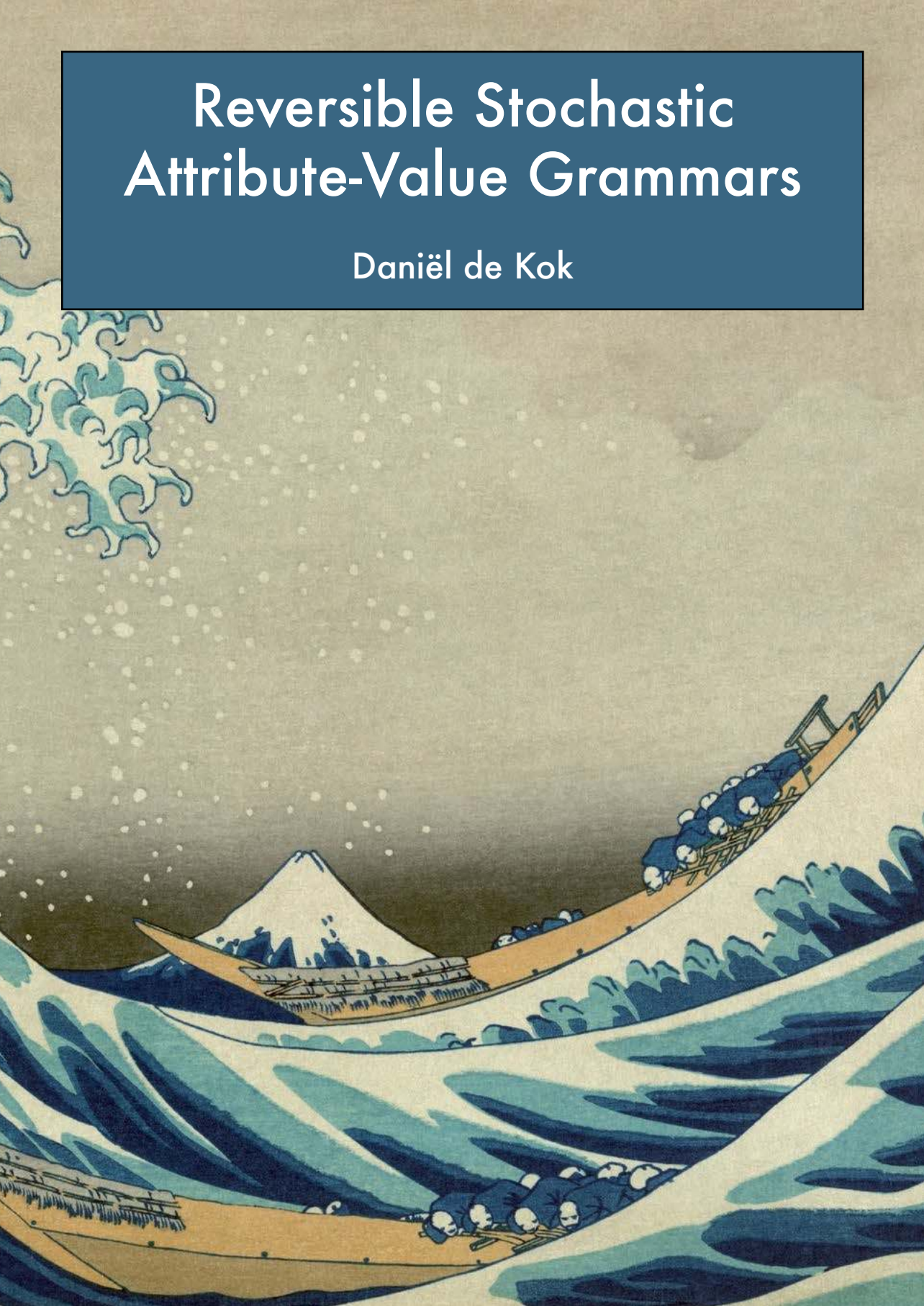
Reversible Stochastic Attribute-Value Grammars

Daniël de Kok



Reversible Stochastic Attribute-Value Grammars

Daniël de Kok

university of
groningen

faculty of arts

CLCG

The work presented here was carried out under the auspices of the Dutch-Flemish NTU/STEVIN project DAISY and the Center for Language and Cognition Groningen of the Faculty of Arts of the University of Groningen.

Groningen Dissertations in Linguistics 113
ISBN: 978-90-367-6112-3
ISBN (digital): 978-90-367-6111-6

Cover image: *The Great Wave off Kanagawa*, Hokusai, 1830-1833
Printed by Wöhrmann Print Service, Zutphen, The Netherlands

RIJKSUNIVERSITEIT GRONINGEN

Reversible Stochastic Attribute-Value Grammars

Proefschrift

ter verkrijging van het doctoraat in de
Letteren
aan de Rijksuniversiteit Groningen
op gezag van de
Rector Magnificus, dr. E. Sterken,
in het openbaar te verdedigen op
donderdag 11 april 2013
om 16:15 uur

door

Daniël Jakob Alex de Kok
geboren op 24 april 1982
te Groningen

Promotores:	Prof. dr. G.J.M. van Noord
	Prof. dr. ir. J. Nerbonne

Beoordelingscommissie:	Prof. dr. S. Clark
	Prof. dr. S. Oepen
	Prof. dr. S. Riezler

ISBN (electronic version): 978-90-367-6111-6
ISBN (print version): 978-90-367-6112-3

Contents

Acknowledgements

First and foremost, I would like to thank my daily supervisor and co-promotor Gertjan van Noord. His 'lifework', Alpino, provided me with the laboratory setting that most computational linguists can only dream of. Besides providing this 'ontleedkundig laboratorium', it was always a joy to discuss and try ideas with him. His enthusiasm and prompt reactions, often resulted in a positive feedback loop, where we refined ideas in a matter of hours. I will also dearly remember Gertjan outside work, such as the hikes in Oregon with him and Barbara.

I also owe many thanks to John Nerbonne, my other promotor. First of all, because he provided the final encouragement to apply for a PhD position. Without him, this thesis would not have been written. His close-reading also improved the quality of my thesis substantially.

Stefan Riezler, Stephan Oepen, and Stephen Clark agreed to be on my reading committee. I am honoured that my thesis was judged by them and would like to thank them a lot for their work and suggestions.

Besides my promotors, there are a few colleagues I would like to mention in particular, because they were so closely involved with my work. I would like to thank Barbara Plank and Yan Zhao for the extensive discussions on maximum entropy modeling — exchanging ideas with you improved my understanding of such models enormously. I would like to thank Jelmer van der Linde, with whom I developed Dact, a tool for analyzing treebanks. Last, but not least, I would like to thank Harm Brouwer for his courage to start writing a book on Haskell and natural language processing with me and the many great coffee/tea-break discussions that we had.

The 'alfa-informatica' group in Groningen was a great group to be part of. Besides being a fertile ground for aspiring and established computational linguists, it is also a very social collective of people. We had many great Sinterklaas evenings, Wii nights, pub outings, and other festivities. Thank you for the four great years, Barbara, Çağrı, Dörte, Dicky, Erik, Geoffrey,

George, Gertjan, Gideon, Gosse, Harm, Henny, Ismail, Jelmer, Jelena, Johan, John, Jörg, Kilian, Kostadin, Leonie, Lucia, Ma, Marjoleine, Martijn, Noortje Nynke, Peter (times three), Proscovia, Simon, Tim, Valerio, and Yan! Extra bonus points go to Martijn Wieling, Yan Zhao, Proscovia Olango, Jianqiang Ma, and Simon Suster for being my office mates for prolonged periods.

I would also like to thank other friends and colleagues for the great soccer matches, dinners, concerts, pub visits, and help, including: Alessandro, Aljar, Audrey, Ferdau, Gisi, Hilbert, Jasper, Jessica, Karin (times two), Laura, Liesbeth, Maarten, Marieke, Radek, Rimke, Rika, Roel, Roelien, Ryan, Trevor, Tristan, Veerle, and Wyke.

I served as a board member and chairman of the Groningen Association for PhD Candidates (Grasp). I would like to thank my fellow board members for the many nice dinners, pub nights, and productive meetings: Barbara, Britta, Ingrid, Jesse, Job, Killian, Lotte, Magda, Na, Thomas, and Veerle.

Of course, a PhD candidate is nothing without his or her paranimfs. And I am very honored that Jasper Spaans en Nynke van der Vliet agreed to be my paranimfs. I have known Jasper since my first first year in the 'Informatiekunde' program and am grateful to have him (and Karin!) as a friend since then. Nynke has been a colleague for nearly four years, during which I very much enjoyed our vehement discussions about philosophy and life.

Like in most academic papers, the people who contributed the most to this work (scientifically or emotionally) are listed first or last.

One of the advantages of marrying is that it doubles your family. Thank you Carsten, Marlen, Frauke, Sascha, and Jana for readily adopting me and for the many good times!

I am honoured to thank my parents, Luit and Gertie, for their help, love, and, of course, for bringing me into this world. I would also like to thank Gideon and Marliesa for their support. I am very thankful for having a brother who is also a good friend.

The last paragraph of my acknowledgements is dedicated to the girl who changed my life forever and whom I had the fortune to marry - Dörte. Thank you for all your love, insightfulness, and positivity! My greatest wish is spending a long and healthy life with you!

Chapter 1

Introduction

Reversible grammars, grammars that can be used in parsing and generation, were introduced as early as 1975. Kay [1975] argues that grammars should not be seen as well-formedness conditions on strings of words, but as a relation between sentences and structures. Kay argues that the theoretical appeal of reversible grammars is even stronger than the practical advantages:

> *The practical advantages of a reversible grammar are obvious. A computer program that engages in any kind of conversation must both parse and generate and it would be economical to base both processes on the same grammar. But, to me, the theoretical appeal is stronger. It is plausible that we have something in our heads that fills the function I am ascribing to grammar, though I am not insensitive to the claims of those who deny this. But it is altogether implausible that we have two such things, one for parsing and one for generation, essentially unrelated to one another.*

Kay [1984] revisits the topic of reversibility. Here, he specifies the requirements for such a reversible grammar more sharply — to allow any order of processing, linguistic descriptions should be defined declaratively rather than imperatively. Since Kay's pioneering work, wide-coverage grammars have been developed that are reversible in theory or in practice.

Since the mid-nineties, statistical methods have changed the field of natural language processing permanently. As a result, hand-written grammars were augmented with statistical models to resolve ambiguity in parsing and to select the most fluent sentence in generation. However, given that reversible grammars were a topic of significant interest in the eighties, it is surprising that, to

our knowledge, no one attempted to extend hand-written reversible grammars with a probabilistic model that could also be used in parsing and generation.

The overarching objective of this work is to show that it is possible to make a system where a parser and a generator not only share one grammar and lexicon, but also one statistical component for parse disambiguation and fluency ranking. We only deem such such an attempt to be successful if (1) such a *reversible* component performs as well as carefully developed parse disambiguation and fluency ranking components; and (2) such a component has a certain amount of integration between the tasks, in that at least a part of the model should be active in both directions.

