible. The node can be removed from the graph and the dependency tree if necessary adjusted to this removal. For instance, it may be that the morphological tag needs to be changed. Objects and characters can be referred to in different ways. Referential Expression Generations selects referents that clearly refer to a specific object or character without creating confusion, and yet avoid monotonicity. In some cases, 'he' or 'she' are adequate, at other times (perhaps when there are a prince and a villain, instead of a princess and a villain), more specific referents such as 'the villain' may be needed.
The result of Syntactic Aggregation is one or more Dependency Trees. Their structure and morphological information may differ from the original Dependency Trees that were the units of the Rhetorical Dependency Graph, but the lexical information passed by the Clause Planner should remain constant.

The next step is the transformation of these Dependency Trees to Surface Forms. A Surface Form is simply a sentence, a piece of text. To achieve this sentence, the nodes in the dependency tree have to be ordered using their syntactic category and rel-tags. The Grammar-module contains the necessary rules for this ordering (see figure 3.6). Every grammatical category has its own rules that indicate how its children can be ordered, according to their rel-tags.

```
SMAIN:  SU HD, SU HD OBJ1, SU HD VC, SU HD PP, …
SQUEST: HD SU, HD SU OBJ1, HD SU VC, HD SU PP, …
….
Gaan: ga, gaat, gaan; ging, gingen; gegaan;
Lopen: loop, loopt, loopt; liep, liepen; gelopen;
```

Figure 3.6: Grammar rules and strong verb inflections

If a word has a morph-tag, its root is inflected using, again, rules about inflection from the Grammar. If the root is a strong verb or another irregular wordform, the Grammar should contain a listing of its inflections.

Finally orthography is added, using comma's, full stops, capitals, etc. Then the sentence gets exported to the Speech Generator, in a Java String format. In the future, prosodic information could be added, to help and improve the Speech Generator module.

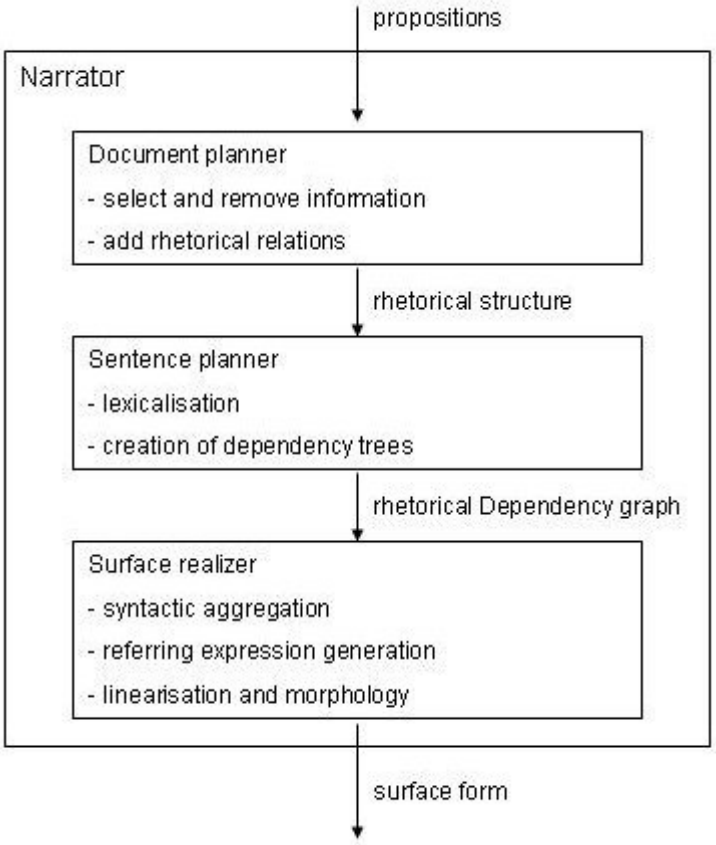


Figure 3.7: Narrator architecture

3.3 Cue word taxonomy

In section 2.6 we discussed the necessity of creating a cue word taxonomy for the Virtual Storyteller, to improve syntactic aggregation. This section describes the cue word taxonomy that was created for the Virtual Storyteller, and how it was created.

3.3.1 The sentences

In the section on the cue phrase taxonomy, I have discussed the taxonomy for Dutch cue words, created by Knott & Sanders (1997). In the construction of this taxonomy, they used a so-called 'substitution test'. To apply this test to a (set of) cuewords, you take a sentence which contains one of those cue phrases, and try to replace it by other cue phrases. If a cue phrase can be substituted, the original must be more specific or just as specific in the features

it signals, as its substitute. If a certain cue word cannot be a substitute, it must signal some features that the original cue phrase does not.
In this way, a hiërarchic classification of cue words can be created. Knott & Sanders have done so for a subset of the Dutch cue phrases. Additionally, they have given criteria to separate the classes in the classification, based on the cognitive primitives found by Sanders et al. (1992). These primitives, however, do not seem useful for the Virtual Storyteller, for several reasons.
In the first place it is hard for the Virtual Storyteller to discover which primitives are valid for certain cases. Especially the distinction between semantic and pragmatic is impossible to make for the Virtual Storyteller, because it does not distinguish between speech acts and other propositions. Besides, in (Sanders et al., 1992) this primitive is already found to be vague and to create confusion.
Second, at the stage that the taxonomy will be used, the Virtual Storyteller does not work with text and sentences, but with Dependency Trees. This means that a primitive such as Basic, which distinguishes between cue phrases that put the antecedent first and phrases that put it last, becomes far less relevant. As long as the order of antecedent and nucleus does not influence the meaning that is expressed, both options are equally valid, and therefore both sets of cue phrases can be used. However, as mentioned in section 4.2.1, it might be convenient to have a primitive that denotes which clause should have emphasis.
Finally, I remark that the taxonomy for Dutch cue phrases, as defined by Knott & Sanders, is very convoluted, so that it seems very hard to implement.

For these reasons I have tried to create a small classification of Dutch cue words myself. I have used words that seemed useful to me for the Virtual Storyteller, and that appear over a hundred times in the Dutch part of the CGN-corpus (Corpus Gesproken Nederlands, release four, 2001, first CD). I assume that words that have a frequency of more than a hundred are fairly common in Dutch texts. I have chosen the figure hundred because it is a nicely rounded number, and because, when I looked at the words that appeared less than a hundred times in the corpus, words that seemed rather less common started to appear, such as 'politiek' (politics, 70 times), 'akkoord' (agreement, 81 times) and 'Europa' (Europe, 84 times). These words probably scored well because of the subject material. If a corpus is based on newspaper articles, the word 'journalism' is likely to appear rather more often than when it is based on fairy tales.
I have adapted the substitution test to my uses, because of the difference between Surface Forms and Dependency Trees. I have been less strict in approving substitutes. When it made no significant difference in the meaning of the sentence, I have allowed both cue words that put the antecedent first, and that put it last, to mark the same Dependency Trees. In addition I have even allowed the structure of the clauses to be adapted if a cue word needed it and if it did not affect the meaning. The Conjunctor Transformator will make sure the resulting aggregate Dependency Tree is grammatical again. Finally, I have also approved cue words which were more specific than the original cue word (if there was one), if the new features that the substitute signals are in accordance with the story meaning.
In this way I expect to get a taxonomy that specifies which cue words can be used to connect two clauses and to get a specific meaning across.

I have used two kinds of test sentences. The first type were taken from a 'Sprookjes en Vertellingen' by Hans Christian Andersen, a Dutch translation dating from 1975. Here the underlined phrase is the original cue phrase, the cursive words are the possible substitutes. The words marked with a star seemed to me unsuitable for the substitution, the question mark means I was not sure.

The second type are based on actual output by theVirtual Storyteller. When two sentences are combined, what cue words are suitable? These examples do not have an original cue phrase, but otherwise they have the same notation. Examples of both types of sentences are given in figure 3.8.
When I look at the sentences taken from Andersen, it appears that the cue words 'maar' (but), 'en' (and), 'toen' (then), 'want' (because) and zodat ('so that?') were sufficient for the purposes. Nearly all sentences used one (or more) of these cue words, and no others. Especially when a contrast was expressed, only 'maar' was used, never 'hoewel' (although) or 'ondanks' (despite). Even so, the story was not monotonous or boring. It may be that, for the purpose of telling a fairytale, only a few cue words are sufficient, or that these are the first cue words children learn.
The complete set of sentences that I used are given in appendix A.

Er was eens een meisje, maar 's zomers liep zij altijd op blote voeten want ze was arm,
ze was zo fijn en zo lief, *echter* *omdat*
**toch* *dus*
**hoewel* *zodat*
**ondanks* *daardoor*
doordat
daarom

en 's winters op grote klompen, zodat haar wreef helemaal rood werd.
**ook* *waardoor*
bovendien *doordat*
omdat

Amalia is bang. Ze wil vluchten. Ze wil Brutus doden. Ze steekt hem neer.

maar (ook)	*maar ook*	*dus*
toch	**toch*	*om*
**terwijl*	*?terwijl ook*	*want*
hoewel	*hoewel ook*	*omdat*
echter	*echter ook*	*daarom*
ondanks	**ondanks*	**zodat*
		**daardoor*

Figure 3.8: test sentences

3.3.2 The taxonomy
Using these sentences I have grouped the cue words and tried to find criteria to distinguish these groups. The classification is shown in figure 3.10. The most important criteria are names of rhetorical relations: Cause, Temporal, Contrast and Additive. Purpose turned out to be greatly similar to a part of the Cause-set of cue-words. Each relation or primitive divides the set of cue words in smaller subsets. A cue word in a small subset can always be replaced by a more general cue word, from a superset.
In the Cause-relations, one primitive denotes whether the cause is put first or last, and whether the consequence was voluntary (consciously willed by an actor) or not. In the context of the Virtual Storyteller, this is the same distinction as between Attitude (the way a character reacts to a certain emotion) and Plan (the first step of a plan towards some goal). The cue-words of the Purpose relation are the same as the cue words of the voluntary Cause relation, with one addition.
In the Temporal relation, I have distinguished several subsets dealing with the order of events: before, after, sequence, finally, during, suddenly and once (for 'Er was eens…' or 'Once upon a time… '). In the Contrast relation, there is one subset named 'unrealized cause'. This subset

encompasses the cases when one clause is the direct opposite of what you would expect it to be, based on the other clause, for instance in 'Although Amalia was scared, she did not flee'.

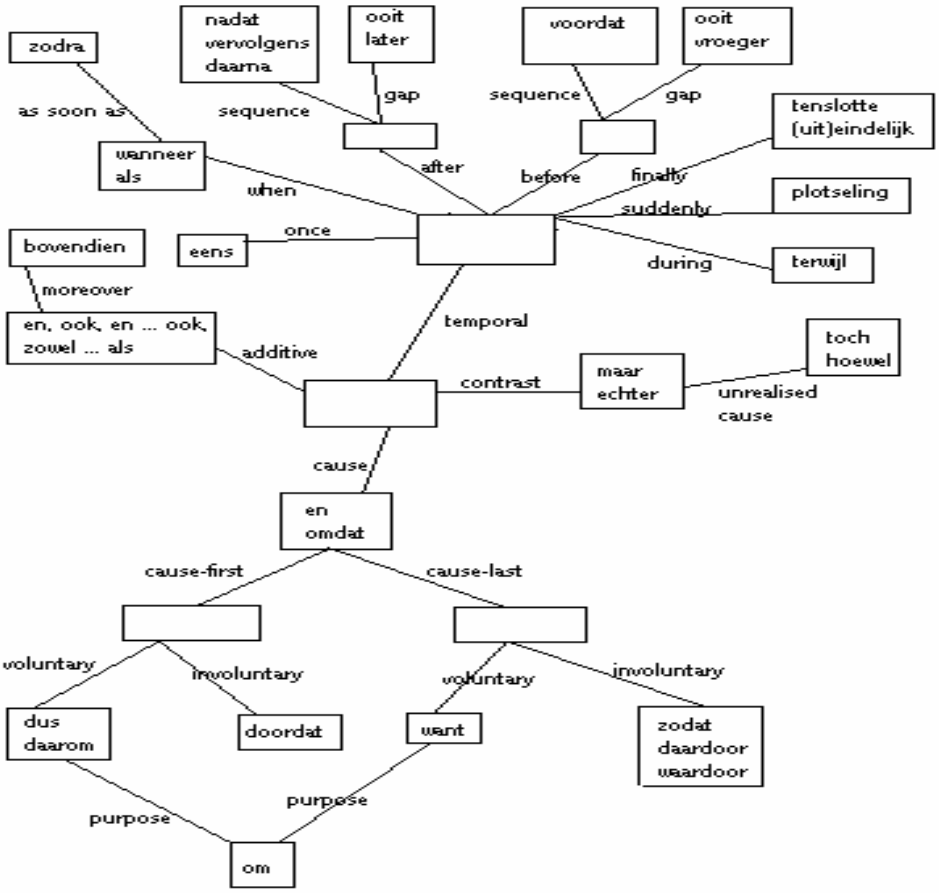


Figure 3.9: Taxonomy for the Virtual Storyteller

RELATIONS	PRIMITIVES		CUE WORDS
Cause	Voluntary	Cause-First	dus, daarom
		Cause-Last	want
		Purpose	om
	Involuntary	Cause-First	doordat
		Cause-Last	zodat, daardoor, waardoor
			en, omdat
Additive	Moreover		bovendien
			en, ook, en … ook, zowel … als
Contrast	Unrealized Cause		toch, hoewel
			maar, echter
Temporal	After	Gap	ooit, later
		Sequence	nadat, vervolgens, daarna
	Before	Gap	ooit, vroeger
		Sequence	voordat
	Finally		tenslotte, (uit)eindelijk
	Suddenly		plotseling
	During		terwijl
	Once		eens
	when	As soon as	zodra
			wanneer, als

Figure 3.10: cue word classification

3.4 Syntactic Aggregation

This section describes the process of syntactic aggregation, as it is performed by the Surface Realizer of the Virtual Storyteller. The goal was to be able to produce several different sentences for one piece of input. There must be different appropriate cue words for each rhetorical relation. These cue words determine the structure of the Surface Form: for instance, a paratactic cue word will produce a conjunction, while a modifier will result in two separate trees.
After the cue word has been selected and been added to a tree, the resulting tree should be ellipted as much as possible. A tree can only be elliptic if it has a paratactic structure. Additionally, some forms of ellipsis are only applicable for additive relations (see section 2.6.1). There are several constructions we wanted to be able to produce:

- Paratactic constructions: *Amalia left the desert and Brutus entered the forest*
- Hypotactic constructions*: Amalia left the desert, because she saw Brutus*
- Conjunction Reduction: *Amalia entered the desert and saw Brutus*
- Right Node Raising: *Amalia entered and Brutus left the desert*
- Gapping: *Amalia entered the desert and Brutus the forest*
- Stripping: *Amalia entered the desert and Brutus too*
- Coördinating one constituent: *Amalia and Brutus entered the desert*

3.4.1 Rules and Constraints

The Rhetorical Dependency Graph is transformed by RhetDepGraphTransformers. There are three kinds of transformers:

- **RelationTransformer**: Transforms the Cause, Purpose, Additive, Contrast and Temporal rhetorical relations, using likewise named libraries holding information on appropriate cue words. The Dependency Trees that figure as nucleus and satellite may be combined into one, or not. A cue word will be added to signal some lexical features of the relation. A relation can be passed with extra variables, which indicate which primitives should be signalled (such as 'plan' or 'attitude' in the Cause-relation – see section 3.3). If these variables are known, a highly specific cue word can be selected. The most specific cue word is not necessarily selected; discourse history plays a part as well. If a cue word has been recently used, it is less likely to get chosen again. If no extra variables are passed along, a more general cue word is chosen.
  Also, the size of the resulting Dependency Tree is constrained. A conjunction can only be nested once (so at most have three conjuncts), to prevent the Surface Form getting too complicated. So if one of the clauses in a relation is already a conjunction, no paratactic or hypotactic cue words is allowed.
  When a cue phrase is selected, a suitable node is created and it is passed, along with the relation, to Conjunctor, where either the dependency trees are combined, or the cue word is added to one of them.
  - Constraint: There is only one rigid constraint that works on the RelationTransformers: the concerning rhetorical relation must be present in the Rhetorical Dependency Graph. The Discourse History influences the selected cue word, to prevent monotony.
  - Example: Assume the RelationTransformer receives a Rhetorical Dependency Graph (shown in figure 3.11) consisting of one Causal-Voluntary relation, which has as its

nucleus 'Amalia vlucht' (Amalia flees) and as its satellite 'Amalia is bang' (Amalia is afraid). To process this relation, a cue word is selected by the Cause-library. The primitive 'Voluntary' is specified, so if 'want' has not recently been used this will be selected, because it is the most specific (see the cue word taxonomy in section 3.3). If either nucleus or satellite of the Causal relation had been a nested relation itself, 'want' would not have been chosen because it is a co-ordinator, and the resulting aggregate tree would be too complicated.
Now that the cue word has been selected, the tree node representing the co-ordinator 'want' and the Causal relation are passed to the Conjunctor.