However, to test this overarching objective, we require a parser and a generator. For Dutch, a wide-coverage grammar and parser are available in the form of Alpino [van Noord, 2006]. In this work, we introduce our generator and fluency ranker for the Alpino grammar. After a detailed evaluation of these new components, we propose *Reversible Stochastic Attribute-Value Grammar* (RSAVG), a formalism that combines reversible grammars with *one* model for parse disambiguation and fluency ranking. We show that RSAVG performs as well as the directional parsing and fluency ranking models, and as such fulfills the first requirement above. Finally, we will use feature selection methods to prove that there is indeed integration between parse disambiguation and fluency ranking, fulfilling the second requirement of a fully reversible system.

Contributions

In this thesis we make the following scientific contributions:

Semantic restriction We describe a new method for top-down guidance in generation. This method unifies lexical items with the relevant portions of the input to guide generation. As a result, unification will fail when filling slots in grammar rules when such unifications would introduce semantics that are not in correspondence with the input. As such, our method is more effective than existing methods such as semantic filtering [Shieber, 1988].

Fluency ranking for Dutch We describe an effective fluency ranker for Dutch, that selects the most fluent sentence from a set of competing sentences. Our error analysis for this ranker shows that many of the remaining challenges are related to weaknesses in the input representation, rather than to the fluency ranker.

Probability estimation in training data We provide an extensive analysis of methods to estimate the probability of sentences, abstract representations, and derivations in the training data. We show that the method that uses the quality estimations of derivations indirectly perform the best. As an additional benefit, this method does not skew the training data in favor of parse disambiguation or fluency ranking in reversible models.

Reversible Stochastic Attribute-Value Grammars We propose a new formalism, that is an extension of Stochastic Attribute-Value grammars. This formalism uses a single model and set of features to perform both parse disambiguation and fluency ranking. We show that this model performs as well as models that are specifically trained for parse disambiguation and fluency ranking. As a result, this formalism uses only one grammar and one model to perform both parsing and generation.

Extensive evaluation of feature selection methods We propose a new feature selection method, *correlation selection*, and modify the gain-informed selection method proposed by Zhou et al. [2003] for tasks with arbitrary feature values. We compare these and three other feature selection methods and show their effectiveness for constructing parse disambiguation and fluency ranking models. We show that the *grafting* method [Perkins et al., 2003], which uses the gradients of features in its selection criterion, outperforms the other methods consistently in both tasks.

Effective features in fluency ranking In the literature, fluency ranking models are often constructed by automatically extracting a huge number of features using templates, effectively treating feature-based models as black boxes. We show that, by applying feature selection, only a small number of features is required to rank sentences effectively by fluency. We also provide an analysis of the most discriminative features and not only show that these features can be understood, but that the fluency of a sentence can be described by a small number of features that model word and part-of-speech trigram distributions, topicalization, modifier adjoining, and ordering in the middle field.

Generalized iterative error mining We propose a new technique for finding errors and shortcomings in wide-coverage grammars and lexicons, in order to improve parsing and generation. This technique combines two earlier techniques to find surface patterns (words and part-of-speech tags) to pinpoint the sources of parsing errors as exactly as possible. We also propose an automatic,

qualitative evaluation method, that can be used to compare error miners in different configurations.

It should be emphasized that nearly all the scientific contributions outlined above are applicable to most natural language parsing and generation systems.

Besides the scientific contributions, work on this thesis resulted in the development of several software components. Here, we give a short description of these components:

Generator and fluency ranker for Dutch The generator and fluency ranker that are described in this thesis, are integrated in the Alpino natural language processing system and can be used with minimal effort.[1]

Maximum entropy estimator with feature selection During this project, we developed a new maximum entropy parameter estimator for ranking tasks, named TinyEst[2] to replace the venerable TADM parameter estimator.[3] It uses TADM's data format, but adds three feature selection methods: selection using an ℓ_1 regularizer, grafting, and grafting-light. Besides the additional features, TinyEst is easier to install, since it is written in C99 without any external dependencies.

Error miner We implemented a fast error miner[4] that supports the error mining techniques of Sagot and de la Clergerie [2006] and the error miner described in Chapter 8. It also provides a graphical user interface to examine features within their context.

Approximate randomization test package We developed a Haskell package[5] for paired and unpaired approximate randomization tests (Section 4.6.4). The package also provides command-line utilities for performing such tests.

Treebank viewer We developed Dact,[6] a user-friendly tool for viewing, querying, and analyzing large Alpino treebanks. Dact is now being used by

[1] http://www.let.rug.nl/vannoord/alp/Alpino/
[2] https://github.com/danieldk/tinyest
[3] http://tadm.sourceforge.net/
[4] https://github.com/rug-compling/errormining
[5] http://hackage.haskell.org/package/approx-rand-test
[6] https://github.com/rug-compling/dact

a growing number of linguists and was used to obtain some of the corpus statistics used in this thesis.

Chapter guide

Chapter 2 provides an introduction to the attribute-value grammar formalism as it is used in the Alpino grammar. It also discusses dependency structure, which is the abstract representation used by Alpino to describe the structure of a sentence.

We introduce our generator for the Alpino grammar in Chapter 3. We first describe abstract dependency structures, which form the input to the generator. Then, we give an overview of the items and inference rules that are used in the generator as well as its theorem prover. We introduce *semantic restriction*, a very effective method for top-down guidance, that makes generation tractable. Finally, we conclude the chapter with an evaluation of the coverage of the generator.

In Chapter 4, we introduce our fluency ranking model. We first discuss various aspects that influence fluency. Then, we describe n-gram language models, which have traditionally been used with great success for fluency ranking. However, there are many syntactic phenomena that cannot be captured by an n-gram model. To integrate such characteristics, we use maximum entropy models. After a general introduction to maximum entropy modeling, we describe the features that we use in our fluency ranking model. We then introduce our experimental setup and use this setup to evaluate our ranker. We conclude the chapter with a detailed analysis of examples where the ranker did not choose the best realization.

Reversible Stochastic Attribute-Value Grammars are introduced in Chapter 5. This formalism combines reversible attribute-value grammar with one component for both parse disambiguation and fluency ranking. Moreover, we show that Reversible Stochastic Attribute-Value Grammars do not perform significantly differently from directional models for parse disambiguation and fluency ranking.

In Chapter 6, we describe four feature selection methods and introduce the correlation selection method. We then compare these five methods, and show that the *grafting* method is the most effective selection method in fluency ranking and parse disambiguation. We then use this method to isolate the most discriminative features in fluency ranking models and provide an analysis of these features.

We apply feature selection in Chapter 7 to find the most discriminative

features in parse disambiguation, fluency ranking, and reversible models. We propose a new methodology to estimate the contributions of classes of features. We then use this method to show that reversible models are truly reversible, and rely on features used in both directions, even more than directional models.

In Chapter 8 we apply feature selection to detect incorrect and incomplete descriptions in attribute-value grammar, so-called *error mining*. Since the data sets and feature space for this task are much larger, we cannot use the feature selection methods that were described in Chapter 6. Instead, we describe two methods from the literature that were specifically developed for error mining and show their shortcomings. We then propose a new method that combines the strengths of the former proposals. We evaluate the three *error miners* using a new quantitative evaluation method.

Chapter 2

Attribute-Value Grammar in Alpino

2.1 Introduction

Attribute-value grammar (AVG) has been studied extensively and is used in various natural language processing systems. In this chapter, we give a short description of the AVG formalism and then discuss the implementation of AVG in Alpino. We refer to Shieber [1986] for a more extensive introduction to attribute-value grammars in various guises.

2.2 Attribute-value grammar

2.2.1 Attribute-value structure

We base the following discussion of attribute-value structures on Carpenter [1992] and Copestake [2000]. An attribute-value structure is a rooted directed acyclic graph, where the edges are labeled using *attributes* and the vertices using *values*. Figure 2.1 shows an example of an attribute value structure that stores information about the music album *Yod*, such as its composer and personnel.

Formally, an attribute-value structure is a tuple $\mathcal{A} = \langle Q, A, V, r, \delta, \alpha \rangle$, where the following hold:

- Q: a finite set of vertices

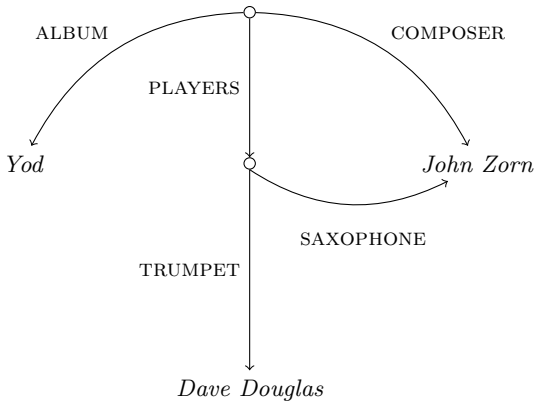


Figure 2.1: Rooted directed acyclic graph containing information about a music album.

- A: a finite set of symbols, corresponding to attributes
- V: a finite set of symbols, corresponding to values
- $r \in Q$: the root vertex
- $\delta : A \times Q \rightarrow Q$: a partial function that maps an attribute and a vertex to another vertex
- $\alpha : F \rightarrow V$: a function that maps a vertex to a value, where $F = \{q \in Q \mid for\ all\ l \in A,\ \delta(q, l)\ is\ undefined\}$

If Path = A$*$ and ϵ is the empty path, we can extend the definition of δ such that for a given path $\pi \in$ Path, $\delta(\pi, r)$ returns the vertex that is obtained by following π from the root vertex r. Formally:

- $\delta(\epsilon, q) = q$
- $\delta(f \cdot \pi, q) = \delta(\pi, \delta(f, q))$, where '.' is concatenation.

Using the function δ, two additional constraints that apply to attribute-value structures can be defined:

- **Reachability**: every vertex must be reachable from the root vertex r. For every $q \in Q$ there should be a path π such that $\delta(\pi, r) = q$.

- **Acyclicity**: no cycles should occur in the graph. For each state $q \in Q$ there should be no non-ϵ path π such that $\delta(\pi, q) = q$.

For an attribute-value structure $\mathcal{A}$, the *path equivalence* function $\equiv_{\mathcal{A}} \subseteq$ Path $\times$ Path and the *path value* function $\mathcal{P}_{\mathcal{A}}$: Path $\rightarrow V$ are defined such that:

- $\pi \equiv_{\mathcal{A}} \pi'$ iff $\delta(\pi, r) = \delta(\pi', r)$
- $\mathcal{P}_{\mathcal{A}}(\pi) = \sigma$ iff $\alpha(\delta(\pi, r)) = \sigma$

When an attribute-value structure $\mathcal{A}'$ contains all the information of $\mathcal{A}$, $\mathcal{A}$ is said to subsume $\mathcal{A}'$ ($\mathcal{A} \sqsubseteq \mathcal{A}'$). Formally, $\mathcal{A} \sqsubseteq \mathcal{A}'$ iff:

- $\forall_{\pi \in Path} \forall_{\pi' \in Path} [\ if\ \pi \equiv_{\mathcal{A}} \pi'\ then\ \pi \equiv_{\mathcal{A}'} \pi']$
- $\forall_{\pi \in Path} [\ if\ \mathcal{P}_{\mathcal{A}}(\pi) = \sigma\ then\ \mathcal{P}_{\mathcal{A}'}(\pi) = \sigma]$

Figure 2.2(a) shows an attribute-value structure that subsumes the structure in Figure 2.1. It is easy to see that this is the case: the paths of the subsuming graph are all present in the subsumed graph, and evaluate to the same value. Figure 2.2(b) shows a graph that does not subsume the structure in Figure 2.1, since there are two violations of the requirements for subsumption. If we name these graphs $\mathcal{A}$ and $\mathcal{A}'$ respectively: (1) $\mathcal{P}_{\mathcal{A}}(\text{ALBUM}) = \textit{Yod}$, while $\mathcal{P}_{\mathcal{A}'}(\text{ALBUM}) = \textit{Alef}$ and (2) PLAYERS$\cdot$SAXOPHONE $\not\equiv_{\mathcal{A}}$ COMPOSER, while PLAYERS $\cdot$ SAXOPHONE $\equiv_{\mathcal{A}'}$ COMPOSER.

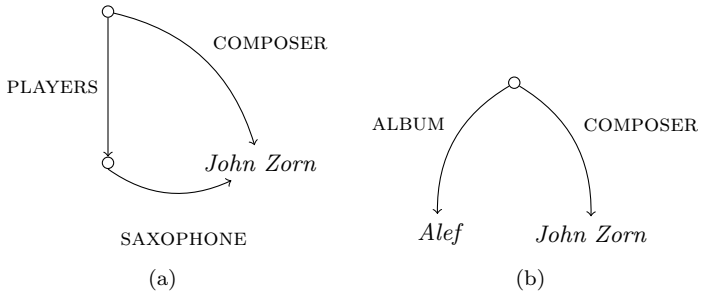


Figure 2.2: Attribute-value structures that (a) subsume and (b) do not subsume the structure in Figure 2.1.

An attribute-value structure $\mathcal{A}_u$ is a unifier of the structures $\mathcal{A}$ and $\mathcal{A}'$ iff: $\mathcal{A} \sqsubseteq \mathcal{A}_u$ and $\mathcal{A}' \sqsubseteq \mathcal{A}_u$.

If $u(\mathcal{A}, \mathcal{A}')$ is the set of all unifiers of $\mathcal{A}$ and $\mathcal{A}'$, then $\mathcal{A}_{mgu}$ is the *most general unifier* of $\mathcal{A}$ and $\mathcal{A}'$ iff:

- $\mathcal{A}_{mgu} \in u(\mathcal{A}, \mathcal{A}')$
- $\forall_{\mathcal{A}_u \in u(\mathcal{A},\mathcal{A}')} \mathcal{A}_{mgu} \sqsubseteq \mathcal{A}_u$

$\mathcal{A}_{mgu}$ is also called the *unification* of $\mathcal{A}$ and $\mathcal{A}'$ ($\mathcal{A}_{mgu} = \mathcal{A} \sqcup \mathcal{A}'$).

The attribute-value structure in Figure 2.4(a) is a unifier of the attribute-value structures in Figure 2.1 and Figure 2.3. However, it is not the most general unifier, since the attribute-value structure in Figure 2.4(b) is also a unifier of these structures and 2.4(a) is subsumed by that structure. The structure in Figure 2.4(b), on the other hand, is the most general unifier, since it subsumes all other unifiers.

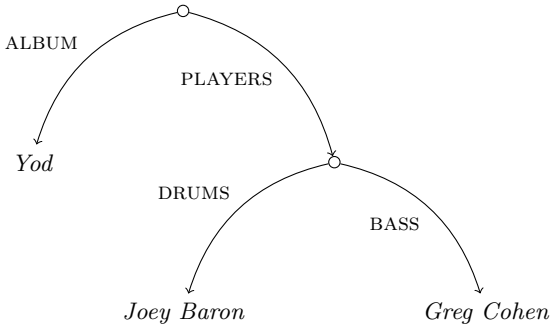
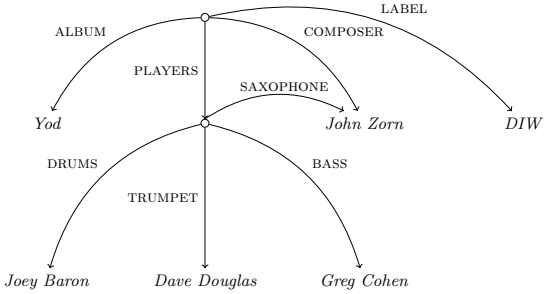


Figure 2.3: Another attribute-value graph that contains information about an album.

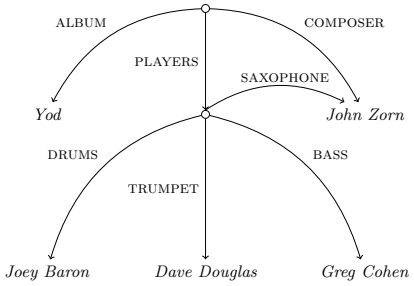
Attribute-value matrices

An alternative, more compact, representation of attribute-value graphs are attribute-value matrices. A matrix represents a vertex using a set of attribute/value pairs. Reentrancies in the graph (multiple paths leading to the same vertex), are expressed using coindexing — the first occurrence of a reentrant value is listed explicitly with an index number, subsequent occurrences only have the index. Figure 2.5 shows the attribute-value matrices corresponding to the graphs in Figure 2.1 and Figure 2.3.

Since the attribute-value matrix representation is much more compact, it will be used in the remainder of this book.



(a)



(b)

Figure 2.4: Attribute-value structures that are (a) a unifier (b) the most general unifier of the structures in Figure 2.1 and Figure 2.3.

2.2.2 Attribute-value grammar

A computational grammar is a mapping between sentences and syntactic structure or semantics. One popular category of computational grammars is phrase structure grammar. Phrase structure grammar uses production rules to define syntactic categories in terms of other categories or morphemes. Production rules have the form $X \rightarrow Y$, where X and Y are sequences of categories or morphemes. Such a rule states that X can be rewritten as Y.

One commonly-used type of phrase structure grammars is context-free grammar (CFG). A context-free grammar is a grammar consisting of rules of the form $V \rightarrow w$ where V is a category label and w any number of category labels and morphemes. Such grammars are context-free because the left-hand side can only consist of one category. The rule $S \rightarrow NP\ VP$ is an example of a CFG rule. This rule states that a sentence (S) can be rewritten to a noun phrase

Figure 2.5: Two attribute attribute-value structures.

(*NP*) followed by a verb phrase (*VP*). Or more colloquially: a sentence consists of a noun phrase followed by a verb phrase.

An attribute-value grammar is a grammar that resembles a context-free grammar; however its categories are not simply labels, but attribute-value structures. We will first define a simplified version of attribute-value grammar, $\mathcal{G}_s$, that follows naturally from the treatment of attribute-value structures in the previous section.

The simplified attribute-value grammar is a tuple $\mathcal{G}_s = \langle N, \Sigma, S, P \rangle$ where the following hold:

- N: the set of syntactic categories, represented as attribute-value structures.
- Σ: the set of morphemes.
- S: the category corresponding to a sentence.
- P: a set of production rules of the form $N \rightarrow (N|\Sigma)*$

$\mathcal{G}_s$ can be used to structure categories more conveniently than a CFG with category labels. For instance, the CFG rule $S \rightarrow NP\ VP$ could be expanded to ensure that there is agreement between the verb in the VP and the NP. For instance, the production rule in Figure 2.6 is a variant of this rule that ensures that the *NP* and *VP* are both in first person plural.

The limitations of $\mathcal{G}_s$ become apparent quickly. Since the categories in productions are isolated, it is not possible to state that (sub-)structures of categories should unify. Taking the rule in Figure 2.6 as an example, a grammar in $\mathcal{G}_s$ would have to enumerate $S \rightarrow VP\ NP$ rules for all possible combinations of the PERSON and NUMBER attributes.

A related problem is that $\mathcal{G}_s$ does not allow for syntactic information to percolate across categories. For instance, in the rule $VP \rightarrow V\ NP$ we would like the *VP* to have the syntactic properties of its head, *V*.

Attribute-value grammar becomes particularly powerful if we allow sub-structures of the attribute-value structures of a production rule to be unified.

$$\begin{bmatrix}\text{CAT} & s\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT} & np & & \\ \text{HEAD} & \begin{bmatrix}\text{AGR} & \begin{bmatrix}\text{PERSON} & 1\\ \text{NUMBER} & pl\end{bmatrix}\\ \text{CASE} & nom & & \end{bmatrix} & & \end{bmatrix} \begin{bmatrix}\text{CAT} & vp & & \\ \text{HEAD} & \begin{bmatrix}\text{AGR} & \begin{bmatrix}\text{PERSON} & 1\\ \text{NUMBER} & pl\end{bmatrix}\end{bmatrix} & & \end{bmatrix}$$

Figure 2.6: A rule for $S \rightarrow NP\ VP$ in $\mathcal{G}_s$.

To that end, we define attribute-value grammar $\mathcal{G} = \langle N, \Sigma, S, P, \gamma \rangle$ where the following hold

- $\langle N, \Sigma, S, P \rangle$ is a $\mathcal{G}_s$.
- $\gamma : P \rightarrow \{\pi_{\mathcal{N}} \equiv_{\mathcal{N}'} \pi'\}$: a partial function that returns a set of path equivalences, where $\pi_{\mathcal{N}} \equiv_{\mathcal{N}'} \pi'$ iff $\delta(\pi, r_{\mathcal{N}}) = \delta(\pi', r_{\mathcal{N}'})$

By exploiting path equivalences, the production rule in Figure 2.6 can be generalized to that in Figure 2.7. Here, the AGR paths of the right-hand side attribute-value structures are unified, guaranteeing agreement between the *NP* and *VP*.

$$\begin{bmatrix}\text{CAT} & s\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT} & np & \\ \text{HEAD} & \begin{bmatrix}\text{AGR} & \boxed{1}\\ \text{CASE} & nom\end{bmatrix} & \end{bmatrix} \begin{bmatrix}\text{CAT} & vp & \\ \text{HEAD} & \begin{bmatrix}\text{AGR} & \boxed{1}\end{bmatrix} & \end{bmatrix}$$

Figure 2.7: A rule for $S \rightarrow NP\ VP$ in $\mathcal{G}$. The category *VP* inherits the syntactic properties of its verb.

$\mathcal{G}$ also allows for percolation of information. For example, consider the rule $VP \rightarrow V\ NP$ in Figure 2.8. Here, the syntactic information in the HEAD attribute of the *V* category is unified with that attribute of the *VP* category. This makes the *VP* inherit syntactic properties of its head.