- **Conjunctor**: This Transformator is called by a RelationTransformer, to create a grammatical Dependency Tree out of the relation and the cue word passed by the RelationTransformer. Only entire trees are conjoined; the conjunction of single constituents (nodes) is handled by the Elliptor. The structure of the resulting Dependency Tree depends on the relation-tag of the cue word. If the cue word is a co-ordinator, a paratactic structure is chosen. If it is a complementer, a hypotactic structure is the result. If the cue word is a modifier, the cue word is added to either the nucleus or the satellite (depending on the cue word), and both separate trees are added to result, the unmodified tree first.
  - Constraint: The relation-tag of the cue word must be either crd, cmp or mod. The relation must have a nucleus, and if the cue word is a co-ordinator or a complementer, a satellite as well. The categories of the root nodes of the nucleus and satellite must be either 'SMAIN' or 'CONJ'.
  - Example: Take the example we discussed when treating the RelationTransformer. The cue word 'want' is a co-ordinator, so it needs a paratactic tree. A new Dependency Tree is created, with as its root node a Conjunction. The nucleus and satellite of the tree become its conjuncts (with the former nucleus labeled 'CNJ-NUCLEUS', and the satellite 'CNJ-SATELLITE', because the distinction will be important when performing linearization), and its co-ordinator is 'want'. This new tree is then passed to the Elliptor.

- **Elliptor**: The Elliptor Transformator is called by Conjuction, and removes superfluous nodes or branches from a dependency tree. First both conjuncts of the tree are checked for identical nodes or branches (the identical nodes have to be direct children of the root). If the conjunction is nested, the nodes that the nested conjuncts share are compared to the nodes in the other conjunct. Any identical nodes are marked.
With the marked nodes of one conjunct, an appropriate construction is determined. If no nodes are marked at all, the Elliptor can do nothing. If all but one node is marked, two constructions are equally eligible: Stripping and Coordinating a constituent.
Otherwise, the other constructions, Gapping, Conjunction Reduction and Right Node Raising, are performed if possible. For example, if the subjects are marked, the 'Conjunction Reduction' is selected. These three constructions all remove one node, and can be performed simultanuously. If both the subjects and the main verbs are marked, then both Conjunction Reduction and Gapping are performed.
Superfluous nodes are deleted, but their parent node receives a connection to their remaining twin, marked 'borrowed', so that the component that produces the Surface Form by ordering the nodes knows that the node should not appear in the Surface Form at this position. If we did not add this edge, the tree could not be ordered because it would not conform to the grammatical rules. Take, for instance, the tree is figure 3.11. There is no rule that says an SMAIN (sentence) has only a head; it must have a subject as well. This

way, the tree still fits the grammatical rules as it has the correct number and kind of children, but the generated Surface Form is an elliptic sentence.

- Constraint: The Dependency Tree must have a paratactic structure. At least one node or branch must be identical over both conjuncts. If the conjunction is nested, there must be an identical node in the topmost conjunct to the node that the nested conjuncts share.
  Some constructions have their own restrictions, such as Gapping, which demands an additive relation.
- Example: Proceeding with the earlier example, the Dependency Tree created by the Surface Realizer was paratactic. The conjuncts also have one node in common, i.e. their subject, 'Amalia'. When only the subject is identical, Conjunction-Reduction is the suitable elliptic structure. The subject node of the second conjunct (the nucleus) is removed, and replaced by a connection with the subject of the first conjunct (the satellite). The result is shown in figure 3.11.

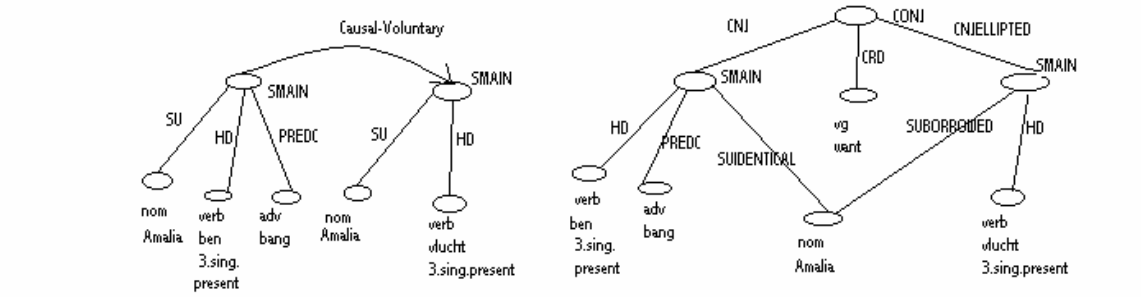


Figure 3.11: input SR, output Elliptor

- **ReferentialExpressionGenerator:** When a standard object in the story world, such as a character or a certain place, is present in the Rhetorical Dependency Graph, a referential expression must be chosen for it. This could be a name or a description (such as 'princess'). When the object is present more than once, or when it is recent in the Discourse History, a pronoun such as 'he', 'she' or 'it' can be selected.
  - Constraint: An object or character has to feature in the story world. To receive a pronoun, it has to be the last object of that gender that was named, and have the same dependency relation as when it was mentioned before as well. This last rule was added to avoid ambiguity in sentences such as 'Brutus hit the Prince and he screamed'. This sentence suggests that Brutus screamed, while it was actually the Prince. Demanding that the dependency relation must be identical prevents such ambiguities.
  - Example: In the earlier example, 'Amalia' is a character, and so would be checked. If 'Amalia' is marked as recent, which would mean that she was the last female to be mentioned in the last three sentences, and had the dependency label 'subject' at that time, she receives a pronoun. As she is the subject, the pronoun would be 'zij' (she). If she is not recent, 'Amalia' is set to recent in the Discourse History, but no pronoun is selected.

3.4.2 Operations on the Rhetorical Dependency Graph

The Transformators must be able to operate on the Rhetorical Dependency Graph. The necessary operations are:

- Adding a node to a Dependency Tree. The Conjunctor has to add one or more nodes for the cue word, and to combine two Dependency Trees. The Elliptor sometimes adds a node as well, when Stripping or coordinating a constituent.
- Removing a node or a branch from a Dependency Tree, when it is superfluous. This is a task of the Elliptor.
- Adding an edge between two nodes. This is also a task of the Elliptor, to replace the node that was deleted.
- Changing the attributes of a node. The ReferentSelector needs to change the root-tag of a node when it has chosen a suitable referent. The Elliptor may sometimes need to change the morph-tag of a node, for instance the verb when coordinating two subjects.

3.4.3 Results

The result of syntactic aggregation is one or more Dependency Trees. In most cases, there will be fewer left Dependency Trees than the Rhetorical Dependency Graph contained, but the new Dependency Trees will be more complicated and more diverse. They will contain conjunctions, different referential expressions. They will not unnecessarily repeat words, but they will not be too long and complicated either.

The Dependency Trees are then returned to the Surface Realizer, for further processing.

4. Implementation

This document describes the classes that form the Surface Realizer, the natlang.rdg package, and their functionality. It consists of five packages: model, transformer, discourse, libraries and view.

4.1 Modelling

This section describes the natlang.rdg.model package, which reads a RSGraph (a Rhetorical Dependency Structure) from an xml-file and represents it as a tree structure.

4.1.1 RSGraph & RhetRelation
The Rhetorical Dependency Structure is represented by RSGraph, a graph containing RSVertices. These RSVertices can be RhetRelations and RSDepTreeModels. A RhetRelation is a vertex with a relation category, which is connected by RSEdges to a nucleus and a satellite, both RSVertices. A RhetRelation represents a Discourse Relation.

4.1.2 RSDepTreeModel
A RSDepTreeModel is a graph that represents a Dependency Tree. It contains RSDepTreeNodes and MLabeledEdges, which connect the nodes and label the syntactic relations between them. The topmost node is the root of the Tree.

4.1.3 RSDepTreeNode
A RSDepTreeNode represents a vertex in the graph. It contains a RSTreeNodeData, a data-structure holding information. Each node has a relation to its parent node. If the node is not a leaf, it has a syntactic category, else it contains a pos-tag, a root form, a morph-tag and, after inflection, a word. Each node can further contain an index, which signals that this node features elsewhere in the RSDepTreeModel.

4.1.4 OrderedDepTreeModel
Out of a RSDepTreeModel, an OrderedDepTreeModel can be constructed. This contains the same RSDepTreeNodes, but the MLabeledEdges are exchanged for MOrderedEdges, which indicate the position of a target node in the surface form. The edges are numbered according to a rule selected from a library (see section 4.3.2). The OrderedDepTree then produces the Surface Form by putting the nodes in a String, following the order of their edges. If necessary, the root forms of the leaves are inflected (see figure 4.1 and section 4.2.1) and punctuation is added as well. If the label of an edge contains the phrase 'borrowed', the target node is left out of the surface form at that position, which may result in an elliptic sentence (see section 4.2.3).

7

```
FUNCTION Order(RSDepTreeModel) RETURNS surfaceForm
BEGIN           //order tree recursively, starting with the root
        OrderedDepTreeModel = OrderTree(RSDepTreeModel.root);
        SurfaceForm = getSurfaceForm(OrderedDepTreeModel.root);
        doOrthography(SurfaceForm);     //orthography – see section 2
END;

FUNCTION OrderTree(currentNode)
BEGIN
        Rule = selectRule(currentNode);   //select a rule to order the children of this node
                                          //see section 3.2
        FOR (i = 0 TO numberChildNodes) DO BEGIN
                MOrderedEdge(i) = MLabeledEdge(i);   //MLabeledEdges changed for ordered
                number = Rule.getPositionOfChild(MOrderedEdge.Label); //get position
                MOrderedEdge.setPositionOfEdge(number);  //tell the edge
                MOrderedEdge.getTarget().OrderTree();     //recurse
        END;
END;

FUNCTION getSurfaceForm(currentNode) RETURNS result
BEGIN
        IF (isLeaf(currentNode)) DO BEGIN                   //if currentNode is a leaf
                doMorphology(currentNode.root);             //inflect with Morphology – see section 2
                result = currentNode.inflectedWord;         //and return the resulting word
                return result;
        END ELSE BEGIN                  //else, get the surface form of its children, in the right order
                WHILE (isNextChild) DO BEGIN
                        getNextChild(currentNode);
                        result.append(getSurfaceForm(getNextChild));   //append child surface form
                END;
        END;
END;
```

Figure 4.1: ordering algorithm and production of Surface Form

4.2 Transformers

This section describes the natlang.rdg.transformer package, which is responsible for all transformations worked on RSGraphs and Dependency Trees. It comprises Morphology and Orthography classes, which play a part in the production of the Surface Form; a Conjunctor and an Elliptor, which perform syntactic aggregation and a RelationTransformer and DepTreeTransformer, which perform transformations on the RSGraph or Dependency Tree they wrap.

4.2.1 Morphology and Orthography

As stated in section one, the Surface Form is produced with the help of Morphology and Orthography classes. The Orthography class adds a comma every time a complementing or coordinating word is processed (except when this cue word is 'en'), and at the end of a CP-node (subordinate clause). When the whole Surface Form has been produced, it adds a full stop at the end of the sentence and capitalizes the first letter. The Morphology class uses language-specific library classes (see section 4.3) to inflect nouns and verbs. Later, there should be functionality to inflect adjectives as well.

4.2.2 DepTreeTransformer

To make all the necessary transformations possible, there are certain basic operations which must be performed on RSDepTreeModels. Such operations are the adding and removing of RSDepTreeNodes and MlabeledEdges. The information that a RSDepTreeNode contains can be adapted as well. For instance, when two subjects have been coördinated, the morphological tag of the main verb must be set to 'plural'. The DepTreeTransformer holds a Dependency Tree on which it performs operations. The operations either take place at the root node or the 'current' node, which can be set to a particular RSDepTreeNode by the user.

4.2.3 RelationTransformer

The main functionality of the transformer package is to process the rhetorical relations in the RSGraph and to combine and aggregate their RSDepTreeModels. The RelationTransformer transforms all relations, starting with the relations that are part of another relation, and finishing with the topmost relation.

The relations are transformed by selecting an appropriate cue word for each relation. This cue word is given in the form of a RSDepTreeNode, with its pos-tag and its root specified. A DiscourseHistory (see section 4.4) ensures that different cue words get chosen as often as possible. If both nucleus and satellite of a relation are conjunctions, or either of them already is a nested conjunction, any cue words that combine the two trees are disapproved. This puts a check on the length of the generated sentences, so that the sentences will remain easy to understand, in spoken form as well.