From a theoretical perspective, $\mathcal{G}$ does not add anything new to the attribute-value structures described in the previous section. Every category in a gram-

$$\begin{bmatrix} \text{CAT} & vp \\ \text{HEAD} & \boxed{1} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} & v \\ \text{HEAD} & \boxed{1} \end{bmatrix} \quad \begin{bmatrix} \text{CAT} & np \\ \text{HEAD} & \begin{bmatrix} \text{CASE} & acc \end{bmatrix} \end{bmatrix}$$

Figure 2.8: A rule for $VP \rightarrow V\ NP$ in $\mathcal{G}$.

mar rule is a normal attribute-value structure. Path equivalences just make attribute-value structures share nodes in their graphs. For instance, the grammar rule in Figure 2.8 corresponds to the attribute-value structure in Figure 2.9. The syntactic categories of the rule correspond to substructures within that structure.

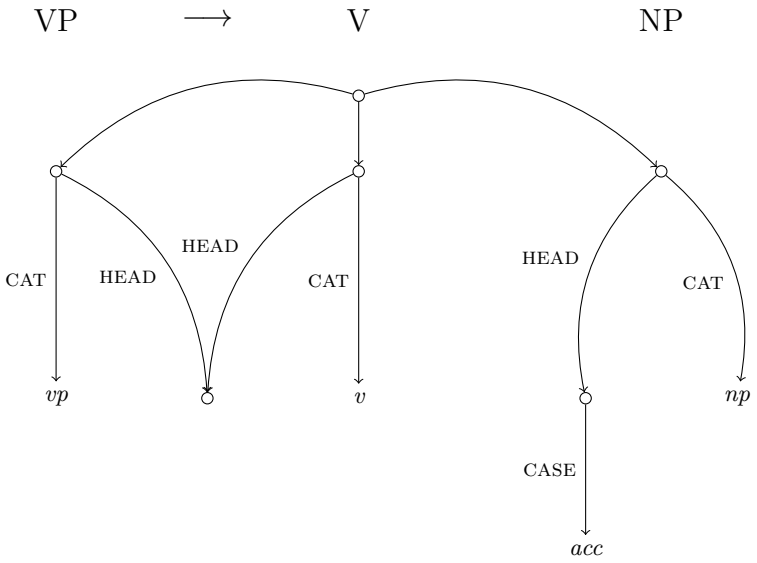


Figure 2.9: Attribute-value structures of the rule in Figure 2.8 as graphs.

The language of the grammar, $L(\mathcal{G})$, is the subset of $\Sigma*$ that can be formed by successively rewriting S, such that no syntactic categories (N) remain. A sequence of morphemes $w \in \Sigma*$ is a valid sentence in $\mathcal{G}$ if $w \in L(\mathcal{G})$. If $x \Rightarrow^*_{\mathcal{G}} y$ means that x can be rewritten to y in a finite number of steps in $\mathcal{G}$, and $\{x \Rightarrow^*_{\mathcal{G}} y\}$ is the set of all possible ways to rewrite x to y, then the attribute-value structures of a sentence w are: $\{S|S \Rightarrow^*_{\mathcal{G}} w\}$. Each attribute-

value structure represents a different syntactic analysis of w.

2.2.3 Derivation

As was briefly mentioned before, rewrite rules of the form $N \rightarrow (N|\Sigma)*$ in attribute-value grammar can be used to rewrite N on the left-hand side as $(N|\Sigma)*$. Such a rewriting step is used in a *top-down derivation*. A top-down derivation attempts to prove that e.g. a given sentence is valid according to a grammar by successively rewriting category S (see definition of $\mathcal{G}$), such that the sentence is obtained.

Alternatively, we can see rewrite rules of the form $N \rightarrow (N|\Sigma)*$ as a statement that if we have a sequence $(N|\Sigma)*$, we can deduce the left-hand side N. Such a deduction step is used in *bottom-up derivation*. A bottom-up derivation attempts to prove S using successive derivations, starting with words. Figure 2.10 shows a bottom-up derivation using the rule in Figure 2.8 and lexical attribute-value structures for *zie* 'see', the singular first-person inflection of *zien*) and the name *Jan*.

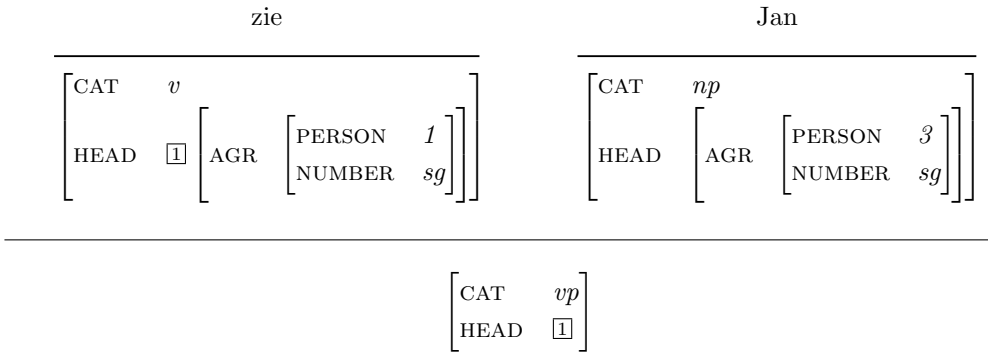


Figure 2.10: Derivation of the category *VP* from the words *zie* and *Jan*.

Other strategies for derivation are possible. For instance, left-corner parsers combine bottom-up and top-down derivation [Matsumoto et al., 1983].

A derivation is a tree-like data structure, where each node contains a deduced category with the list of its antecedents as its daughters. Of course, such a structure is only somewhat tree-like, since there may be reentrancies in the structures of categories.

While a natural language parser or generator creates derivations to prove the sentences or deduce S, they are usually too large to obtain a quick overview

of its general structure. For this reason, it is often useful to obtain a *derivation tree*. A derivation tree is a tree that abbreviates every derivation step by showing the identifiers of the grammar rules that were used in the derivation. For instance, the derivation tree in Figure 2.11 corresponds to the derivation in Figure 2.10.

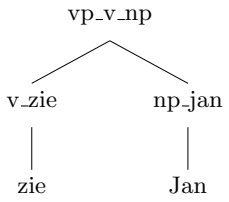


Figure 2.11: A derivation tree corresponding to the derivation in Figure 2.10

Derivation trees have another attractive property: if each grammar rule has an identifier that is unique, the derivation can be reconstructed trivially by recursively applying the grammar rules. This property is exploited in packing (Section 3.4) to reconstruct derivations from a sparse representation of derivation trees.

In this section, we have only discussed the construction of derivations in the abstract. We will return to this subject in Chapter 3, where we discuss a proof procedure that uses bottom-up derivation steps extensively.

2.3 Abstract representations

In the previous section, we gave a glimpse of how derivation can be used to prove that a sentence is valid in the language that a grammar describes. A particular derivation gives a detailed syntactic description of that sentence. However, derivations are usually too specific to the grammar for further processing in a natural language processing pipeline. For instance, it may be useful for other components to know what the subject of a particular verb is, but not how the grammar deduces that certain words constitute the subject. To accommodate processing in a pipeline, a grammar could construct a more abstract description of a sentence or discourse.

Abstract descriptions can be rooted in syntax, semantics, or both. Examples of formalisms rooted in semantics are *flat semantics* [Phillips, 1993, Trujillo, 1995], *Quasi-Logical Form* [Alshawi, 1990], *Minimal Recursion Semantics* [Copestake et al., 2005], and Discourse Representation Structure [Kamp and

Reyle, 1993, Bos, 2008]. Commonly-used formalisms rooted in syntax are *dependency graphs* [Tesnière, 1959, Mel'čuk, 1988] and *LFG f-structure* [Kaplan and Bresnan, 1982].

An abstract representation can be stored in syntactic categories using an attribute, where more complex representations are composed through unification. A simple example is shown in Figure 2.12. This attribute-value structure represents the singular third-person inflection of the verb 'to love'. A predicate describing its semantics is stored in the SEM attribute. The arguments of the 'love' predicate are variable, but reentrant with the SEM values of the attributes representing the subject (SUBJ) and object (OBJ). When a grammar rule unifies the subject or object of the verb, the relevant information becomes available in the SEM attribute of the verb.

$$\begin{bmatrix}
\text{CAT} & v \\
\text{HEAD} & \begin{bmatrix} \text{AGR} & \boxed{1} \begin{bmatrix} \text{PERSON} & 3 \\ \text{NUMBER} & sg \end{bmatrix} \end{bmatrix} \\
\text{SUBJ} & \begin{bmatrix} \text{CAT} & np \\ \text{HEAD} & \begin{bmatrix} \text{AGR} & \boxed{1} \\ \text{CASE} & nom \end{bmatrix} \\ \text{SEM} & \boxed{2} \end{bmatrix} \\
\text{OBJ} & \begin{bmatrix} \text{CAT} & np \\ \text{SEM} & \boxed{3} \end{bmatrix} \\
\text{SEM} & \begin{bmatrix} \text{PRED} & love \\ \text{ARG1} & \boxed{2} \\ \text{ARG2} & \boxed{3} \end{bmatrix}
\end{bmatrix}$$

Figure 2.12: An attribute-value structure for the singular third-person inflection of the verb 'to love'. The SEM attribute contains a semantic representation of the verb.

2.4 AVG in the Alpino system

In this work, we investigate reversible natural language processing systems in the context of the Alpino grammar and lexicon for Dutch [van Noord, 2006].

The following sections give an elementary description of the Alpino system. Although we believe that the contributions in this thesis are applicable to most systems that use an attribute-value grammar, we used the Alpino system to implement and perform experiments with generation and reversibility.

2.4.1 Representation of Attribute-Value Structures

Definite clause grammar

The core of the Alpino system, such as its grammar and lexicon, is written in Prolog. Prolog is a programming language that is rooted in first-order logic. Prolog specifies a program in terms of relations between predicates. It relies extensively on the unification of terms. For example, consider the definition of the *ancestor* relation in Prolog:

```
%% Facts
parent(jack,john).
parent(carl,jack).

%% Rules
ancestor(X,Y) :-
  parent(X,Y).

ancestor(X,Y) :-
  parent(X,Z),
  ancestor(Z,Y).
```

The two rules in this fragment can be read as "X is an ancestor of Y when X is the parent of Y, or if X is the parent of Z and Z is an ancestor of Y". Within a rule, all instances of variables (X, Y, and Z) with the same name are unified. Since unification is commutative and associative, this program is 'reversible' in the sense that the query `ancestor(A,john)` will unify A with every ancestor of *john*, and `ancestor(carl,D)` will unify D with all descendants of *carl*.

In a similar vein, we can construct simple grammars in Prolog. Consider the following Prolog definition of a verb phrase:

```
vp(P0,P2) :- v(P0,P1), np(P1,P2).
vp(P0,P2) :- vp(P0,P1), pp(P1,P2).
```

This fragment can be read as "A *VP* consists of a *V* followed by an *NP*, or a *VP* consists of a *VP* followed by a *PP*". Each syntactic category is represented

by a Prolog term having two arguments that encode positions. Adding positions is necessary to ensure that the constituents are consecutive.

Since adding arguments for positions is a bit clumsy, most Prolog implementations provide the definite clause grammar (DCG) arrow. If the DCG arrow is provided, the following fragment is rewritten by the Prolog interpreter to the fragment (with positions) above:

```
vp --> v, np.
vp --> vp, pp.
```

One interesting aspect of this definition of verb phrases is how easily ambiguity is encoded (under the assumption that the *pp* can also attach to the *np* in the *vp*) - we simply add a rule headed by a term with the same term functor and cardinality.

Of course, in a practical grammar, more constraints ought to be applied to a category. For instance, if we want to have agreement in number of a determiner and a noun in a noun phrase, we could write a rule that uses category terms with an argument:

```
np(Num) --> det(Num), n(Num).
```

Since all instances of the variable `Num` in this rule are unified, it is guaranteed that the number of the determiner agrees with that of the noun. Additionally, this information is unified with the argument of the `np` category, making this information available to rules that have an *np* on the right side of the arrow. This information could, for instance, be used to assure that the main verb of the sentence has the same number as the NP:

```
s --> np(Num), vp(Num).
```

However, when modeling more and more syntactic phenomena, the number of arguments steadily increases. Since the meaning of such arguments is only defined by convention, the grammar writer has to keep track of what arguments are used for what syntactic information. This is both tedious and error-prone.

Attribute-value structures as Prolog terms

Although definite clause grammar may be too inconvenient to write complex grammars, it clearly bears resemblance to attribute-value grammar: a DCG consists of rewrite rules, relies on unification, and can express relations between categories (through the re-occurrence of a variable).

It turns out that an attribute-value structure can also be represented as a Prolog term, by storing values as term arguments. However, one restriction applies. Since the number of arguments of a Prolog term are fixed, a vertex in an attribute-value structure should have a predetermined set of attributes that are allowed on outgoing edges.

Most combinations of attributes will occur frequently. For instance, every attribute-value structure that represents a specific syntactic category will have the same set of attributes. Consequently, it makes sense to make the vertices in an attribute-value graph typed, where the *type* of a vertex specifies what attributes are allowed on outgoing edges. To this end, we give a new definition of attribute-value structures. $\mathcal{A}_{typed}$ is a tuple $\langle Q, A, T, V, r, \theta, \delta, \alpha \rangle$, where the following hold:

- Q: a finite set of vertices
- $r \in Q$: the root vertex
- A: a finite set of symbols, corresponding to attributes
- T: a finite set of types
- $\tau : Q \rightarrow T$: a function that maps a vertex to a type
- $\theta : T \rightarrow \{A\}$: a function that maps a type to a set of attributes
- $\delta : R \times Q \rightarrow Q$: a partial function that maps an attribute and a vertex (q) to another vertex, where $R = \theta(\tau(q))$.
- V: a finite set of symbols, corresponding to values
- $\alpha : F \rightarrow V$: a function that maps a vertex to a value, where $F = \{q \in Q : for\ all\ l \in A,\ \delta(q, l)$ is undefined$\}$

Note that $\mathcal{A}_{typed}$ only provides partial typing - the set of attributes of outgoing edges is limited by the type of a vertex, but an edge can lead to a vertex of any type. We refer to Carpenter [1992] for an extensive treatment of fully-typed attribute-value structures. Alpino uses partially typed attribute-value structures, because type specifications of $\mathcal{A}_{typed}$ can be translated into Prolog fairly easily, as we will see soon. The implementation of full typing is more involved, since it requires type inference, as discussed by [Carpenter, 1992]. In the development of Alpino, the use of partial typing has never posed serious problems.

The attribute-value structures in $\mathcal{A}_{typed}$ can be represented using Prolog terms: a vertex q corresponds to a term, the term functor is $\tau(q)$, the number

of term arguments is $n = |\theta(\tau(q))|$, and if the attributes $\theta(\tau(q))$ are numbered $a_{1..n}$ then the m^{th} argument of the term is $\delta(a_m, q)$.

Consider the structures in Figure 2.13. Each structure is augmented by an atom indicating its type. The type of both structures (or more specifically, their root vertices) is *np*. Both structures also have a substructure with the type *agr*. The first structure can be represented with the term `np(agr(3,sg,de),_)` and the second with `np(agr(3,sg,_),nom)`.

$$np\left[\begin{array}{ll} & agr\left[\begin{array}{ll}\text{PER} & 3\\ \text{NUM} & sg\\ \text{DET} & de\end{array}\right]\end{array}\right] \qquad np\left[\begin{array}{ll}\text{AGR} & agr\left[\begin{array}{ll}\text{PER} & 3\\ \text{NUM} & sg\end{array}\right]\\ \text{CASE} & nom\end{array}\right]$$

Figure 2.13: Simple feature structures of an NP (a) with and (b) without a specific case.

All that is required now, is a mapping from attributes and types to arguments. Such a mapping can be expressed conveniently in Prolog. For instance, for the *np* and *agr* types, the mapping can be defined as follows:

```
%% attr(Attr,Struct,Value)
attr(agr,np(Agr,_),Agr).
attr(case,np(_,Case),Case).
attr(per,agr(Per,_,_),Per).
attr(num,agr(_,Num,_),Num).
attr(det,agr(_,_,Det),Det).
```

Each `attr/3` fact above unifies a given attribute within an attribute-value structure of the type *np* or *agr* with the third argument of `attr/3`. The mapping exploits the fact that the term corresponding to a vertex can be destructured in-place.

Alpino provides a type system to specify $\mathcal{G}_{typed}$ attribute-value structures. A type-compiler constructs (among other things) the aforementioned mappings from types and attributes to arguments.

Alpino also provides the operators in Table 2.1. These operators form a domain-specific language for constructing attribute-value structures. With this language, grammar rules can be defined using Prolog rules. The following definition of the rule $NP \rightarrow Det\ N$ makes use of these operators to construct a structure of the type *np* from structures of the types *det* and *n*.

Operator	**Meaning**
`Path => Type`	Evaluates `Path` and assigns `Type` to the result.
`PathA <=> PathB`	Evaluates `PathA` and `PathB` and unifies the results.

Table 2.1: Operators for defining and modifying attribute-value structures in the Alpino system.

```
grammar_rule(np_det_n,NP,[Det,N]) :-
    NP => np,
    Det => det,
    N => n,
    N:agr <=> NP:agr,
    Det:agr <=> N:agr.
```

The effective structure of the right-hand-side of this rule is shown in Figure 2.14.

$$\texttt{grammar\_rule}\left(\texttt{np\_det\_n},\ {}_{np}\left[\text{AGR}\quad \boxed{1}\right],\ \left\langle {}_{det}\left[\text{AGR}\quad \boxed{1}\right],\ {}_{n}\left[\text{AGR}\quad \boxed{1}\right]\right\rangle\right)$$

Figure 2.14: Structure of the rule $np \rightarrow det\ n$.

Delayed constraints

Most syntactic restrictions in the Alpino system are encoded directly in attribute-value structures of words or grammar rules. However, in some cases this is not possible. For instance, consider the adjunct *vandaag* in the dependent clause

(1) a. omdat Jan stijlvol probeert te winnen
because Jan stylishly tries to win

The modifier *stijlvol* is collected by a category headed by the control verb *probeert*. However, it can be read as a modifier of *probeert* (the attempt is stylish) or *winnen* (the winning is stylish). Since the transfer of modifiers is not deterministic and the number of modifiers is not known beforehand, the 'ownership' of the adjunct cannot be specified directly in the attribute value structure of the head *probeert*.

In such cases, Alpino uses *delayed constraints* [van Noord and Bouma, 1994]. A delayed constraint is a (Prolog) goal that is blocked until a variable becomes instantiated. This mechanism can be used to state things such as: "if the list

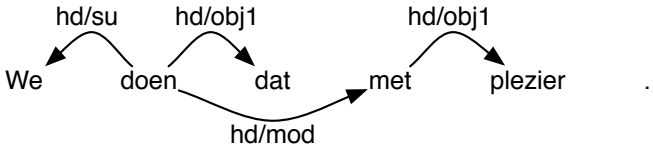


Figure 2.15: Head-dependent relations in the sentence *We doen dat met plezier.* 'We do that with pleasure.' The arcs point towards the dependents.

of modifiers of a syntactic head is final, then the modifiers are divided non-deterministically between the control verb and the embedded verb phrase". Section 2.4.4 discusses the treatment of modifiers in more detail.

2.4.2 Dependency Structure

The Alpino system uses dependency graphs as its abstract representation. A dependency graph is a directed acyclic graph that describes the grammatical relations (arcs) between words (nodes) in a sentence. In such a relation, one word acts as the head, the other as the dependent. The relation indicates the type of head and dependent. Each word has one or more heads and zero or more dependents. Each word is also associated with a part-of-speech tag and its begin and end positions in the parsed (or generated) sentence.

For instance, the Dutch sentence *We doen dat met plezier.* can be described by the nodes and arcs in Figure 2.15.

In Alpino, dependency graphs also specify the categories of constituents. For instance, the phrase *met plezier* in Figure 2.15 would be annotated with the category *pp*. A detailed description of Alpino dependency graphs, including the categories and dependency relations that are used, is provided in van Noord et al. [2011].

Given that Alpino dependency graphs contain words, part-of-speech tags, syntactic categories of constituents, and grammatical dependency relations between heads and dependents, it is a purely syntactic representation.

Previous work shows that dependency graphs are well-fit to use in language processing applications. For instance, the following three studies describe systems that work with Alpino dependency graphs:

- Bouma et al. [2006] used Alpino in a question answering system for Dutch. The Alpino system was used for question analysis and for extracting facts

from huge newspaper corpora.

- Marsi and Krahmer [2005] propose a generalized sentence fusion and generation algorithm that they use to combine related sentences to generate paraphrases. Paraphrases are formed by applying *union fusion* or *intersection fusion*. Union fusion combines all the information that is available, while intersection fusion only retains the information provided by all sentences. Their fusion and generation algorithm uses aligned dependency graphs.
- Atteveldt [2008] uses dependency structures to extract source-quote and object-subject relations from newspaper texts for the automatic extraction of semantic networks.

Since a dependency graph is a directed acyclic graph, it can be stored in attribute-value structure. In fact, Alpino's dependency graphs are constructed as substructures in the attribute-value structures of categories. To this end, Alpino uses attribute-value structures of the type *dt*. Each category has a DT attribute, that has the dependency structure for that category as its value. Attribute-value structures of the type *dt* store information about the head of the dependency structure, such as its stem, inflection, and tag. They also contain attributes for dependents of the head, such as SU, OBJ1, OBJ2, MOD, DET (for respectively storing the subject, direct object, secondary object, modifiers, and determiners). The value of such an attribute is either the dependency structure of the dependent, or a list of dependency structures if there can be multiple dependents of the same type (e.g. modifiers). A special atomic value is used to represent the lack of a dependent of a particular type. For instance, if a verb does not have a direct object, then the value of the attribute OBJ1 will be the atom *nil*.

Consider, for example, the dependency structure in Figure 2.16 for the sentence *De adviezen beklijven* 'The advice persists'. The word *adviezen* ('advice' in plural), represented by the stem *advies* has one dependent, *de* 'the' with the relation *det* (determiner). The word *beklijven* 'to persist' is represented by the stem *beklijf*, and takes the dependency structure associated with *advies* as its subject. The categories of constituents are also available in the dependency structure as the value of the CAT attribute. For instance, the category of *de adviezen* is *np*. In this structure we only list the attributes which have a value different from the special value *nil*.