The relation and the cue word are passed to the Conjunctor class (see section 4.2.3), which produces one or two correct and grammatical RSDepTreeModels out of the nucleus and satellite of the relation and the selected cue word. See figure 4.2 for the pseudo-algorithm.

```
FUNCTION transformGraph RETURNS List
BEGIN
        List resultingDepTrees;     //only transform the topmost relation(s), because the others will be
        FOR ALL RhetRelations rel IN RSGraph DO          //transformed recursively
                IF NOT (RhetRelation.isPartOfOtherRelation) DO
                        resultingDepTrees.add(transformRelation(RhetRelation));
        return resultingDepTrees;
END;

FUNCTION transformRelation(RhetRelation) RETURNS RSDepTreeModel
BEGIN
        IF (RhetRelation.nucleus isa RhetRelation)  DO      //if nucleus or satellite is a relation,
                nucleus = transformRelation(nucleus);       //that must be transformed first
        IF (RhetRelation.satellite isa RhetRelation)        //and replaced by the resulting tree
                satellite = transformRelation(satellite);

        List = getOptionalCueWords(RhetRelation.category);    //get appropriate cue words for this relation
        cueWord = selectCueWord(List, DiscourseHistory)  //select non-recent cue word (see section ??)

        Conjunctor.transform(cueWord, RhetRelation);
END;
```

Figure 4.2: transforming the RSGraph

4.2.4 Conjunctor

The Conjunctor receives a RhetRelation and a RSDepTreeNode representing a cue word, and has to transform these into one or two correct RSDepTreeModels. The method by which the nucleus and satellite of the relation are transformed is determined by the relation-tag of the cue word, which denotes its relation to its parent node (figure 4.3). If the cue word is a coordinator, the two trees are co-ordinated in a paratactic structure (figure 4.4); if the cue word is a complementizer, they are co-ordinated in a hypotactic structure (figure 4.5); and if it is a modifier, it is added to one of the two trees, while the other is left unchanged (see figure 4.6). After a transformation is performed, the resulting Dependency Tree(s) are added to a result list. See figure 4.7 for a examples of the three structures.

```
FUNCTION transform(cueword, relation)
BEGIN
        if (cueword isa coordinator)
                coordinate(cueword, relation);
        else if (cueword isa complementer)
                complement(cueword, relation);
        else if (cueword isa modifier)
                modify(cueword, relation);
END;
```

Figure 4.3: Conjunctor

```
FUNCTION coordinate(cueword, relation)
BEGIN
        new RSDepTreeNode root(CONJ);           //create a new root for a new tree with syntactic
        new RSDepTreeModel tree(root);          //category Conjunction
        tree.addAtRoot(cueword);                //add the cue word below the Conjuntion
        tree.addAtRoot(nucleus.root);   //add the roots of nucleus and satellite to the new tree; all other
        tree.addAtRoot(satellite.root); //nodes automatically with it
        tree.addEdge(CNJ-nucleus);      //add edges labeled 'conjunct' root and old roots of nucleus and
        tree.addEdge(CNJ-satellite);    //satellite
        result.add(tree);
END;
```

Figure 4.4: Co-ordination algorithm

When the cue word is a coordinator, a new RSDepTreeModel is constructed with a root labeled 'CONJ' (conjunction). Its child nodes are a coordinator (the cue word) and two conjuncts (the nucleus and satellite of the old relation). Other than in Alpino-trees, the conjuncts do not have the same relation. The distinction between nucleus and satellite is kept, because in a causal relation and a paratactic structure, we cannot put the consequence before the cause; it would create confusion.

When the cue word is a complementer, a hypotactic structure is created. First it is checked to which tree the cue word should be added, the nucleus or the satellite ('want' has to be added to the cause, i.e. the satellite, but 'daarom' to the consequence, i.e. the nucleus). This tree is the antecedent, the other the head. A new RSDepTreeModel is constructed with root SMAIN, and all nodes and edges in head (except the root) are added to it. The nodes that were connected to the old root are connected to the new root, otherwise the structure remains the same.

Then a modifier is added with category CP, under which are placed the cue word and the subordinate clause. All nodes in antecedent are copied below CP; the category of its root node is changed to SSUB, its relation to BODY.

```
FUNCTION complement(cueword, relation)
BEGIN
        new RSDepTreeNode root(SMAIN);          //create new tree with root SMAIN
        new RSDepTreeModel tree(root);
        IF (cueword.addToNucleus) DO BEGIN      //check which tree is subordinate
                head = satellite;
                antecedent = nucleus;
        END ELSE DO BEGIN
                head = nucleus;
                antecedent = satellite;
        END;

        FOR ALL nodes IN head.root.getChildren DO
                tree.addToRoot (node);     //add all nodes of head to tree, below the new root

        tree.addAtRoot(new RSDepTreeNode(CP));          //create hypotactic structure
        tree.addEdge(MOD);
        tree.addAtCp(cueword);
        tree.addEdge(CMP);
        tree.addAtCp(new RSDepTreeNode(SSUB));
        tree.addEdge(BODY);

        FOR ALL nodes IN antecedent DO                  //add nodes in antecedent to subordinate clause
                tree.addToBody(node);
        result.add(tree);
END;
```

Figure 4.5: Complemening algorithm

If the selected cue word is 'om', some more adaptions need to be made. 'Om' seems to be a special case. With 'omdat', the above suffices to create a grammatical tree, when the input trees represent 'Diana stabs Brutus' and 'Diana wants to kill Brutus': 'Diana steekt Brutus neer omdat Diana Brutus wil doden'. With 'om', however, the main verb and subject of the antecedent must be deleted: 'Diana steekt Brutus neer om Brutus te doden' (Diana stabs Brutus to kill him'). So if the cue word is 'om', an extra function is called which deletes the head (=main verb) and subject of the antecedent, and which changes the syntactic category of the CP node to OTI and the category of the SSUB node to TI, following the CGN-annotation.

```
FUNCTION modify(cueword, relation)
BEGIN
        IF (addToNucleus(cueword)) DO   //add cueword to nucleus and return both trees, satellite first
                result.add(satellite);
                nucleus.addToRoot(cueword);
                result.add(nucleus);
        END ELSE DO                     //same, but other way around
                result.add(nucleus);
                satellite.addToRoot(cueword);
                result.add(satellite);
        END;
END;
```

Figure 4.6: Modifying algorithm

If the cue word is a modifier, the transformation is relatively easy. First it is checked to which tree (nucleus or satellite) the cue word should be added. The other tree is first added to the result list. The cue word is added to the root of the second tree, after which that is also added to the result list.

With some Temporal relations, after the transformation, the tense of one of the conjuncts or trees has to be altered. For instance, if a Temporal-before-gap has been transformed by adding ‘ooit’ (once) to one of the trees, the main verb of that tree must become past tense, if it is not already. A function ‘changeTense(String cat)’ is called when the transformation is finished, which checks whether it is necessary to change the tense, and if so performs this operation. So if the tree represents 'Brutus is ooit een lief jongetje', and the relation is Temporal-before-gap, the verb must become past tense, resulting in 'Brutus was ooit een lief jongetje' (Once Brutus was a nice little boy).

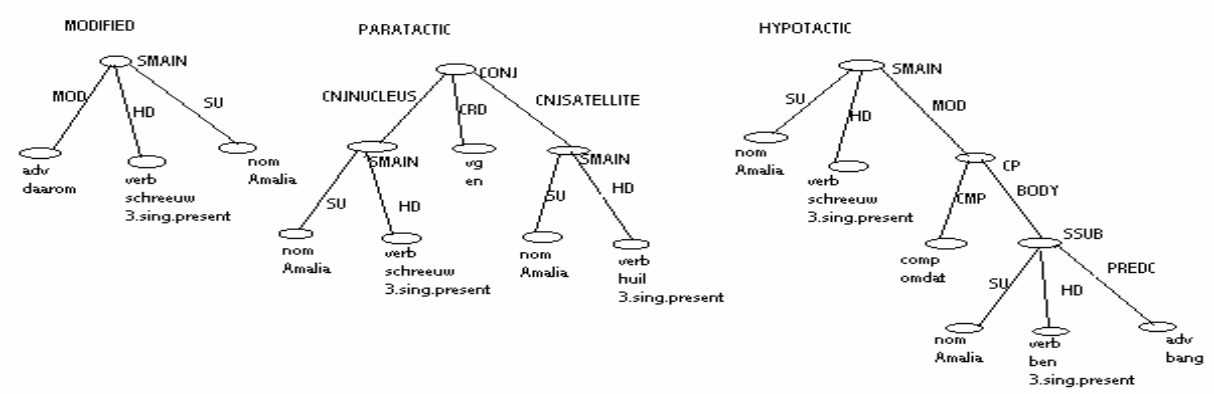


Figure 4.7: different outputs of Conjunctor

4.2.5 Elliptor

The Elliptor class is called, after the Conjunctor, to ellipt a RSDepTreeModel as much as possible. The tree is checked for identical nodes. The relation of these nodes to their parent determines which construction(s) are appropriate. The category of the RhetRelation influences this as well.

4.2.5.1Checking and marking identical nodes

The first step is to check whether the tree can be elliptic. It has to be a conjunction and the cue word must be paratactic. Further, there must be at least one child node of root that has identical syntactic category of part-of-speech tags, root forms and dependency labels, in both conjuncts. The morphology does not need to be identical. If they are leaves, their relation, pos and root-tags should be identical, if not, their relation and cat-tags should be the same, and all their child nodes as well. If two nodes are identical, their dependency labels are marked 'identical' so that they can be retrieved easily (see figure 3.11).

If one of the conjuncts is a conjunction itself, its own conjuncts are searched for any nodes labeled 'borrowed' (see section 4.2.5.2), which are then compared to the nodes in the other conjunct. If they are identical, the nodes in the other conjunct are marked while the 'borrowed' nodes stay the same. Figure 4.8 and 4.9 give a pseudo-algorithm.

When all identical nodes have been found and marked, it is determined which operations are suitable. If no nodes are marked, not structure is feasible. If only one node is unmarked, as for instance in the tree representing 'Amalia gaat naar het bos en Brutus gaat naar het bos' (Amalis goes to the forest and Brutus goes to the forest), where only the subject is not identical in both conjuncts, either a Stripping construction is chosen (resulting in 'Amalia gaat naar het bos en Brutus ook'), or the non-identical node is co-ordinated ('Amalia en Brutus gaan naar het bos'). If more than one node is unmarked, the labels of the nodes that are identical determine which operation is performed. In this case, more than one operation can be performed. For instance, when both the main verb and the subject are identical, Gapping and Conjunction-Reduction are both performed.

```
FUNCTION check(RSDepTreeModel tree) returns BOOLEAN
BEGIN
        IF NOT (tree.cat == CONJ)
                return false;
        IF NOT (cueWord.cat == VG)
                return false;
        boolean result = false;
        IF NOT ((conjunct1.cat == CONJ) OR (conjunct2.cat == CONJ)) DO BEGIN
                FOR ALL nodes  IN conjunct1 DO          //get the child with same relation, and compare
                        IF (identical(node, conjunct2.getChild(node.rel))) DO BEGIN
                                mark(n1,n2);
                                result = true;
                        END;
        END ELSE BEGIN
                IF (conjunct1.cat == CONJ) DO BEGIN
                        conjunction = conjunct1;        //at most one nested conjunction (see 2.3)
                        smain = conjunct2;
                END ELSE BEGIN
                        conjunction = conjunct2;
                        smain = conjunct1;
                END;

                FOR ALL MLabeledEdges edge IN conjunction DO
                        IF (edge.label.contains(BORROWED)) DO
                                IF (identical(edge.target, smain.getChild(edge.label))) DO BEGIN
                                        mark(edge.target, smain.getChild(edge.label));
                                        result = true;
                                END;
                END;
        END;
END;
```

Figure 4.8: Check()

```
FUNCTION identical (RSDepTreeNode node1, RSDepTreeNode node2) RETURNS BOOLEAN
BEGIN
        IF (node1.isLeaf AND node2.isLeaf)
                RETURN node1.equals(node2);
        IF NOT (node1.isLeaf OR node2.isLeaf) DO BEGIN
                FOR ALL childnodes IN node1 DO
                        IF NOT identical(childnode, node2.getChild(childnode.rel))
                                RETURN false;
                RETURN true;
        END;
        RETURN false;
END;

FUNCTION mark(RSDepTreeNode node1, RSDepTreeNode node2)
BEGIN
        IF NOT (node1.rel.contains(BORROWED))
                node1.rel.append(IDENTICAL);
        IF NOT (node2.rel.contains(BORROWED))
                node2.rel.append(IDENTICAL);
END;
```

Figure 4.9: comparing and marking nodes

4.2.5.2 Elliptical operations concerning one node

There are three elliptical structures that are realized by the removal of a single node: Gapping (where the verb of the second conjunct is removed), Right Node Raising (removal of the rightmost constituent of the first conjunct) and Conjunction Reduction (removal of the subject of the second conjunct). The functions that handle this all use the function Ellipt(), which

removes an identical node from one conjunct, and lets its parent point to its twin in the other conjunct, giving it the label 'borrowed' (see figure 4.10).

```
FUNCTION Ellipt(String label, RSDepTreeNode intact, RSDepTreeNode adapt)
BEGIN
        RSDepTreeNode shared = intact.getChildNode(label);          //get identical node in intact
        String newLabel = label.replace(IDENTICAL, BORROWED);
        IF (shared == null)
                shared = intact.getChildNode(newLabel);

        adapt.removeChild(label);                                   //remove twin in adapt
        MlabeledEdge edge = new MlabeledEdge(newLabel);             //connect adapt to identical node in
        edge.setSource(adapt);                                      //intact
        edge.setTarget(shared);
END;
```

Figure 4.10: Function Ellipt()

The functions that handle Right Node Raising and Gapping introduce a new constraint: only additive or contrast relations, as Resemblance relations, can get such a construction (see section 4.2.6.1). In classic RST, these are multi-nuclear relations – if someone ever decides to implement multi-nuclear relations, they might constitute a good criterium to use at this point. Conjunction Reduction is feasible with all relations.
The conjunct that is adapted by Ellipt() is marked 'CNJ-ELLIPTED', unless the operation was Right Node Raising. Then the conjunct is marked 'CNJ-RAISED', because with Right Node Raising something is deleted from the first conjunct of the Surface Form, while with the other operations a node from the second conjunct is removed. So the OrderedDepTreeModel needs different labels to determine which conjunct comes first.
Which conjunct is first, and which second, and which gets adapted, depends on which is nucleus and which is satellite. The satellite is always first, the nucleus second (put the cause before the consequence). This means that with Right Node Raising the satellite is adapted while the nucleus stays intact. With the other elliptic structures, it is the other way around.

4.2.5.3 Ellipsis removing all but one node

If all nodes but one were found to be identical, the transformation is more radical. Two constructions can be chosen: Stripping and coordinating the unmarked node. With Stripping, all the identical nodes of the nucleus are deleted, while the root is connected to their twin in the satellite, with the edges marked as 'BORROWED'. Then a new node is added, a node representing the modifier 'ook' (too), to realize a construction such as 'Brutus gaat naar de woestijn en Amalia ook' (Brutus goes to the desert and Amalia does too'. If the relation is nested, the modifier 'ook' becomes a shared node as well.
To coordinate a constituent, the non-identical node in nucleus is replaced by a conjunction between that node and its twin in the satellite. The nucleus is then returned. If the relation is nested, a third conjunct is added to the conjunction that is already present.
When the Elliptor class is finished, a RSDepTreeModel will have ellipted as much as possible, to ensure that the generated sentences are clear and unrepetitive.

4.2.6 Referential Expression Generator

The ReferentialExpressionGenerator selects referential expressions for characters, objects and locations, if they are marked as recent in the CharacterHistory (see section 4.4.2). Male characters receive 'hij' or 'hem', female characters 'zij' or 'haar', objects 'het' and locations 'er',

'erheen', and so on. 'hij en zij' is transformed again to 'wij'. Later on, perhaps other referential expressions could be added, such as 'the princess' or 'the villain'.
Take for instance the sentences 'Amalia ziet de prins. Amalia slaat Brutus en roept de prins' (Amalia sees the prince. Amalia hits Brutus and calls the prince). The first 'Amalia' does not receive a pronoun, because Amalia was not recently mentioned. It is now set to recent, with dependency label 'Subject'. The same happens to the first 'de prins', which is set to recent with dependency label 'Object1'. Then Amalia is mentioned again, as a subject, as so is recognized as recent and receives a pronoun, 'she'. When 'Brutus' is encountered, he is set to recent (and so replaces the prince, because they are both male) with dependency label 'Object1'. So when the prince is mentioned again, he does not receive a pronoun, as he is not marked recent anymore.

4.3 Libraries

This section describes the natlang.rdg.libraries package, which contains all language-specific information that the Surface Realizer needs. There are libraries containing morphological information, cue words or grammatical rules.

4.3.1 MorphLib, VerbLib and NounLib
MorphLib is an abstract class which contains functionality needed by all classes that inflect the root-tag to a word, such as reading the desired irregular inflection from a file, and checking whether a character is a vowel. There are, at the time of writing, three extending classes: NounLib, VerbLib and DetLib.
NounLib inflects nouns. The singular form serves as the root, from which the plural form can be derived if the noun is regular. First, however, it is checked if the noun is irregular, by searching the root in a irregular noun file. If the root is not found, it receives a regular inflection (see appendix B for the algorithm). Most Dutch nouns receive either 'en' or 's' as plural inflection, depending on the last sound segments of the singular form.
VerbLib inflects all verbs to the requested tense. Irregular verbs are looked up in a file. If the given root is not found, it is inflected regularly. The most common rules have been implemented (again, see appendix B).
Detlib inflects the determiners to plural form, if the morph-tag specifies that they should be. 'De' and 'het' become 'de', 'een' becomes zero1 (Amalia loves a cat – Amalia loves cats).

4.3.2 ConstituentLib
ConstituentLib is the parent class of all classes that contain grammatical rules to produce the surface form. For every syntactic category there is a class that contains rules, stating what children a node of that category can have, and how they should be ordered. For instance, class SmainLib contains (among others) the following rules:

SMAIN -> SU HD
SMAIN -> SU HD OBJ1
SMAIN -> SU HD OBJ1 OBJ2
SMAIN -> MOD HD SU
….

When the Surface Form is produced by OrderedDepTreeModel, a suitable rule is selected for every node by calling the related class and picking a rule that has the same children as the node. When more than one rule fullfills the requirements, the Discourse History is used to select (see section 4.4).

4.3.3 CueWordLib

CueWordLib is the parent class of all classes that contain cue words suited to a particular rhetorical relation. At the moment, there are five extending classes: CauseLib, PurposeLib, ContrastLib, AdditiveLib and TemporalLib. More rhetorical relations can easily be added. Given the category of a rhetorical relation, CueWordLib returns the extending class you need. It can also check whether a cue word has to be added to the nucleus or to the satellite, by looking at its relation.

The extending classes contain lists of cue word nodes. The lists are divided according to certain principles derived from the cue word taxonomy (see section 3.3). Given the category of a rhetorical relation (which includes the primitives), they return a list of suitable cue words. The list starts with the most specific cue word, so that specific cue words are more likely to be selected than less specific words. As less specific cue words are appropriate in far more situations, this evens out again over longer texts.

4.4 Discourse Histories

The Discourse package (natlang.rdg.discourse) contains the classes that keep track of objects, cue words, grammatical rules etc. that have recently been used. This information is used to prevent unnecessary repetitions.

4.4.1 CueWordHistory

This DiscourseHistory keeps track of the cue words that have been used in the last three sentences. It contains a list with three elements, all recently used cue words. If the CueWordHistory is informed that a cue word has been used, it is set to the first position in the list. The CueWordHistory checks whether a cue word is recent by comparing it to the cue words in the list.

Take, for instance, a Cause-Involuntary relation. CueWordHistory will receive a list of feasible cue words, and return the most specific cue word that has not recently been used (in this case, 'doordat' or 'daardoor'). If all the cue words are recent, the cue word that has not been used longest is returned.

4.4.2 CharacterHistory

This DiscourseHistory keeps track of the characters, objects and locations in the story domain that have recently been mentioned. There is a male, a female, a neutral and a location slot. A given RSDepTreeNode is compared to the nodes in each slot, and if it corresponds to one of them (specifically, in its dependency relation and its root) it is pronounced to be recent. If it does not correspond to any of them, it is pronounced not to be recent. Then its gender is determined and the right slot is filled with the node, because this is now the node that was last used. The slot's counter is set to zero. If the given node does correspond to the node in a slot, the counter of that slot is increased by one. If a counter reaches 3, the slot is reset, to prevent a

continuous and ultimately confusing use of pronouns. This way, a character is named by its full name at least once in every three sentences.

4.4.3 RuleHistory

This DiscourseHistory keeps track of which rules have been used to order the child nodes of a certain grammatical constituent. Only SMAIN, SSUB and TI are tracked, as only these can have more than one suitable rule for a certain list of child nodes. Each of these constituents has a list with three elements, indicating which rules were used recently. If all possible rules are recent, the least recent rule can be selected.

4.5 View

The natlang.rdg.view package contains three classes to view the intermediate stages of the Surface Realization process. The RSGraphViewer reads a Rhetorical Dependency Graph from a xml-file and creates a graph that shows all the relevant information on the screen. It cannot show an existing RSGraph.

The RSDepTreeViewer creates a viewable Dependency Tree. In the process of showing the tree, it destroys it, so the tree cannot be used for further processing. The RDGEditor takes a RSDepTreeViewer or a RSGraphViewer, and shows them in a scrollable JFrame. This image can be saved as well.

5. Results

This section shows some representative output of the Surface Realizer. Its input was the plot of a story, based on the stories shown in section 1.2 (repeated in figure 5.1). Appendix A shows all sentences that were generated.

Verhaal 1
Er was eens een prinses. Ze heette Amalia. Ze bevond zich in Het kleine bos.
Er was eens een schurk. Zijn naam was Brutus. De schurk bevond zich in De woestijn2.
Er ligt een Zwaard. in De bergen.
Er ligt een Zwaard. in Het grote bos.
Amalia loopt naar de woestijn1.
Brutus loopt naar de woestijn1.
Amalia ervaart angst ten opzichte van Brutus vanwege de volgende actie :
Amalia ziet Brutus
Amalia loopt naar de kale vlakte.
Brutus loopt naar de kale vlakte.
Amalia ervaart angst ten opzichte van Brutus vanwege de volgende actie :
Amalia ziet Brutus
Amalia loopt naar de bergen.
Brutus loopt naar de bergen.
Amalia pakt zwaard op.
Brutus ervaart angst ten opzichte van Amalia vanwege de volgende actie :
Amalia pakt zwaard op.
Brutus schopt de mens.
Amalia steekt de mens neer.
en ze leefde nog lang en gelukkig!!!

Verhaal 2
Er was eens een prinses. Ze heette Amalia. Ze bevond zich in Het kleine bos.
Er was eens een schurk. Zijn naam was Brutus. De schurk bevond zich in De woestijn2.
Er ligt een Zwaard. in De bergen.
Er ligt een Zwaard. in Het grote bos.
Amalia loopt naar de woestijn1.
Brutus loopt naar de woestijn1.
Amalia ervaart angst ten opzichte van Brutus vanwege de volgende actie :
Amalia ziet Brutus
Amalia gaat het kasteel binnen.
Brutus gaat het kasteel binnen.
Amalia ervaart angst ten opzichte van Brutus vanwege de volgende actie :
Amalia ziet Brutus
Amalia slaat de mens.
Brutus schopt de mens.
Amalia schreeuwt.
Brutus slaat de mens.
Amalia schreeuwt.
Brutus pakt de mens op.
Amalia schreeuwt.
Brutus ervaart hoop ten opzichte van Amalia vanwege de volgende actie :
Brutus pakt de mens op.
Brutus neemt in het kasteel de mens gevangen.
en de mensen spraken jaren later nog over deze treurnis

Figure 5.1: stories generated by Virtual Storyteller

The stories in figure 5.2 are based on the stories shown above. There is one slight difference: Brutus is in the beginning of the story no longer in the second desert, but in the swamp (having two separate deserts seemed unnecessarily confusing). The Surface Realizer received

as input Dependency Trees corresponding to the sentences the Virtual Storyteller generated, with some appropriate rhetorical relations added. The input for the first two sentences of the stories is shown in figure 5.3. Most of the relations I added were causal relations that the system would know about. We can see that for the same situation, different expressions are possible. The connections between clauses, and so the connections between actions, emotions etc., are explicitly stated, causing the story to be more coherent. Also, because of ellipsis and Referential Expression Generation, there is less repetition of words, causing the individual sentences to be more concise. The length of the sentences varies, which is essential to prevent a monotone recital.

1
Er was eens een prinses.
De prinses heette Amalia en bevond zich in het klein bos.
Er was eens een schurk.
De schurk heette Brutus en bevond zich in het moeras.
Er lag een zwaard in het groot bos en ook in de bergen.
Brutus ging, terwijl Amalia naar de woestijn ging, erheen.
Doordat zij Brutus zag, was zij bang.
Brutus ging het kasteel binnen, want zij ging het kasteel binnen.
Omdat zij Brutus zag, was zij bang.
Amalia sloeg hem.
Brutus schopte Amalia en Amalia schreeuwde.
Doordat hij Amalia sloeg, schreeuwde Amalia.
Zij schreeuwde, maar hij was hoopvol, omdat hij Amalia op pakte.
Hij nam haar gevangen in het kasteel.
Nog jaren spraken de mensen over deze treurnis.

2
Er was eens een prinses.
De prinses heette Amalia en bevond zich in het klein bos.
Er was eens een schurk.
De schurk heette Brutus en bevond zich in het moeras.
Er lag een zwaard in het grote bos en ook in de bergen.
Brutus ging, terwijl Amalia naar de woestijn ging, erheen.
Doordat zij Brutus zag, was zij bang.
Brutus ging naar de kaal vlakte, want zij ging naar de kaal vlakte.
Omdat zij Brutus zag, was zij bang.
Amalia ging naar de bergen en Brutus ging erheen.
Hij sloeg Amalia, want, doordat Amalia het zwaard op pakte, werd hij bang.
Nog lang en gelukkig leefde zij, nadat zij Brutus neer had gestoken.

Figure 5.2: Same stories, text generated by the Surface Realizer

However, it is far from perfect. The lack of a class to inflect adjectives shows in terms as ‘de kaal vlakte’, which should be ‘de kale vlakte’. The word ordering is sometimes awkward or plainly incorrect. The awkward order is mainly caused by the problems I encountered when trying to find grammatical rules to order modifiers. For instance, the subsentences are all nested (Brutus ging, terwijl Amalia naar de woestijn ging, erheen), while the text would be clearer without nested clauses. The group of modifiers is too large, encompasses too many different things, to let them be covered by the same rules (see chapter 8 for a further discussion). Some incorrect things are caused by a word that should, because of the structure of the rest of the sentence, be split up, such as ‘erheen’. This is hard to mend because ‘er’ surfaces outside its parent category. Other words should be glued together, such as ‘op pakte’. In a main sentence, these are separate, but in a subordinate sentence, they become one word.

There is also one imperfection that is not due to the Surface Realizer, but rather to the output of the VS. Relations are now explicitly signalled by cue words. However, most of the (seven) people that I showed these narratives to seemed more confused by statements that one action causes another without a recounting of the underlying goals or reasoning, than by the original narratives where the actions were stated without any relations. This is not a shortcoming of the Surface Realizer, but is caused by the lack of a Content Planner. For this reason, I have adapted the input. I have added a few sentences concerning goals and motives. Amalia goes to the desert to pick flowers (not a very logical occupation, I admit, but then fleeing into the castle is not very smart either), Brutus follows Amalia into the castle because he wants to imprison her, and so on. On the other hand, I have removed the information about the location of the swords from the first story, because they have no part in the rest of the narrative. In the second story, the information about the sword in the mountains is recounted at the moment that Amalia finds it. The Virtual Storyteller, in generating the first narrative, knew there is a castle in the desert, but saw no reason to narrate this. I have added a sentence that says there is a castle in the desert, so Amalia entering the castle is not so wholly unexpected as in the original narrative.
I have also added more relations. For instance, Amalia now sees Brutus 'suddenly' in the desert, to underline that she did not expect to see him there. The resulting narratives show how well the Surface Realizer could perform in combination with a good Content Planner. They are given in figure 5.4.

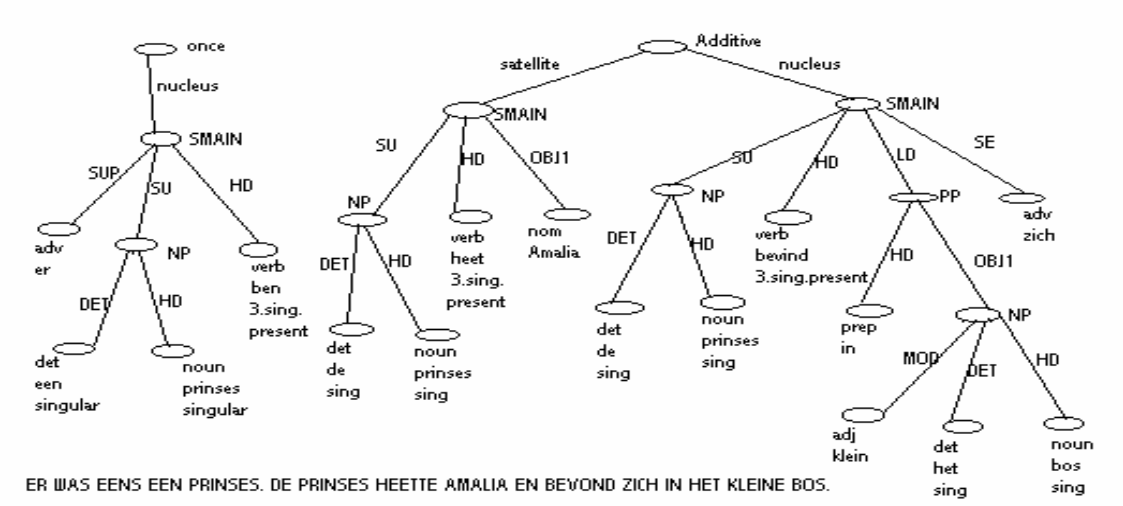


Figure 5.3: Input for the first two sentences of the narrative

Unfortunately, I did not have time to evaluate the generated narratives properly. However, the people that I showed the stories to agreed that the narratives in figure 5.4 make more sense than those in figure 5.2. Research by Callaway & Lester (2001) indicated that aggregation (they call it revision) is an important aspect of Narrative Prose Generation. Test subjects showed a preference for narratives on which revision was performed, over narratives without revision.
The new narratives of the Virtual Storyteller consist of the same actions as the older narratives, but those actions are now motivated by reasons. Also, necessary background information such as ‘There is a castle in the desert’ is given, so the listener will not have to wonder where the castle suddenly came from, and so on. There is a definite improvement when compared to the original output of the VS. Instead of a dry recitation of short facts, we get a coherent and cohesive narrative, with varying sentence length and structure.

1
Er was eens een prinses.
De prinses heette Amalia en bevond zich in het klein bos.
Er was eens een schurk.
De schurk heette Brutus en bevond zich in het moeras.
Amalia ging naar de woestijn, want zij wilde bloemen plukken.
Brutus ging, om de omgeving te verkennen, tegelijkertijd erheen.
Er stond een kasteel daar.
Amalia wilde vluchten, want, doordat zij Brutus plotseling zag, was zij bang.
Zij ging dus het kasteel binnen.
Brutus ging het kasteel binnen, want hij wilde Amalia gevangen nemen.
Doordat Amalia Brutus zag, was zij bang.
Omdat Brutus Amalia, nadat Amalia Brutus geslagen had, schopte, schreeuwde zij.
Brutus pakte, om Amalia te opsluiten, haar op.
Hij was hoopvol, maar Amalia bang.
Hij nam Amalia gevangen in het kasteel.
Nog jaren spraken de mensen over deze treurnis.

2
Er was eens een prinses.
De prinses heette Amalia en bevond zich in het klein bos.
Er was eens een schurk.
De schurk heette Brutus en bevond zich in het moeras.
Amalia ging naar de woestijn, want zij wilde cactussen plukken.
Brutus ging, om de omgeving te verkennen, tegelijkertijd erheen.
Zij wilde vluchten, want, doordat zij Brutus plotseling zag, was zij bang.
Zij ging dus naar de kaal vlakte.
Brutus ging naar de kaal vlakte, want hij wilde Amalia gevangen nemen.
Doordat Amalia Brutus zag, was zij bang.
Brutus ging naar de bergen, want zij ging, om te vluchten, erheen.
Er lag een zwaard daar.
Zij wilde Brutus doden.
Omdat Brutus, doordat Amalia daarom het zwaard op pakte, bang werd, werd hij agressief.
Hij sloeg Amalia.
Nog lang en gelukkig leefde zij, nadat zij Brutus neer had gestoken.