Dependency structure is formed when a parser or generator unifies the attribute-value structures of the right-hand side of a rule. For instance, when the noun phrase *de adviezen* is found to be the subject of *beklijven*, the dependency structure of *de adviezen* is unified as the value of the SU attribute of the

$$
\begin{array}{l}
\left[\begin{array}{ll}
\text{HD} & \left[\text{LEX} \quad \textit{beklijf}\right] \\
\text{CAT} & \textit{smain} \\
\text{SU} & \left[\begin{array}{ll}
\text{HD} & \left[\text{LEX} \quad \textit{advies}\right] \\
\text{CAT} & \textit{np} \\
\text{DET} & \langle \underset{dt}{\left[\text{HD} \quad \left[\text{LEX} \quad \textit{de}\right]\right]} \rangle
\end{array}\right]
\end{array}\right] \\
\quad dt \qquad\quad dt
\end{array}
$$

Figure 2.16: Dependency structure for *De adviezen beklijven.* The begin and end positions are omitted in this an later examples for brevity.

dependency structure of *beklijven.* This property is specified in the attribute-value structure of *beklijven*, shown in Figure 2.17 — the values of SUBJ · DT and DT · SU are reentrant.

$$
\begin{array}{l}
\left[\begin{array}{ll}
\text{VFORM} & \textit{fin} \\
\text{SUBJ} & \boxed{1} \underset{np}{\left[\begin{array}{ll}
\text{AGR} & \textit{pl} \\
\text{DT} & \boxed{3} \\
\text{CASE} & \textit{nom}
\end{array}\right]} \\
\text{SC} & \langle \boxed{1} \rangle \\
\text{CMODS} & \langle\rangle \\
\text{MODS} & \boxed{2} \\
\text{DT} & \underset{dt}{\left[\begin{array}{ll}
\text{HD} & \left[\begin{array}{ll}
\text{LEX} & \textit{beklijf} \\
\text{SURF} & \textit{beklijven}
\end{array}\right] \\
\text{SU} & \boxed{3} \\
\text{MOD} & \boxed{2}
\end{array}\right]}
\end{array}\right] \\
\quad v
\end{array}
$$

Figure 2.17: Attribute-value structure of the verb *beklijven.* The dependency structure of the subject is reentrant in the dependency structure of *beklijven.*

Since dependency structure is formed as a 'side-effect' of creating a derivation, the topmost category of such a derivation contains the dependency structure that comprises the sentence.

2.4.3 Lexicon

The Alpino grammar is strongly lexicalized, meaning that it relies on detailed syntactic information in the descriptions of a word. If a word is unknown to the lexicon, Alpino attempts to deduce the type of unknown words (and thus their syntactic properties) through a set of heuristics. In this section, the Alpino lexicon is described in more detail.

Representation

In attribute-value grammars, words are also represented by attribute-value structures. In Alpino, the attribute-value structures of a word typically have a large number of attributes to store detailed syntactic information. For instance, the lexical verb type (*v*) has about thirty attributes, which specify, amongst other things, characteristics that are expected of the subject, direct object, and indirect object (when applicable).

Consider the word *verft*, the present-tense second/third person inflection of *verven* 'to paint', which can be transitive and intransitive. One of the possible attribute-value structures of the transitive reading of *verft* is shown in a simplified fashion in Figure 2.18. As we see here, the attribute-value structure already specifies the characteristics of the subject that it can be combined with - the subject should be singular and nominative. It also selects for one complement (direct object) that is in the accusative case. Since the attribute-value structure of a word specifies such combinatory information, grammar rules can be relatively simple.

Another interesting aspect of the attribute-value structure in Figure 2.18 is that it already contains a partially specified structure for its dependency graph. The *su* and *obj1* attributes in the dependency structure of *verft* are unified with the *dt* attributes of the attribute-value of the subject and the direct object. So, once a subject or direct object is unified by a grammar rule, their dependency graphs immediately become a subgraph in the dependency graph of *verft*.

One could construct a dictionary for a system that uses an AVG as a direct mapping between words and attribute-value structures. Effectively, this would make the lexicon a set of grammar rules of the form $N \rightarrow \Sigma$. However, such a lexicon would be unwieldy — if an attribute is added to e.g. the lexical verb type, all entries storing verbs would have to be modified. Also, many properties are shared by e.g. finite verbs, so it would be more convenient to define such a property once to reuse it. In Alpino each word is mapped to one or more compact tags instead. Such tags consist of a simple part-of-speech tag (such as *verb* or *noun*) and additional syntactic information. For instance, the word *verft* is mapped to the tags *verb(hebben,sg3,intransitive)* and

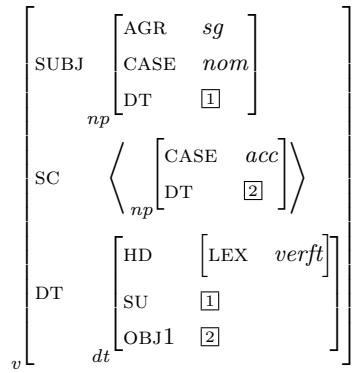


Figure 2.18: Simplified attribute-value structure for *verft*.

verb(hebben,sg3,transitive), the intransitive and transitive readings of *verft*. In turn, tags are mapped to one or more attribute-value structures when used in conjunction with the attribute-value grammar (we call such attribute-value structures *lexical attribute-value structures* from now on).

Static lexicon

The Alpino lexicon contains static many-to-many mappings from words to tag/stem pairs. At the time of writing, the lexicon contained approximately 180,000 individual mappings, as well as 190,000 mappings for named entities. This lexicon is stored as a finite state automaton for compactness and efficiency [Daciuk, 2000].

There are some combinations of words that have a syntactic structure that cannot be derived with grammar rules. For instance, consider the phrase *helemaal niemand* (lit: 'at all nobody', 'nobody at all'), which functions as a pronoun. In this phrase, *helemaal* is an intensifier for the pronoun *niemand*. However, since *helemaal* cannot be applied to most other pronouns, its use with pronouns cannot be generalized to a grammar rule.

Such combinations of words are represented as lexical entries with a more detailed dependency structure. For instance, the entry for 'helemaal niemand' has a dependency structure where *niemand* is the head and *helemaal* a modifier. Since these entries have a non-trivial dependency structure, they are appropriately called *with_dt* structures.

Productive lexicon

Some types of words, such as compounds, named entities, and ordinal numbers, are not good candidates to be enumerated in a static lexicon. For instance, an infinite number of ordinals or compounds can be formed by combining numbers or words. For this reason, the Alpino lexicon has a small definite clause grammar to analyze such words.

Another component applies heuristics to deduce the possible tags and stems of unknown words. For instance, this component can deduce that *Rijksuniversiteit Groningen* 'University of Groningen' is the name of an organization. Particular cues that this is such a named entity are: (1) both words are capitalized and (2) the word *universiteit* is an indicator of an organization name.

These components are together called the *productive lexicon*.

2.4.4 Grammar

Structure

The Alpino grammar consists of approximately 850 construction-specific rules that exploit the detailed syntactic information in lexical attribute-value structures. Rules use general principles that are shared between different rules. For instance, one principle percolates the dependency structure of the projected head on the left-hand side of the rule.

A grammar rule is a Prolog rule that is headed by a term of the following form:

```
grammar_rule(Identifier,LHS,RHS)
```

For instance:

```
grammar_rule(np_det_n,NP,[Det,N])
```

The first term argument is an atom that uniquely identifies the rule, the second argument is the left-hand side of the rule, and the third argument is a list of right-hand side *slots*.

The Prolog rules construct attribute-value structures. Consequently, calling a rule with variable second and third arguments will construct the attribute-value structures of the left-hand side and right-hand side. The parser and generator do not work directly with the hand-written rules, since this representation does not allow for first argument indexing (Section 3.2.5) and reconstructing the attribute-value structure upon each use of a rule would be inefficient. Instead, a compiled version of the grammar is used, that stores rules as Prolog

facts that have the constructed attribute-value structures in their second and third arguments. The compilation procedure is straightforward: each rule is called to construct the left-hand and right-hand sides; these are then stored together with the rule identifier.

Derivations

The derivation of categories using attribute-value grammar rules was discussed in Section 2.2.3. In this section, we give some examples of derivations that were made using the Alpino grammar. Since the complete derivations that Alpino produces are too large to be printed here on paper, we will instead give an example of a derivation tree of a sentence and attribute-value structures of some categories.

Figure 2.19 shows the derivation tree of (the most likely analysis of) the sentence *We doen dat met plezier.* 'We do that with pleasure.' The interior nodes (except those dominating leaf nodes) of the derivation tree show the rules that were used in the derivation. For instance, one particular subtree was constructed using the grammar rule *vp_arg_v(np)* by filling its right hand slots with categories constructed using the *np_pron_weak* and *vp_mod_v* rules. The leaf nodes of the tree are words, the immediately dominating nodes are the tags of words in that particular analysis.

This derivation tree has two types of epsilon leaf nodes. The first, *optpunct(e)* is used to fill right-hand side slots with a special *optpunct* category. This category is used when a constituent can contain non-obligatory punctuation. The *optpunct(e)* item fills such slots when there is no punctuation in that position. The other epsilon node, *vgap*, plays a role in verb-second word order handling. In Dutch, the position of the finite verb is different in main clauses (second position) and subordinate clauses (final position). The usual analysis is that in main clauses, the finite verb has been moved from the final position to the second position [Koster, 1975]. As proposed in Shieber et al. [1990], the Alpino grammar creates an empty verb (the *vgap*) to fill the 'original' position of the finite verb. This empty verb is created when the finite verb is analyzed and contains the relevant constraints that are imposed by the verb.

Figure 2.21 shows the category that was derived in the last derivation step (if bottom-up derivation is used), using the application of the *top_start* rule.[1]

Since (nearly) all rules have path equivalences between the left-hand and right-hand sides of a rule, information can percolate through the derivation. This means that when the *np_n* item dominating *plezier* in Figure 2.19 was unified with the third daughter of *pp_p_arg(np)*, the relevant information is also

[1] Attributes having the value *nil* or the empty list as their value are removed for brevity.

part of the attribute-value structure of *met*. This can be seen in Figure 2.20, in the list that is the value of SC. Also, the dependency structure of the noun phrase is the value of the POBJ1 attribute of the dependency structure of the preposition. To summarize: every unification of a right-hand side category not only carries over information to the left-hand side, but also to the head of the rule and potentially other categories.

In Figure 2.20 we also see a feature of Alpino's type system that has not been discussed before. The type system provides a special mechanism to specify conjunctions, disjunctions and negations of atomic values. For instance, in Alpino, *NP*-internal agreement is specified to be the cross-product of {*sg, pl*} (number), {*de, het*} (gender), and {*def,indef,measdef*} (definiteness). A specification such as *de&def;pl&het&def* says that agreement should either be *de* (feminine/masculine) and *def* (definite), or *pl* (plural) and *het* (neuter) and *def*. The cross-product specification is compiled to a term, so that conjunction and disjunction can be 'checked' via unification [Mellish, 1988].