Figure 5.4: Surface Realizer with help from Content Planner

The stories unfortunately seldom produce ellipsis. The characters never do anything together, so there are hardly any Additive relations, which are most convenient for ellipsis. For this reason, I will give some other examples here that were generated by the program, to demonstrate that the program can indeed produce all the ellipsis that we desire. All examples consisted of two Dependency Trees connected by an Additive relation.
The structures we desired were Stripping, Gapping, Conjunction-Reduction, Right Node Raising and the conjunction of a single constituent. For every category I will give a few generated sentences:

Conjunction-Reduction:
De prins zit op een terrasje en drinkt een biertje. (input: De prins zit op een terrasje. De prins drinkt een biertje)
Amalia gilt en schreeuwt. (input: Amalia gilt. Amalia schreeuwt)

Gapping:
Amalia gaat naar het bos en Brutus naar de woestijn. (Amalia gaat naar het bos. Brutus gaat naar de woestijn)
Amalia houdt van de prins en de prins van Amalia. (Amalia houdt van de prins. De prins houdt van Amalia)

Right Node Raising:

De prins gaat te paard en Amalia te voet naar de bergen. (De prins gaat te paard naar de bergen. Amalia gaat te voet naar de bergen)
De prins rijdt en Amalia loopt naar de bergen. (De prins rijdt naar de bergen. Amalia loopt naar de bergen)

Stripping:

Amalia gaat naar de woestijn en Brutus ook. (Amalia gaat naar de woestijn. Brutus gaat naar de woestijn)
Amalia is boos en ook bang. (Amalia is boos. Amalia is bang)
Amalia gaat morgen naar het bos en vandaag ook. (Amalia gaat morgen naar het bos. Amalia gaat vandaag naar het bos)

Constituent-Coordination:

Amalia danst en zingt. (Amalia danst. Amalia zingt)
De prins en Amalia gaan te paard naar de bergen. (De prins gaat te paard naar de bergen. Amalia gaat te paard naar de bergen)
Amalia wandelt naar het bos en naar de bergen. (Amalia wandelt naar het bos. Amalia wandelt naar de bergen)

The full set of generated sentences is given in Appendix A. As we can see, all the desired forms of ellipsis were generated. The first three forms can be performed simultaneously as well: for instance, in the first example of RNR, Gapping has been performed as well. Nested relations are also handled well. We can see this in the stories in figure 5.2 and 5.4, but figure 5.5 gives a few extra sentences, to show that ellipsis can be performed in nested relations as well. Only the Rhetorical Relations in the input are shown, not the Dependency Trees, to prevent the figure from getting too complicated.

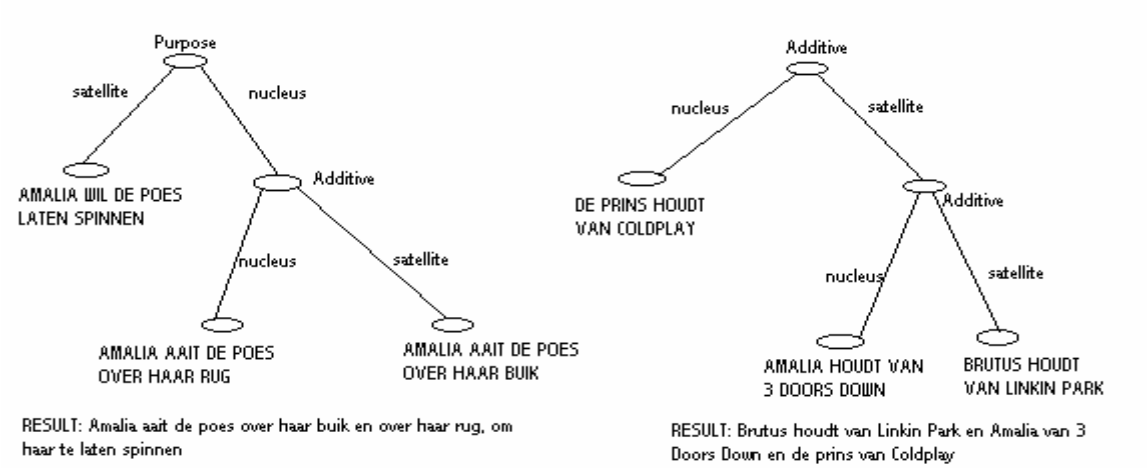


Figure 5.5: input and output for nested relations

The inflection and ordering of the words, i.e. the production of the Surface Form, is not yet optimal, but the goal that was set at the beginning, to perform syntactic aggregation and to generate ellipsis, has been fulfilled.

6. General Discussion

The Surface Realizer that was built in this project is capable of combining two or more clauses in various ways, using different cue words and producing several elliptic structures. Depending on the type of relation and the number of constituents in the clauses that are identical, as much as five different surface forms can be produced to express it. This will certainly improve the variety of the generated text, as we wanted at the start of the project.
The goal has been accomplished – but is the method satisfactory? This section discusses the choices that were made in the construction of the Narrator and the Surface Realizer. First we defend the situating of syntactic aggregation in the Surface Realizer. Then we discuss the suitability of Dependency Trees as input for the Surface Realizer, and finally the necessity of adding rhetorical relations between them.

6.1 Syntactic Aggregation in the Surface Realizer

As discussed in section 2.1.6, one of the findings of the RAGS-project was that there is no consensus on the right location for aggregation to take place. This is at least partly caused by the use of different definitions of aggregation, which comprise processes quite different in level. So-called lexical aggregation (for example: 'Saturday and Sunday' becomes 'weekend') has to take place at the lexicalisation level, because this is a very language-dependent process – not all languages have words for certain concepts, and so in some languages something could be lexically aggregated, while in others it could not. As the Lexicon that is used in lexicalisation already contains all the concepts of a language, the lexicalisation level is the logical place for lexical aggregation. Syntactic aggregation, however, concerns itself with grammatical processes and therefore would be better placed in the Surface Realizer.
There is evidence that ellipsis is language-dependent as well. Not all forms of ellipsis are permissible in all languages. Some forms depend on word order, so it seems that ellipsis is influenced by the same grammatical rules that perform linearization. The Surface Realizer already has access to these rules, an extra argument to perform syntactic aggregation (and the generation of ellipsis) in the Surface Realizer. However, the Surface Realizer, at the time of writing, does not use such rules while performing ellipsis, as the forms of ellipsis that are implemented often occur in other languages that are related to Dutch (English, German) as well.

6.2 Dependency Trees

Is the Dependency Tree language-independent? Mel'cuk (1988) designed Dependency Trees to be free of word order to allow for languages where the word order is vastly different from English. But is it only word order that makes languages differ from one another? In Latin, it is not necessary to mention the subject of a sentence – and that certainly shows in a Dependency Tree. And all languages have some concepts that do not translate well, though these might not crop up often in the telling of a simple fairy tale. Still, even if the Dependency Trees that the Surface Realizer receives as input are not completely language-independent, the methods to process them and turn them into Surface Forms *are*. Substituting the cue words and grammatical rules should be sufficient to enable the Surface Realizer to process Dependency Trees lexicalised to a different language. For this reason alone, I think Dependency Trees are excellent input for a Surface Realizer that tries not to commit itself to one language. Another advantage of using Dependency Trees in syntactic aggregation is that they can easily be

manipulated, because the role a word or a constituent plays in a sentence is given by a label. For instance, if you want subject-deletion, you can simply delete the node labelled 'subject'.

6.3 Discourse Structures

Discourse structures do not seem to be affected by the language that is used. Are there languages in which one can't signal rhetorical relations? Such basic relations as Cause and Contrast? Dependency Trees are supposed to be language-independent – I claim that the rhetorical relations that hold between the propositions represented by the trees are as well. Because these relations are neutral, they can be processed at the latest stage, after lexicalisation, in the Surface Realizer. This also puts less strain on the component that has to create the Dependency Trees. The trees only represent simple facts ('Diana is scared', 'Diana flees'), which makes them fairly simple themselves.

The rhetorical relations can determine which syntactic construction should be used, and rule out others (such as a gapped sentence in a causal relation). Rhetorical relations are an excellent mechanism to carry a certain meaning across to the level when it is finally of use. And if the relations even influence the grammatical structure that an elliptic sentence can have, the Surface Realizer certainly should have access to them.

However, the relations that were used in this project were not the set that is given by Mann & Thompson (1987). Only a few relations were selected for the time being, those deemed of the most importance to the narrating of a fairy-tale, or the production of ellipsis. Cause and Contrast are very basic concepts, and Temporal relations are vital for any narrative. These relations were then divided into subclasses that correspond to groups of cue words. The cue words were derived from a small cue word taxonomy that was created for this purpose. The properties that distinguish the subclasses are moulded in terms of information that is available to the story generation system. This way, rhetorical relations can easily be added between propositions, because the information is really already there. The primitives that we currently distinguish in the taxonomy were selected based on linguistic evidence, on the groups of cue words that were determined. They can be adapted to information already present in the Virtual Storyteller. For example, volitional cause corresponds to plans, actions in the pursuit of a goal, involitional cause to character attitudes. In the future, our cue word grouping should be experimentally confirmed, and if it is not confirmed, the taxonomy should be adapted. Because, as Reape & Mellish (1999) have said, NLG systems should be based on linguistic theories and linguistic evidence to be truly successful.

7. Conclusion

In this thesis, I have argued toward the following conclusions:

- The most appropriate place for syntactic aggregation is at the level of the Surface Realizer
- The combination of Dependency Trees and rhetorical relations is excellent input for such a Surface Realizer, because Dependency Trees are easily manipulated and rhetorical relations are a good determinant for the syntactic constructions that can be used

The text generated by the Surface Realizer has not been evaluated. However, according to Callaway & Lester (2001), aggregation (revision) is an important part of Narrative Prose Generation. Without it, the text seemed "choppy" or ungrammatical.
Using syntactic aggregation, generated sentences can vary a great deal in their syntactic structure and a little in lexical choice (different cue words), even if they communicate the same propositions. I feel certain that any narrative, and most other texts as well, are improved by (syntactic) aggregation.

Because syntactic aggregation takes place in the Surface Realizer, the Dependency Trees only have to represent simple facts, which reduces their complexity. This means that the tasks of the Clause Planner, the module that will have to generate the Dependency Trees, should not be too difficult to handle. Additionally, the rhetorical relations are adapted to the knowledge that is present in the Virtual Storyteller, so the Content Planner should be able to add these as well. In the construction of at least rudimentary Content and Clause Planners, no unsolvable problems should be encountered.

Much work remains to be done in the Narrator-module. The Surface Realizer itself is far from perfect. The Referential Expression Generator does not function as well as I'd like, applying pronomina when it should not and failing to when it should. The number of grammatical rules should be expanded, and the 'modifier-problem' (see section 8) should be solved.
The cue word taxonomy should be constructed anew, based on experiments using the substitution test of Knott & Dale (1993). I also think it should only contain cue words that are used regularly in fairy tales, perhaps taken from a corpus of fairy tales. Certainly, cue words such as 'daarentegen' (on the other hand) or 'desalniettemin' (nevertheless) are not often used when speaking to children. The Narrator of the Virtual Storyteller should expect an audience of children, and especially as the narrative will be spoken, the text should not be too complicated or contain difficult words. This is also something to keep in mind in the construction of the Clause Planner and its lexicon.

A very important task lies in the constructing of the Content Planner, and especially in the selection of facts that must be told, and the discarding of superfluous information. The main reason that the generated stories are still somewhat boring, is that every event is narrated, and its effect with it, while the listener often only needs to hear it just once. For instance, if Brutus hits Amalia three times, we do not really need to hear every time that she screams as a result. Conversely, adding more goals and background information will render the narrative more cohesive and comprehensible.
The Content Planner is also responsible for adding rhetorical relations. A careful mapping of the connections known to the system, to the existing categories of rhetorical relations, should render the story far more comprehensible, as well as inducing more variety as a result of syntactic aggregation. To specify why something happens, or why a character reacts in a

certain way, or which goals he or she has, will perhaps bring us even closer to a gripping narrative, than improvement of the plot would. How you tell a story is just as important as its plot.

The Narrator is at the time of writing only capable of syntactic aggregation and referential expression generation. However, we have seen in chapter 2 that there are different kinds of aggregation as well. In the Clause Planner, lexical aggregation should take place. Enumerations could be substituted for more general terms. In the example above, where Brutus hits Amalia three consecutive times, it could add the word '(al)weer' (again), 'nog steeds' (still), etc. This would probably prove to be as great an improvement as syntactic aggregation has.

Then there is the matter of prosody. The Virtual Storyteller has a Speech Agent that transforms the Surface Form to speech. This Speech Agent would function better if its input contained prosodic information. Especially if a sentence has been aggregated, we need to stress certain parts of it to ensure that the listener understands it.
Enough work remains to be done. However, I feel satisfied that this project has taken the Narrator a distinctive step forwards toward a text narrative that is grammatical, varied and concise.

8. Future work

Although the Surface Realizer is already capable of producing several sentences for the same input that can vary quite widely, there are of course still some tasks that should be performed.
On the one hand, some of the work that has been done should be done more thoroughly. Some components that were not part of the main issue of this project were only implemented shallowly. The number of grammatical rules in the Constituent libraries, for example, is rather small and should be expanded. The same is true for the number of rhetorical relations. Expanding these would enable the generated sentences to communicate more complex meanings. For instance, apart from Additive relations you could have Elaboration, apart from Purpose you could have Justify and Reason, etc.
The cue word taxonomy should also be expanded. Even better would be to construct it again using the substitution test on test subjects instead of just one person. The taxonomy constructed in this project was a tentative attempt to see if it would be useful; now that it has proven its use, a new, better taxonomy should take its place. As we discussed in section 3.2.1 perhaps an extra primitive should be added to it, to distinguish between cue words that lay emphasis on either cause or consequence, reason or action, to provide a cue for the listener which clauses are essential to understanding the story.
Finally, the Referential Expression Generator at the moment only produces pronouns, and does not always do that properly either. If it also produced descriptions like 'the princess' this would improve variety as well. Appendix B gives detailed instructions on how to extend the number of grammatical rules, rhetorical relations and cue words.

On the other hand, there are tasks that have not been implemented at all but are badly needed. The most important and obvious is the need for a Content and a Clause Planner. The Content Planner should at least contain functionality to select pertinent information, and to specify the rhetorical relations that hold between the chunks of information. The Content Planner should add paragraph boundaries as well. Finally, it would be a very interesting task to try to tell a story from different perspectives, or to tell it in a different than chronological order.
The Clause Planner should at least be able to transform the Text Plan it receives from the Content Planner to a Rhetorical Dependency Graph. For that, it must be able to create a Dependency Tree out of a logical proposition. It will need an extensive lexicon. Any lexical aggregation should be performed in the Clause Planner as well.
The Narrator should be connected to the rest of the Virtual Storyteller. The Narrator should receive all the information it needs from the Director and the Actors, and communicate the Surface Forms it produces to the Speech Agent. To accommodate the Speech Agent, the Narrator should also add prosodic information to its Surface Forms, so that they could be pronounced with the right emphasis. An evaluation of storytelling speech by Meijs (2004) indicated that listeners prefer manipulated speech, which has climaxes at key points of the story, as more natural dan the flat speech of a neutral speech generator.

Finally, there is a problem that has arisen from the Dependency Tree approach. The ordering of modifiers, in the process of ordering the Dependency Tree and producing the Surface Form, is flawed. In Alpino Dependency Trees, 'modifier' applies to a large number of constituents. Adjectives such as 'too' or 'not', indicators of time such as 'yesterday', embedded sentences ('because…..') all receive the label 'mod'. However, some of these groups appear at different positions in a sentence than others. For instance, you could say 'Amalia huilt omdat ze boos is', but 'Amalia huilt daarom' does not sound quite right. As 'daarom' (that's why) points to the sentence before, you'd expect it at the start of the second sentence. However,

'Amalia lacht en ook huilt' is not right either. A solution would be to differentiate the 'mod' label, so that the kind of modifier can be communicated. It is not possible to come up with a rule that produces the correct surface form in all situations, if we leave the modifier-label as it is.
Some kinds of ellipsis, especially some sorts of Right Node Raising, depend on surface structure, not constituents. The current system cannot performs this kind of ellipsis, because it ellipts before the tree is ordered. It may however not be too difficult to change this: the Conjunctor and the Elliptor should be able can take an ordered Dependency Tree as well as an unordered one, and then the order of the subjects could be used to determine whether Right Node Raising can be performed. A drawback to this approach would be that the added nodes would have to be ordered again after the Conjunctor and the Elliptor have finished.

Unfortunately, there was no time to evaluate the generated sentences and stories properly. This should be done in the future. One way to evaluate the quality of the generated stories would be to let test subjects compare them to fairy tales taken from a text book, and let them mark anything confusing or ungrammatical.
The current system ellipts constituents as often as possible. However, there is no evidence that humans do the same (although there is no evidence to the contrary either). A corpus study could indicate whether the frequency of ellipsis, and the kind of elliptic structures, in the generated text, are similar to the frequency of ellipsis in natural language.

Obviously, there is plenty of work left if we want to achieve a full NLG system. However, one of the standard three modules of a NLG-system has been produced and performs well, a good beginning.

References

Andersen, Hans Christian; Sprookjes en Vertellingen; translated 1975 by W. van Eeden; Bussum, Van Holkema en Warendorf, 1975

Bateman, J.A. & Teich, E.; 1995; Selective Information Presentation in an Integrated Publication System: an Application of Genre-driven Text Generation; in *Information Processing and Management*, Volume 31(5): pages 753-767, 1995

Bouma, Van Noord & Malouf; 2001; Alpino: Wide Coverage Computational Analysis of Dutch. In: *Computational Linguistics in the Netherlands CLIN 2000*

Britton, B. K., Glynn, S. M., Mayer, B. J. F., & Penland, M. J.; 1982; Effects of text structure on use of cognitive capacity during reading; in *Journal of Educational Psychology, 74,* pages 51–61.

Cahill, L. & Reape, M.; 1999; Component tasks in applied NLG Systems; Information Technology Research Institute Technical Report Series

Callaway, C.B. & Lester, J.B.; 2001; Evaluating the Effects of Natural Language Generation Techniques on Reader Satisfaction; in *Proceedings of the Twenty-Third Annual Conference of the Cognitive Science Society* (CogSci 2001), pp. 164-169, Edinburgh, UK, August 2001

Caldwell, D. & Korelsky, T.; 1994; Bilingual generation of Job Descriptions from Quasi-Conceptual forms; in *ANLP'94*, pages 1-6

Dalianis, H.; 1999; Aggregation in Natural Language Generation; in *Computational Intelligence*, Volume 15, Number 4, 1999.

Faas, S.; 2002; Virtual Storyteller: An approach to computational story telling; Masters thesis at University of Twente

Hendriks, Petra; (2004); Coherence Relations, Ellipsis, and Contrastive Topics; in *Journal of Semantics* 21:2, pp. 133-153.

Hovy, E.;1993; Automated Discourse Generation using Discourse Structure Relations; *in Artificial Intelligence*, pages 341-385

Knott, Alistair; 1995; An Empirical Methodology for Determining a Set of Coherence Relations; In *Proceedings of the AAAI Spring Symposium on Empirical Methods in Discourse Interpretation and Generation*, Stanford, 1995

Knott, Alistair & Sanders, Ted; 1998; The Classification of Coherence Relations and their Linguistic Markers: An Exploration of Two Languages; In *Journal of Pragmatics*, Volume 30, pages 135-175; Elsevier, 1998

Knott, Alistair & Dale, Robert; 1993; Using Linguistic Phenomena to Motivate a Set of Coherence Relations; in *Discourse Processes*, 18(1), 35-62

Knott, Alistair & Dale, Robert; 1996; Choosing a Set of Coherence Relations for Text Generation: A Data-Driven Approach; in *Trends in Natural Language Generation: an Artificial Intelligence Perspective (G. Adorni & M. Zock, eds.)* pages**:** 47-67; Springer-Verlag, Berlin, 1996
Lavoie, B., Kittredge, R., Korelsky, T., Rambow, O.; 2000; A Framework for MT and Multilingual NLG systems based on Uniform Lexico-structural Processing; In *Proceedings of ANLP/NAACL 2000*, Seattle, Washington.

Lavoie, B. & Rambow, O.; 1997; A Fast and Portable Realizer for Text Generation Systems; in *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 265-268

Levin, Nancy S. & Prince, Ellen F.; 1986; Gapping and Causal Implicature; in *Papers in Linguistics* 19.3, 351-64

Malouf, R.; 2000; The order of prenominal adjectives in Natural Language Generation; in *Proceedings of the 38th ACL*, Hong Kong

Mann, William C. & Thompson, Sandra A.;1987; Rhetorical structure theory: A theory of text organization; in *Technical Report ISI/RS-87-190*, NTIS Identifying Number ADA 183038, University of Southern California, Information Sciences Institute, Marina del Rey, CA, USA, June 1987.

Meijs, K.J.; 2004; Generating natural narrative speech for the Virtual Storyteller; M.Sc. Thesis, March 2004

Mel'cuk, Igor A. (1988); Dependency Syntax: Theory and Practice; State University of New York

Quirk, R.,Greenbaum, S., Leech, G. & Svartvik, J.; 1985; A Comprehensive Grammar of the English Language; Longman Group Limited 1985

Reape, M. & Mellish, C.; 1999; Just What is Aggregation Anyway?; in *Proc. 7th European Workshop on Natural Language Generation*, 1999

Reiter, Ehud & Dale, Robert; 1995; Building Applied Natural Language Generation Systems; in *Natural Language Engineering 1 (1): 000-000*

Reiter, Ehud & Dale, Robert; 2000; Building Natural Language Generation Systems; Cambridge University Press

Rensen, S.H.J.; 2004; De virtuele verhalenverteller: Agent-gebaseerde generatie van interessante plots; Masters thesis at University of Twente

Sanders, Ted J.M., Spooren, Wilbert P.M. & Noordman, Leo G.M.; 1992; Toward a Taxonomy of Coherence Relations; in *Discourse Processes* 15, pages 1-35, 1992

Sanders, Ted J.M. & Noordman, Leo G.M.; 2000; The Role of Coherence Relations and their Linguistic Markers in Text Processing; in *Discourse Processes*, Volume 29(1), pages 37-60; Lawrence Erlbaum Associates, Inc., 2000.

Shaw, J.C.; 2002; Clause Aggregation; an approach to generating Concise Text;
Phd thesis, Columbia University

Shaw, J.C.; 1998; Segregatory Co-ordination and Ellipsis in Text Generation; in *Proceedings of the 36th. Annual Meeting of the Association for Computational Linguistics and the 17th. International Conference on Computational Linguistics*

Skut, W., Krenn, B., Brants, T., Uszkoreit, H.; 1997; An annotation scheme for free word order languages; in *Proceedings of the fifth conference of applied Natural Language Processing*; Washington D.C.

Teich, A. & Bateman, J.A.; 1994; Towards the application of text generation in an integrated publication system; in *INLG'94*

White, M. & Caldwell, T.; 1998; EXEMPLARS: A practical, Extensible Framework for Dynamic Text Generation; in *Proceedings of the Ninth International Workshop on Natural Language Generation*, Niagara-on-the-Lake, Canada, pp. 266-275

Wilkinson, J.; 1995; Aggregation in Natural Language Generation: Another Look; Co-op work term report, Department of Computer Science, University of Waterloo, September.

Wouden, T. van der, H. Hoekstra, M. Moortgat, B. Renmans & I. Schuurman; Syntactic Analysis in the Spoken Dutch Corpus. In *M. González Rodriguez & C. Paz Suárez Araujo, Proceedings of the third International Conference on Language Resources and Evaluation*. 768-773.

Appendix A
Generated sentences

This appendix contains all the sentences that were generated by the Surface Realizer, except for the stories shown in chapter 5. Each input file that contained a Rhetorical Dependency Structure was processed several times in a row, to see how many different Surface Forms could be generated. As a side-effect, the latter Surface Forms often have pronouns instead of names, as, at the second or more processing of the RDG, the characters, objects and places are recognized to have recently been mentioned (i.e., when the sentence was first processed).
For most data, the input consisted of two Dependency Trees connected by a Rhetorical relation. The type of relation is given in the header of each section. In the section 'Nested relations' the input consisted of three Dependency Trees, with differing Rhetorical relations.
The Additive relations were best suited for ellipsis, but because the Surface Realizer ellipts as much as possible, and the Additive relation only has one cue word ('en'), only one or two different Surface Forms could be generated for the average Additive relation (a different referential expression does not count). However, for Cause or Purpose relations, as much as four or five different Surface Forms can be generated, due to the different cue words available for these relations. For a number of sentences, I have annotated the syntactic structure they have received. If an elliptic structure is given, the Surface Form must also have a paratactic structure. Other than the narratives given in Chapter 5, the sentences are mostly in present tense. This is simply because the morphological tag of the verbs is set to present instead of past in these examples. All the test sentences were in present tense at first; I set the narratives in the past tense, because it seemed more fitting (indeed, a fairy tale starts with 'once upon a time…', so it is obviously set in the past).

Additive relations

1. Stripping: Amalia gaat naar het bos en Brutus naar de woestijn.

2. Modifying: Amalia zingt. Zij gaat bovendien naar de woestijn.
Conjunction-Reduction: Amalia zingt en gaat erheen.

3. Stripping: Amalia gaat naar de woestijn en Brutus ook.
Subject-Co-ordination: Amalia en Brutus gaan erheen.
Subject-Co-ordination: Ze gaan erheen.

4. Gapping: Amalia houdt van de prins en de prins van Amalia.

5. Modifying: Amalia danst. Bovendien zingt zij.
Stripping: Amalia danst en ook zingt.

6. Gapping: Amalia schopt Brutus en Brutus Amalia.

7. Stripping: Amalia danst en ook zingt.
Verb-Co-ordinationZij danst en zingt.

8. Gapping: De prins gaat te paard naar het bos en Amalia te voet naar de bergen.

9. Gapping + RNR: De prins gaat te paard en Amalia te voet naar de bergen.

10. Subject-Co-ordination: De prins en Amalia gaan te paard naar de bergen.
Stripping: De prins gaat te paard erheen en zij ook.

11. Right Node Raising: De prins rijdt en Amalia loopt naar de bergen.

12. Conjunction-Reduction: De prins zit op een terrasje en drinkt een biertje.

13. Right Node Raising: Brutus slaat en de prins schopt Amalia.

14. Right Node Raising: Brutus geeft Amalia en de prins de koning een trap.

15. Paratactic: Brutus geeft Amalia een boek en de prins bezorgt haar een bos bloemen.

16. Gapping: Brutus geeft Amalia een boek en de prins haar een bos bloemen.

17. Gapping: De koning geeft Brutus een boek en de prins Amalia een bos bloemen.

18. RNR: Amalia vlucht en de prins kuiert rustig naar de woestijn.

19. Gapping: Amalia gaat te voet naar de woestijn en de prins naar de bergen.

20. Paratactic: Amalia is bang en de prins wordt bang.

21. Gapping + RNR: Amalia gaat morgen en de prins vandaag naar de woestijn.

22. Gapping: Amalia gaat vandaag naar de woestijn en de prins vertrekt naar het bos.

23. RNR: Amalia zag Brutus gisteren en de prins ontmoette de koning in het bos.

24. Paratactic: Amalia wil werken en de prins kan werken.

25. Subject-Co-ordination: Amalia en de prins gaan naar het bos.
Stripping: Zij gaat erheen en hij ook.
Subject-Co-ordination: Ze gaan erheen.

26. Verb-Co-ordination: Amalia gilt en schreeuwt.
Stripping: Zij gilt en ook schreeuwt.

27. LD-Co-ordination: Amalia wandelt naar het bos en naar de bergen.
Stripping: Zij wandelt naar het bos en ook naar de bergen.

28. SVP-Co-ordination: Amalia is boos en bang.
Stripping: Zij is boos en ook bang.

29. Gapping + Conjunction-Reduction: Amalia is boos op de prins en bang voor Brutus.

30. Stripping: Amalia gaat morgen naar het bos en vandaag ook.

MOD-Co-ordintationAmalia gaat morgen en vandaag erheen.

31. OBJ1-Co-ordination: Amalia geeft Brutus een stripalbum en een boek.
Stripping: Zij geeft hem een stripalbum en ook een boek.

32. OBJ1-Coordination: Amalia houdt van de prins en van Brutus.
Stripping: Zij houdt van de prins en ook van Brutus.

33. Conjunction-Reduction: Amalia vindt de prins een watje en haat Brutus.

34. Conjunction-Reduction + Gapping: Amalia vindt de prins een watje en Brutus cool.

Cause relations

1. Modifying: Amalia is bang. Daardoor vlucht zij.
Hypotactic: Doordat Amalia bang is, vlucht zij.
Hypotactic: Omdat Amalia bang is, vlucht zij.

2. Paratactic: Brutus gaat naar de woestijn, want hij ziet Amalia.
Modified: Brutus ziet haar. Hij gaat dus erheen.
Modified: Brutus ziet Amalia. Hij gaat daarom erheen.

3. Paratactic: Amalia gaat naar de woestijn, want zij is bang.
Hypotactic: Amalia is bang. Zij gaat dus erheen.
Hypotactic: Amalia is bang. Zij gaat daarom erheen.

4. Amalia geeft Brutus een lel, want zij vindt Brutus een schurk.
Amalia vindt hem een schurk. Zij geeft Brutus dus een lel.
Amalia vindt Brutus een schurk. Zij geeft Brutus daarom een lel.

5. Amalia pakt het zwaard, want zij wil Brutus doden.
Amalia wil hem doden. Dus pakt zij het.
Amalia wil Brutus doden. Daarom pakt zij het.

6. Doordat Brutus een ongelukkige jeugd gehad heeft, mishandelt hij Amalia.
Omdat Brutus een ongelukkige jeugd gehad heeft, mishandelt hij Amalia.
Brutus heeft een ongelukkige jeugd gehad en mishandelt haar.
Doordat hij een ongelukkige jeugd gehad heeft, mishandelt Brutus Amalia.

7. Doordat Amalia mishandeld wordt, gilt zij.
Omdat Amalia mishandeld wordt, gilt zij.
Amalia wordt mishandeld en zij gilt.

8. Doordat ik de poes aai, spint de poes.
Omdat ik haar aai, spint de poes.
Ik aai de poes en de poes spint.

9. De poes blaast, want van schoot gooi ik de poes.
Van schoot gooi ik haar. Dus blaast de poes.

Van schoot gooi ik de poes. Daarom blaast de poes.