Modifiers

Within the grammar some dependents, such as modifiers, require special treatment. This is necessary, because modifiers can be used in a phrasal category where the syntactic head of the phrase is not the head of the modifiers in dependency structure. For example, consider the dependent clause

(2) a. omdat hij met plezier een taart heeft gebakken
because he with pleasure a cake has baked

The (simplified) derivation tree of this phrase is shown in Figure 2.22. The rule identifiers are replaced by syntactic categories for clarity.

The prepositional phrase *met plezier* is a modifier of *gebakken*. However, *gebakken* is combined with the auxiliary verb *heeft* to form a new category, *vproj*, that is headed by the auxiliary verb. Consequently, the modifier *met plezier* is 'collected' by the syntactic head *heeft*. As we will discuss in more detail later, *heeft* hands over the collected modifiers to *gebakken*.

One of the consequences of the collection of modifiers by the syntactic head is that the modifiers that are eventually collected by a head are not necessarily modifiers of that head in the dependency structure. In the example above, *heeft* will have *met plezier* in its list of collected modifiers. However, its modifier list in the dependency structure will be empty, since *heeft* will give the modifier to *gebakken*.

There is another issue that makes the treatment of modifiers intricate. Since unification cannot mutate values, it is not possible to simply expand an existing

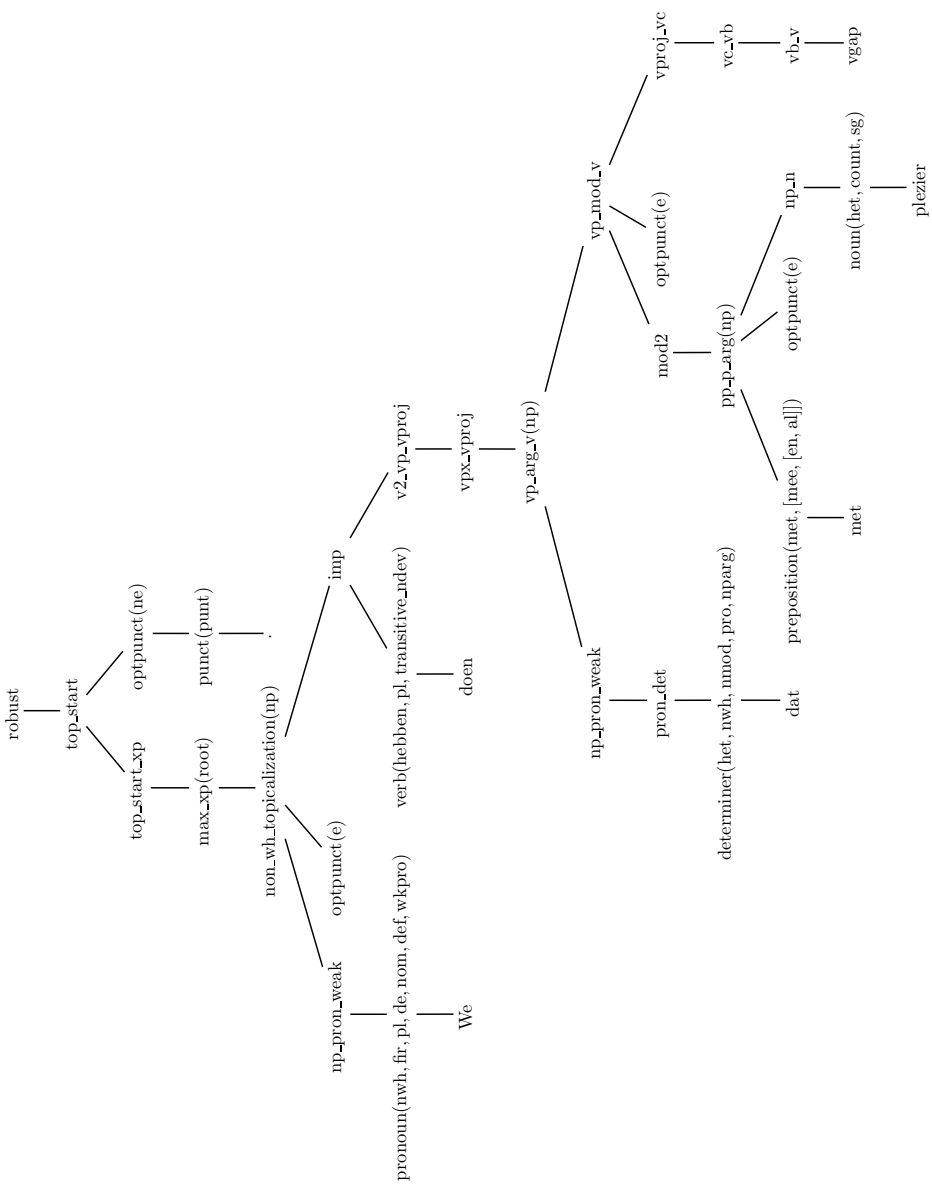


Figure 2.19: Derivation tree for the sentence *We doen dat met plezier.*

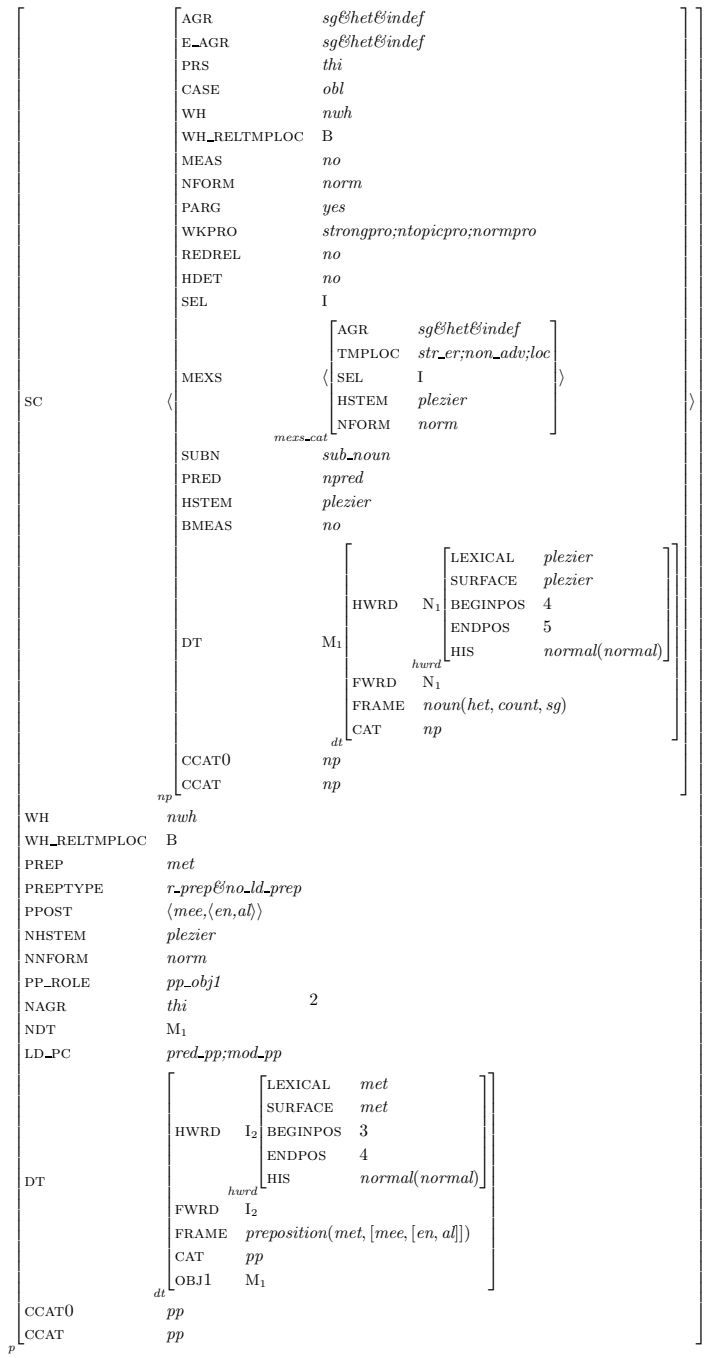


Figure 2.20: Attribute-value structure of *met* in the derivation in Figure 2.19.

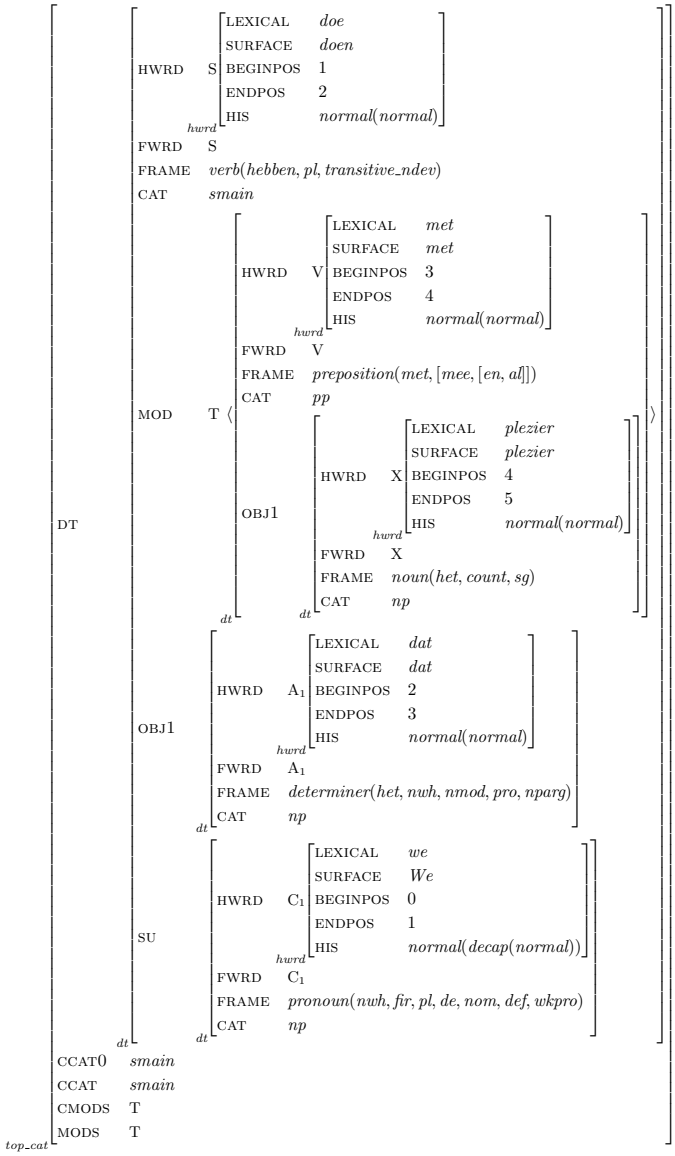


Figure 2.21: Attribute-value structure of the *top_start* node in the derivation in Figure 2.19.