10. Ik pest de poes, want ik verveel me.
Ik verveel me. Dus pest ik haar.
Ik verveel me. Daarom pest ik haar.

11. Ik zet wat muziek op, want het is stil.
Het is stil. Ik zet dus wat muziek op.
Het is stil. Ik zet daarom wat muziek op.

12. Aan Amalia heeft Brutus een hekel, want hij vindt haar een verwend kreng.
Brutus vindt Amalia een verwend kreng. Hij heeft een hekel aan haar dus.
Brutus vindt Amalia een verwend kreng. Hij heeft een hekel aan haar daarom.

13. Mijn trui zit vol kattenharen, doordat de kat verhaart.
Mijn trui zit vol kattenharen, omdat de kat verhaart.
De kat verhaart en vol kattenharen zit mijn trui.

Purpose relations

1. Hypotactic: Brutus loopt, om Amalia gevangen te nemen, naar de woestijn.
Paratactic: Hij loopt erheen, want Brutus wil haar gevangen nemen.
Modified: Hij wil Amalia gevangen nemen. Brutus loopt dus erheen.
Modified: Hij wil haar gevangen nemen. Brutus loopt daarom erheen.

2. Hypotactic: Om de prins te pesten, slaat Brutus Amalia.
Paratactic: Hij slaat haar, want Brutus wil de prins pesten.
Modified: Brutus wil de prins pesten. Dus slaat Brutus Amalia.
Modified: Hij wil de prins pesten. Daarom slaat Brutus haar.

3. Om de poes te spinnen horen, aai ik haar.
Ik aai haar, want ik wil haar spinnen horen.
Ik wil haar spinnen horen. Dus aai ik haar.
Ik wil haar spinnen horen. Daarom aai ik haar.
Om haar te spinnen horen, aai ik haar.

5. De poes gaat op het toetsenbord zitten, want zij wil aandacht krijgen.
Zij wil aandacht krijgen. Zij gaat dus op het toetsenbord zitten.
Zij wil aandacht krijgen. Zij gaat daarom op het toetsenbord zitten.
Zij gaat, om aandacht te krijgen op het toetsenbord zitten.

6. Om te spelen, brengt de poes haar speelgoedmuis.
Zij brengt haar speelgoedmuis, want zij wil spelen.
Zij wil spelen. Dus brengt zij haar speelgoedmuis.
Zij wil spelen. Daarom brengt zij haar speelgoedmuis.

Temporal relations

1. Hypotactic: Zodra Amalia het zwaard ziet, pakt zij het.
Hypotactic: Als Amalia het ziet, pakt zij het.

2. Modified: Brutus mishandelt Amalia. Hij heeft ooit een ongelukkig jeugd gehad.
Modified: Brutus mishandelt haar. Hij heeft vroeger een ongelukkig jeugd gehad.
(input: Temporal-before-gap between 'Brutus mishandelt Amalia' and 'Brutus heeft een ongelukkige jeugd gehad')

3. Modified: Brutus is een gemene schurk. Ooit was hij een lief jongetje.
Modified: Brutus is een gemene schurk. Vroeger was hij een lief jongetje.
(input: Temporal-before-gap between 'Brutus is een gemene schurk' and 'Brutus is een lief jongetje')

4. Hypotactic: Terwijl de prins op een terrasje zit, drinkt hij een biertje.

5. Hypotactic: Als Amalia Brutus ziet, wordt zij bang.

6. Hypotactic: Amalia steekt, nadat zij het zwaard op heeft gepakt, Brutus neer.
Modified: Zij pakt het op. Zij steekt hem vervolgens neer.
Modified: Zij pakt het op. Amalia steekt daarna Brutus neer.
(input: Temporal-after-sequence between 'Amalia pakt het zwaard op' and 'Amalia steekt Brutus neer')

7. Modified: Brutus doodt Amalia. De prins zal ooit gruwelijk wraak nemen.
Modified: Brutus doodt haar. De prins zal later gruwelijk wraak nemen.
(input: Temporal-after-gap between 'Brutus doodt Amalia' and 'De prins neemt gruwelijk wraak')

8. Modified: Brutus gaat plotseling naar de woestijn.

9. Modified: Amalia pakte het zwaard op. Zij stak Brutus toen neer.
Modified: Amalia pakte het op. Zij stak Brutus daarna neer.
Paratactic: Zij stak, nadat zij het zwaard op had gepakt, hem neer.

10. Modified: Er was eens een prinses.
(input: Once-relation with nucleus 'Er was een prinses')

Contrast relations

1. Modified: Amalia is bang. Zij wil toch Brutus doden.
Hypotactic: Amalia wil, hoewel zij bang is, hem doden.
Paratactic: Amalia is bang, maar wil Brutus doden.
Modified: Zij is bang. Amalia wil echter hem doden.

2. Modified: Amalia denkt, dat Brutus een ongelukkige jeugd gehad heeft. Zij geeft Brutus toch een lel.
Hypotactic: Amalia geeft hem, hoewel zij, dat Brutus een ongelukkige jeugd gehad heeft, denkt, een lel.
Paratactic: Amalia denkt, dat hij een ongelukkige jeugd gehad heeft, maar geeft Brutus een lel.
Modified: Zij denkt, dat Brutus een ongelukkige jeugd gehad heeft. Amalia geeft Brutus echter een lel.

3. Amalia ziet het zwaard. Zij kan toch niet bij het komen.
Zij kan, hoewel zij het ziet, niet bij het komen.
Zij ziet het, maar kan niet bij het komen.
Amalia ziet het zwaard. Zij kan echter niet bij het komen.

4. Amalia ziet, maar Brutus pakt het zwaard.
Zij ziet het. Echter pakt hij het.

5. Modified: Amalia haat Brutus. Zij houdt echter van de prins.
Stripping: Amalia haat Brutus, maar houdt van de prins.

6. Amalia houdt van Marco Borsato, maar de prins van Coldplay.
Zij houdt van Marco Borsato. Hij houdt echter van Coldplay.

8. Amalia gaat te voet, maar de prins te paard naar het bos.
Zij gaat te voet erheen. De prins gaat te paard echter erheen.

9. Amalia wordt door Brutus gevangen gehouden, maar de prins zit in de zon op een terrasje.
Amalia wordt door Brutus gevangen gehouden. De prins zit in de zon echter op een terrasje.

10. Brutus gaat niet naar de woestijn, maar Amalia gaat erheen.
Hij gaat niet erheen. Zij gaat echter erheen.

11. Modified: Amalia gaat niet naar het bos. Zij gaat echter naar de woestijn.
Stripping: Zij gaat naar de woestijn, maar niet naar het bos.

12. Stripping: Amalia is bang, maar niet Brutus.
Modified: Niet is hij bang. Echter is zij bang.

Nested relations

1. Input: Two nested Additive relations
Nested Gapping: Amalia gaat naar het bos en Brutus naar de woestijn en Willem-Alexander naar het moeras.

3. Input: One Contrast relation, that is the satellite of a Causal relation
Paratactic + Hypotactic: Brutus haat de prins, want Brutus heeft een kat, maar de prins een hond.
Modified: Brutus heeft een kat. Echter heeft de prins een hond. Dus haat Brutus de prins.

Paratactic: Brutus haat de prins, want Brutus heeft een kat, maar de prins een hond.

4. One Additive relation, that is the satellite of a Causal relation
Stripping + Hypotactic: Doordat Amalia de poes over haar buik aait en over haar rug ook, spint de poes.
MOD-Co-ordination + Paratactic: Doordat Amalia de poes over haar buik en over haar rug aait, spint de poes.

5. Input: One Additive relation, that is the satellite of a Purpose-relation
Stripping + Paratactic: Over haar buik aait Amalia de poes en over haar rug ook, om haar te laten spinnen.
Stripping + Hypotactic: Over haar buik aait Amalia de poes en over haar rug ook, want Amalia wil de poes laten spinnen.
MOD-Co-ordination + Hypotactic: Amalia aait de poes over haar buik en over haar rug, om haar te laten spinnen.

6. Input: Two nested Additive relations
Nested Gapping: Brutus houdt van Linkin Park en Amalia van 3 Doors Down en de prins van Coldplay.

8. Input: One Additive relation that is the satellite of a Contrast relation
Paratactic + Modified: De poes wil niet naar buiten en zij wil niet spelen. Toch mauwt zij.
Modified + Paratactic: Hoewel zij niet naar buiten wil en zij niet spelen wil, mauwt zij.
Stripping + Modified: Zij wil niet naar buiten en ook niet spelen. Toch mauwt zij.

9. Input: One Contrast relation that is the satellite of a Causal relation
Modified + Paratactic: Nog heeft de poes voer. Ik denk, dat zij zeurt, want toch mauwt zij.
Hypotactic + Modified: Hoewel zij voer nog heeft, mauwt zij. Ik denk dus, dat zij zeurt.

10. Two nested Causal relations
Paratactic + Modified: De poes mauwt, want zij wil naar buiten. Ik doe dus de deur open.
Modified: Zij wil naar buiten. Daarom mauwt zij. Ik doe de deur daarom open.

Appendix B
Implementation Details

B.1 Morphology algorithm

```
FUNCTION inflectRegular(root, morph) RETURNS word
BEGIN
        IF (morph.contains(singular))        //root is equal to singular form
                word = root;
        ELSE DO
                IF (lastChar.isVowel())  DO
                        IF (lastChar == 'a' OR 'o' OR 'u') DO                   //auto's, menu's
                                word = root + " 's";
                        END ELSE IF (lastChar == 'i') DO
                                IF NOT (lastButOneChar.isVowel())               //mini's
                                        word = root + " 's";
                                ELSE
                                        word = root + "en";                     //uien, contreien, haaien
                        END ELSE IF (lastChar == 'e') DO
                                IF (lastButOneChar == 'e' OR 'i')
                                        word = root + "ën";                     //zeeën, knieën
                                ELSE
                                        word = root + "s";                      //hertjes, hindoes
                        END;
                END ELSE DO
                        IF (lastButOneChar.isVowel() AND lastButTwoChar.isVowel()) DO
                                IF (lastChar == 's')                            //huizen
                                        root.lastChar = 'z';
                                ELSE IF (lastChar == 'f')                       //staven
                                        root.lastChar = 'v';

                                IF (lastButOneChar == lastButTwoChar)   //graaf - graven
                                        root.delete(lastButOneChar);

                        END ELSE IF (lastButOneChar.isVowel())          //wassen, poppen
                                root = root + lastChar;
                        word = root + "en";
                END;
        END;
END;
```

Figure B.1: inflecting regular nouns

```
FUNCTION inflectRegular(root, morph) RETURNS word
BEGIN
	bool addConsonant = true;
	infinitive = root;
	IF (lastButOneChar.isVowel() AND lastButTwoChar.isVowel()) DO  //huil, maak
		addConsonant = false;
		IF (lastChar == 'f')
			infinitive.lastChar = 'v';                          //hoef - hoeven
		ELSE IF (lastChar == 's')
			infinitive.lastChar = 'z';                          //verhuis - verhuizen

		IF (lastButOneChar == lastButTwoChar)                   //maak - maken
			infinitive = infinitive.delete(lastButOneChar);
	END;
	ELSE IF NOT (lastButOneChar.isVowel())                      //werk - werken
		addConsonant + false;

	IF (addConsonant)
		infintive.append(lastChar);                             //was - wassen
	infinitive.append("en");

	IF (morph.contains(PROGRESSIVE))
		word = infinitive + "d";                                //werkend, lopend
	ELSE IF (morph.contains(PERFECT)) DO
		IF (kofschip(root) AND NOT lastChar == 't')
			root.append('t');                                   //gewerkt
		ELSE IF NOT (lastChar == 'd')
			root.append('d');                                   //gegild
		word = "ge" + root;
	END ELSE IF (morph.contains(PRESENT)) DO
		IF (morph.contains(SINGULAR)) DO
			IF ((morph.contains(SECOND) || (THIRD)) AND NOT (lastChar == 't'))
				word = root + 't';                              //werkt, wast
			ELSE
				word = root;                                    //werk, was
		END ELSE
			word = infinitive;                                  //werken, wassen
	END ELSE IF (morph.contains(PAST)) DO
		IF (kofschip(root))                                     //werkte, vluchtte
			word = root + "te";
		ELSE
			word = root  + "de";                                //gilde

		IF (morph.contains(PLURAL))
			word = word + 'n';                                  //gilden, vluchtten
	END ELSE IF (morph.contains(TEINF)) DO
		word = "te " + infinitive;                              //te gillen, te vluchten
	END ELSE IF (morph.contains(INF))
		word = infinitive;                                      //werken, wassen
END;

FUNCTION kofschip(letter) RETURNS boolean      //determines whether the past tense gets a 't' or a 'd'
BEGIN
	IF (letter == 't' OR 'k' OR 'f' OR 's' OR 'ch' OR 'p')
		RETURN TRUE;
	ELSE
		RETURN FALSE;
END;
```

Figure B.2: inflecting regular verbs

B.2 How to expand the Surface Realizer

This section explains how certain parts of the Surface Realizer can be expanded, such as the number of rhetorical relations and cue words, or the number of grammatical rules.

B.2.1 Adding cue words or primitives

Adding a cue word to an existing CueWordLib is fairly simple. First, you determine which relation the cue word is part of, and if it is governed by any primitives. Each CueWordLib consists of lists with cue words that correspond to boxes in the cue word taxonomy. For instance, the list 'InvoluntaryLast' in CauseLib corresponds to the box that inherits from the causal relation, the 'involuntary' primitive and the 'last' primitive. Once you have established which list a cue word should be put in, add the word with the following line:

```
INVOLUNTARYLASTLIST.add(createNode("daardoor", ADV, MOD));
```

The first argument is the cue word itself, the second its syntactic category and the third its dependency label.
If you want to add a primitive to a CueWordLib, add a constant to the library:

```
public static final String INVOLUNTARY = "invol";
```

Then add lists corresponding to the new boxes in the taxonomy, and fill them.

B.2.2 Adding rhetorical relations

To add a new rhetorical relation, you must create a new CueWordLib. Apart from adding primitives and lists of cue words, as described above, you must implement the abstract function 'public void getOptions(String cat)', which takes as its argument the category of a rhetorical relation, and returns all cue words that are suitable to use in that relationship. The taxonomy will show you how the primitives dictate what cue words are appropriate – construct 'getOptions(String cat)' accordingly.
You may have to overload 'public boolean addToNucleus(RSDepTreeNode cueword)' and 'public boolean addToSatellite(RSDepTreeNode cueword)' as well. These functions are implemented by the abstract 'CueWordLib', but do not produce the right result for all cue words. If any cue word in the new relation gets added to the wrong clause by the original functions, they must be overloaded.
All relations must have a nucleus and a satellite, or only a nucleus. The SurfaceRealizer does not accept multi-nuclear relations. If you want to connect three clauses with the same relation, you must embed the relations in the input. The reason that there are no multi-nuclear relations is that the Conjunctor and Elliptor rely heavily on the expectation of a nucleus and a satellite. They need this distinction while adding the cue word, selecting an appropriate elliptic structure and ordering the clauses. Relations that are, in classic RST, multi-nuclear can still be processed this way, but if things were the other way around, vital information would be missing and syntactic aggregation far more difficult.

B.2.3 Adding grammatical rules

The number of grammatical rules, used in the ordering of a Dependency Tree, is of yet small and should be expanded greatly. The rules that are given now were only meant to enable simple surface forms to be produced, so the result of the syntactic aggregation could be appreciated better. Far more, and better, rules are needed to ensure the generation of a grammatical surface form in all cases (see also section 6).

For example, to add the grammatical rule SMAIN -> SU HD, add the following lines to the SmainLib:

```
String[] sX = {SU, HD};       //X is a number, so we know how many rules there are
rules.add(new Rule(SMAIN, s1));
```

The position of the labels in the array determines the position of their target nodes in the surface form. The first argument of the Rule-constructor is the grammatical category of the parent node, the second is the array with child dependency labels.

In LibraryConstants, a complete list is given of all grammatical categories used in the CGN-corpus. Some of these are marked as comment – they are not implemented yet. To implement a category 'X', create a new class 'XLib' which extends ConstituentLib and implements LibraryConstants, and fill its constructor with rules as shown above.

B.2.4 Referential Expression Generator

The Referential Expression Generator is, at the time of writing, a simple class that does not function very well. It has a CharacterHistory which keeps track of the last named characters, objects and places. If one of these was mentioned in the last few sentences, in a certain grammatical role, and no other character or object of the same gender was mentioned since, and this character, object or place is mentioned again, *fulfilling the same grammatical role* (for instance subject or object1), a new referential expression is selected to apply to it. At the time of writing, this referential expression is always a pronomen.

Unfortunately, the REG sometimes fails to produce a pronomen when it should and sometimes creates it when that leads to confusion. Its algorithm must be adjusted so it will work better.

B.2.5 Other additions

The NounLib and VerbLib only inflect regular nouns and verbs. The irregular nouns and verbs are read from text files, nouns.txt and verbs.txt, in directory data/natlang/rdg/libraries. To add an irregular noun, add its plural form to nouns.txt. To add an irregular verb, add a line with all its inflections to verbs.txt, like this:

```
ben     bent        is      zijn    was     waren   geweest     zijnde      zijn
first   second      third   Plural  Sing    Plural
Present             Past                    Perfect Progressive Infinitive
```

The CharacterHistory needs a list of characters, objects and places that it should track. This list is given in the file characters.txt in the directory data/natlang/rdg/discourse. To add a character, add its name or, if the character is called 'the prince', just add 'prince'. The same goes for objects and places.

B.2.6 Using the package

The package works in a Java environment and can be compiled by jc and run by jr. Compile with 'jc src/natlang/rdg/*.Java' in the Java directory. Run with 'jr natlang.rdg.testclasses.Tester'.

To use the package in another program, write the following:

```
SurfaceRealizer sr = new SurfaceRealizer();
sr.setUrl(someUrl); OR sr.setFile(someFile);
sr.transform();
Iterator it = sr.getResult();
```

The Iterator will be filled with one or more Java Strings.

Appendix C
Sentences used to create the cue word taxonomy

Zinnen uit 'Sprookjes en vertellingen' van Hans Christian Andersen
vertaald door W. van Eeden, onder redactie van A. Schouten. Bussum, Van Holkema en Warendorf, 1975

Uit De rode schoentjes

Er was eens een meisje,	maar	's zomers liep zij altijd op blote voeten	want	ze was arm,
ze was zo fijn en zo lief,	*echter*		*omdat*	
	**toch*		*dus*	
	**hoewel*		*zodat*	
	**ondanks*		*daardoor*	
			doordat	
			daarom	

en	's winters op grote klompen,	zodat	haar wreef helemaal rood werd.
**ook*		*waardoor*	
bovendien		*doordat*	
		omdat	

Karen kreeg nieuwe kleren,	en	nieuwe schoenen moest ze ook hebben.
	bovendien	
	zowel … als	
	ook	

Dat was een	maar	de oude dame kon	en daarom	had ze er weinig aan.
prachtig gezicht,	*echter*	niet meer goed zien	*dus*	
	**toch*		*want*	
	**hoewel*		*omdat*	
	**ondanks*		*zodat*	
			waardoor	
			doordat	

De oude dame gaf de soldaat een stuiver <u>en toen</u> / *daarna* / *voordat* / *nadat* / *vervolgens* ging ze de kerk in.

… tilde haar in het rijtuig; <u>maar</u> / *echter* / *?ondanks dat* / *toch* / *hoewel* de voetjes bleven doordansen, <u>zodat</u> / *want* / *omdat* / *waardoor* / *doordat* / *?dus* / *?daarom* ze de goede oude dame gruwelijk trapte.

Thuis werden de schoentjes in de kast gezet, <u>maar</u> / *hoewel* / *toch* / *echter* / *?ondanks dat* Karin kon het niet laten ernaar te kijken

<u>Maar als</u> / *maar wanneer* / *maar toen* / *maar… dan* ze naar rechts wilde dansten de schoentjes naar links <u>en toen</u> / *als* / *wanneer* / *?zodra* zij vooruit wilde dansten de schoentjes achteruit

…ze dacht dat het de maan was, <u>want</u> / *?dus* / *omdat* / *daarom* / **zodat* / **doordat* het was een gezicht, <u>maar</u> / *echter* het was de oude soldaat

Uit De prinses op de erwt

Er was eens een prins die zo graag een prinses wilde hebben, <u>maar</u> / *echter* / **toch* / **hoewel* het moest een echte prinses zijn.

<u>Toen</u> / *dan* / *?tenslotte* / *?uiteindelijk* / *terwijl* reisde hij de hele wereld rond om er één te vinden, <u>maar</u> / **ondanks dat* / *?hoewel* / *echter* / *?toch* overal kwam er iets tussen.

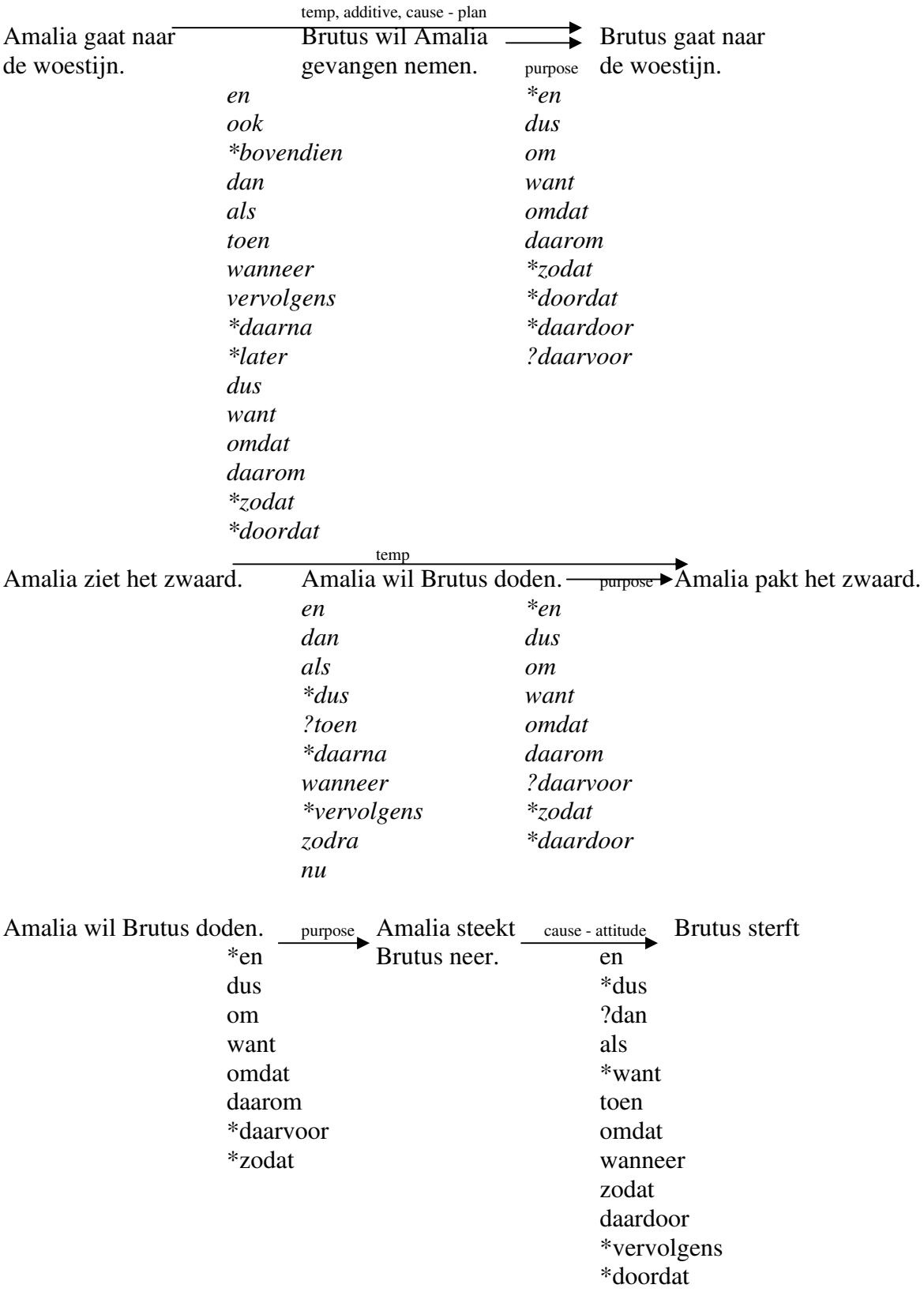
<u>Toen</u> / *wanneer* / *daarna* / *dan* / *later* / *uiteindelijk* kwam hij weer thuis en was erg bedroefd, <u>want</u> / **dus* / **doordat* / *omdat* / **daarom* / ** zodat* hij wou zo graag een heuse prinses hebben.

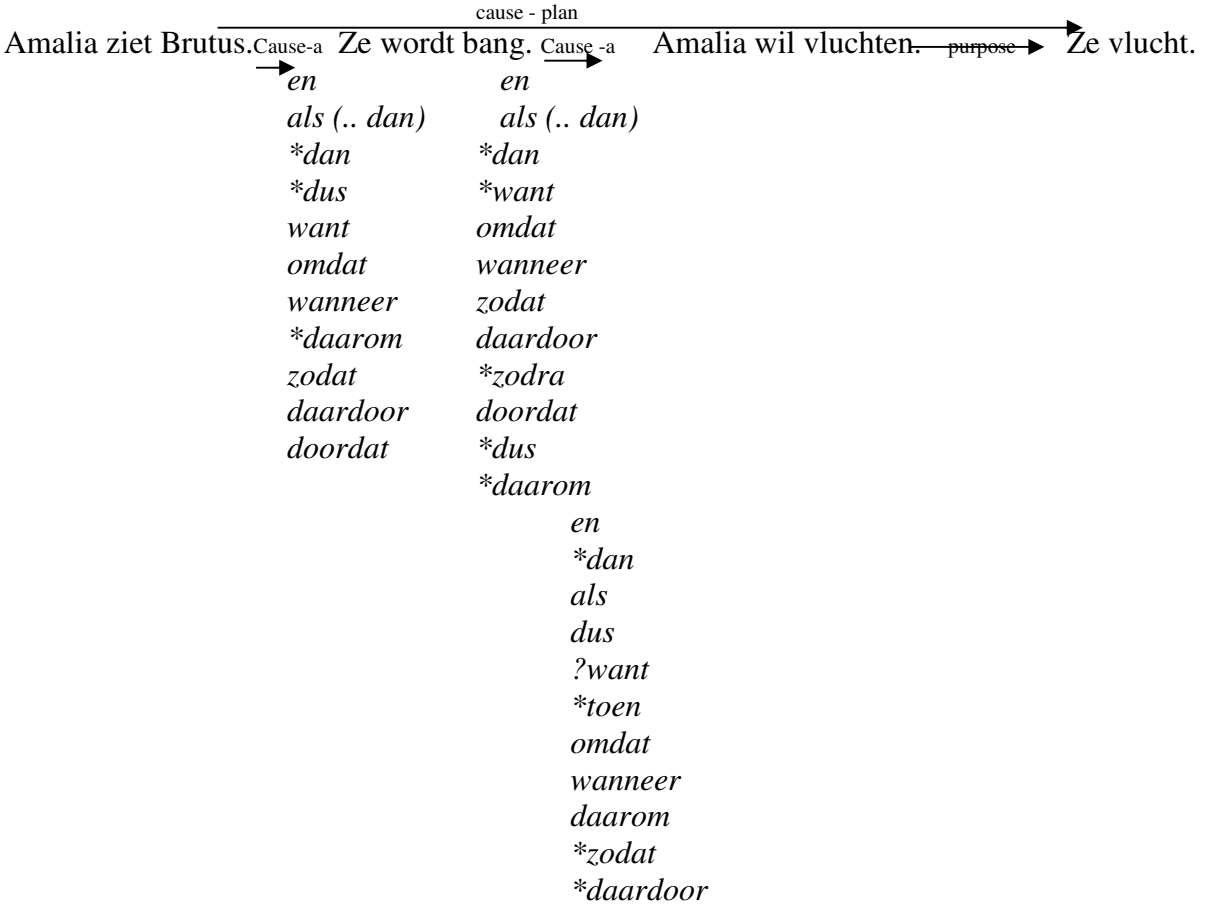
tenslotte **daardoor*
vervolgens

<u>Toen</u>	konden ze zien dat	<u>omdat</u>	ze door de twintig matrassen […]
dan	het een echte prinses was	**zodat*	heen de erwt gevoeld had.
zo		*want*	
daaraan		**daardoor*	
		**doordat*	
		**daarom*	

De prins nam haar toen tot vrouw,	<u>want</u>	nu wist hij dat hij een echte prinses had.
	dus	
	omdat	
	daarom	
	**zodat*	
	**daardoor*	

Voorbeelden gebaseerd op de Virtuele Verhalenverteller




cause - plan
Amalia ziet Brutus.
Cause-a
Ze wordt bang.
Cause -a
Amalia wil vluchten.
purpose
Ze vlucht.
en
als (.. dan)
*dan
*dus
want
omdat
wanneer
*daarom
zodat
daardoor
doordat
en
als (.. dan)
*dan
*want
omdat
wanneer
zodat
daardoor
*zodra
doordat
*dus
*daarom
en
*dan
als
dus
?want
*toen
omdat
wanneer
daarom
*zodat
*daardoor
contrast
Amalia is bang.
Ze wil vluchten.
Contrast
Ze wil Brutus doden.
purpose
Ze steekt hem neer.
maar (ook)
toch
*terwijl
hoewel
echter
ondanks
maar ook
*toch
?terwijl ook
hoewel ook
echter ook
*ondanks
dus
om
want
omdat
daarom
*zodat
*daardoor
purpose
Brutus wil Amalia opsluiten.
purpose
Brutus gaat naar de woestijn.
temp
Brutus gaat naar het kasteel.
dus
om
want
omdat
daarom
?daarvoor
?zodat
en
dan
*als
toen
*wanneer
daarna
nadat
vervolgens
tenslotte
(uit)eindelijk
*zodra
?later

Brutus pakt Amalia op. —attitude - cause→ Brutus krijgt hoop. —attitude–cause→ Brutus wordt agressief.

attitude - cause	attitude–cause
en	*en*
als	*als*
**dus*	**dus*
omdat	*omdat*
wanneer	*wanneer*
**daarom*	**daarom*
zodat	*zodat*
daardoor	*daardoor*
doordat	*doordat*

Brutus wordt agressief. —cause→ Brutus wil Amalia slaan. —purpose→ Brutus slaat Amalia

cause	purpose
en	**en*
**dan*	*dus*
als (..dan)	*want*
omdat	*omdat*
wanneer	**wanneer*
**daarom*	*?daarom*
**zodat*	**zodat*
**daardoor*	**daardoor*
**zodra*	
want	
**dus*	

Brutus schopt Amalia. —Add→ Brutus slaat Amalia —cause - plan→ Amalia schreeuwt.

Add	cause - plan
en (ook)	*want*
zowel .. als	*en*
en .. bovendien	*dan*
	als
	**dus*
	omdat
	?wanneer
	**daarom*
	**zodat*