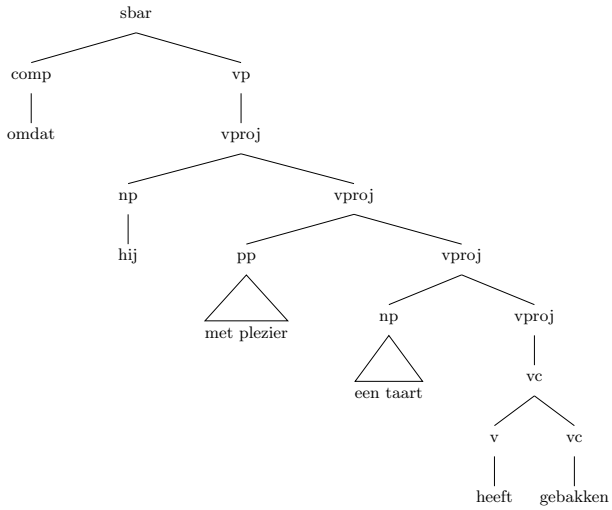


Figure 2.22: Derivation tree of *omdat hij met plezier een taart heeft gebakken* 'because he with pleasure a cake has baked'. Category types are used as node labels.

modifier collection list. Of course, a grammar rule that uses a modifier could create a new collection list, having the modifier as its head and the previously collected modifiers as its tail. For instance, consider the phrase

(3) a. De mooie snelle groene auto
the beautiful fast green car.

The derivation tree for this phrase is shown in Figure 2.23 (again, category types are used as node labels). Following that approach to modifier collection, the lists of modifiers of a given noun (n) category consists of the modifiers collected in the derivation of that category: *1:n* has an empty modifier collection list, *2:n* has a singleton list with *groene*, *3:n* has a list with *snelle* and *groene*, etc. Unfortunately, without extra work, the eventual list of collected modifiers is only available in *4:n* and *np*. This poses a problem for lexically-driven systems such as Alpino, since it makes it impossible to define relations on modifiers in lexical attribute-value structures [van Noord and Bouma, 1994].

We will first discuss how Alpino makes the list of modifiers visible in lexical attribute-value structures. We will then look at an example where the list of modifiers is used in a lexical attribute-value structure.

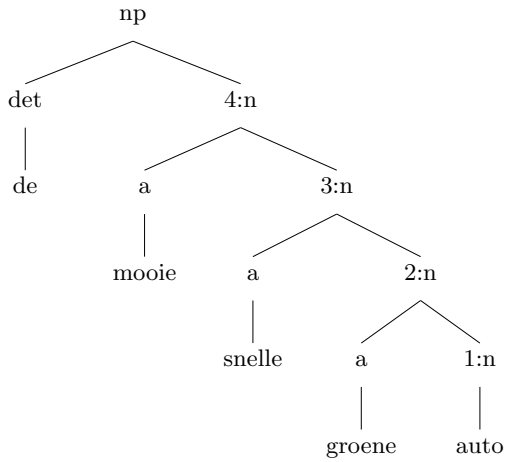


Figure 2.23: Derivation tree for the phrase *de mooie snelle groene auto* 'the beautiful fast green car'. Rule identifiers are replaced by category types.

There are two methods that can be used to make modifiers visible to lexical attribute-value structures:

1. Prolog lists are a recursive data structure: a list is a tuple of a value and a list; or the empty list. A list is expandable if the list in the last tuple is variable, since that variable can be unified with another list. This can be used for modifier collection — as long as the tail of the collection list is variable, modifiers can be added to the list. At the maximal projection, the variable is unified with the empty list, to finalize the modifier list.

2. Rather than using one attribute that collects modifiers, two attributes are used. The value of the first attribute is a list with modifiers that have been collected up to that point in the derivation. The value of the second attribute is the list of all modifiers that have been collected by the head and is reentrant in every derived category with the same head. This value is variable until a derivation is constructed that corresponds to a maximal projection. In that derivation, the second attribute is unified with the first attribute to make the final list of modifiers available to the head.

Alpino follows the second approach, since it works naturally with delayed constraints. On each head, a delayed constraint is put on the list of collected

modifiers, which is variable until maximal projection is reached. In Alpino, categories that can collect modifiers have two attributes, CMOD and MOD. CMOD is used to collect modifiers and MOD is the list of all modifiers that were collected at the maximal projection. Figure 2.24 gives an impression of how these two attributes work for the derivation in Figure 2.23.[2]

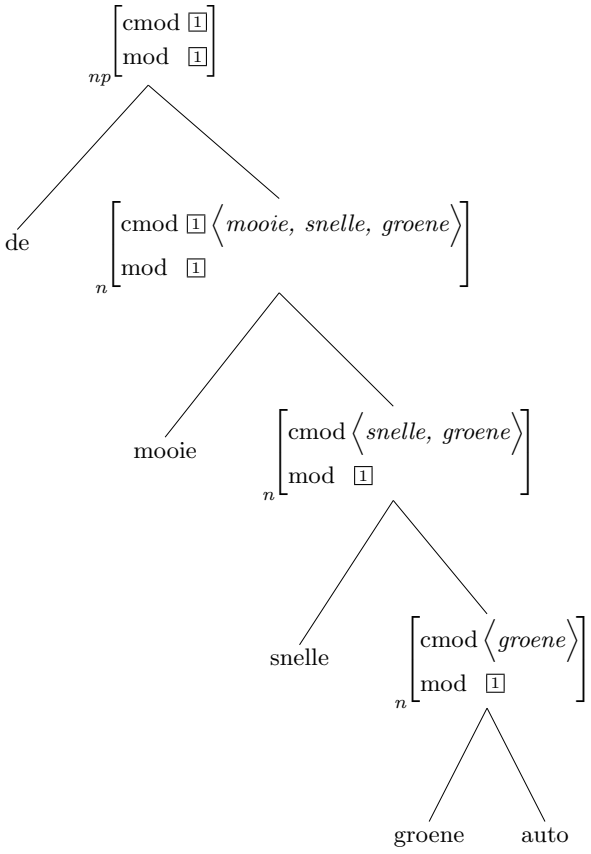


Figure 2.24: Collection of modifiers in the noun phrase *de mooie snelle groene auto* 'the beautiful fast green car'. Modifiers are collected in the CMOD attribute.

[2]In reality, the lists contain the dependency structures of the modifiers.

As we can see in this example, the CMOD attribute always contains the modifiers that were collected up to that point in the derivation. Since the MOD attribute is unified with CMOD when no additional modifiers can be collected, the final list of modifiers is available to each grammar rule in the derivation, including the lexical attribute-value structure for *auto*. In this case, that lexical attribute-value structure would only exploit this mechanism to define the list of modifiers in the dependency structure — it is unified with the DT · MOD attribute.

The earlier example (Figure 2.22), however, uses the MOD attribute more sophisticatedly. Remember that in this case the syntactic head collected the modifiers for their actual head in the dependency structure. The lexical attribute-value structure for auxiliary verbs (such as *heb*) specifies that the modifiers that the auxiliary verb collects are handed over to the verb cluster (*vc*) of the auxiliary verb. This is achieved by adding a delayed constraint on the value of the MOD attribute of *heb*. When MOD is not variable anymore, the lists of modifiers collected by *heb* and those collected by *gebakken* are concatenated and unified with the MOD attribute in of *gebakken*.

The use of delayed constraints and two modifier lists in Alpino provides an effective mechanism for handling modifiers. However, it requires special attention in generation, as we will see in Section 3.3.3

2.5 Conclusion

This chapter provided an overview of the attribute-value grammar formalism, and its implementation in the Alpino system. The next chapter describes how derivations are constructed in generation using the grammar and lexicon that are described in this chapter.

Chapter 3

Chart Generation

The previous chapter provided an introduction to attribute-value grammar and how AVG is implemented in the Alpino system. Such a computational grammar can be used to derive whether a sentence is syntactically correct, and if so, to obtain an abstract representation of that sentence. Vice versa, a grammar can be used to derive sentences that realize a given abstract representation. These two tasks are called *parsing* and *generation* respectively.

In less abstract terms, the role of a parser in a language processing pipeline is to provide syntactic or semantic analyses that can be used by other components to extract and use the information encoded in a text. A generator, on the other hand, is used to produce natural language text that conveys the information that should be presented to a human. Generators are used in many different applications, such as paraphrasing, data summarization, and machine translation.

Reiter and Dale [1997] describes six sub-tasks that a complete natural language generation component should be able to perform:

- **Content determination** decides what information should be communicated in the text. Reiter and Dale [1997] envisions that this task creates messages that contain this information.
- **Discourse planning** settles the structure and order of the messages that should be communicated.
- **Sentence aggregation** groups messages together in sentences.
- **Lexicalization** decides on the words and phrases that are used to express the concepts and relations in a message.

- **Referring expression generation** selects words or phrases to identify entities.
- **Linguistic realization** applies a grammar to make a sentence or text that is syntactically, morphologically, and orthographically correct.

In this work, we focus on the last sub-task, *linguistic realization*. Our motivation to focus on realization, as opposed to any of the other subtasks, is twofold: (1) linguistic realization is the most fundamental task in natural language generation; and (2) no component to perform this task was available for a wide-coverage grammar for Dutch.

We believe that linguistic realization is the most fundamental of these six sub-tasks, because no natural language generation application can function without a rudimentary sentence realizer. The other sub-tasks of a generation component merely output structured information, while realization produces actual sentences. At the same time, there are applications in generation, that can be performed without any of the other sub-tasks, such as: sentence paraphrasing, sentence compression, and machine translation.

In this chapter, we will introduce a sentence realizer for the Alpino grammar, which was described in the previous chapter. This realizer uses the same grammar and lexicon as that used in the Alpino parser, as well as an abstract representation based on dependency structure (Section 2.4.2). This puts the Alpino system firmly in the tradition of reversible unification grammars [Kay, 1984].

The use of reversible unification grammars for generation has been criticized in the past. This criticism is twofold. First, it has been argued that it is difficult to use a grammar that has been written for parsing initially efficiently and correctly for generation [Russell et al., 1990]. Cited concerns include non-termination and the requirement to purge the search space too aggressively. Second, there are concerns that the abstract representation of such systems are too linguistic to be useful in generation [Busemann, 1996]. We will address the concerns that are embedded in the first criticism in Section 3.5, where we show that generation using a reversible unification grammar can be performant using, among other things, our approach to goal-driven generation (Section 3.3.2). We discuss the criticism of our abstract representation in Section 3.1, where we introduce this representation. However, we would like to emphasise that our contributions in this work are applicable to nearly all attribute-value grammars, regardless of their reversibility or their input representation.

In this chapter, we will first introduce the abstract representation that is the input to the generator in Section 3.1. In Section 3.2 we describe generation as a deduction procedure and provide a theorem prover to deduce sentences from

an abstract dependency structure. The remainder of that section discusses some practical issues, such as how the generator finds all possible morphemes and corresponding attribute-value structures given the input, and elementary optimizations to the theorem prover.

In its most basic form, the generator creates all derivations that are possible given the lexical attribute-value structures that are part of the input to the generator. Unfortunately, this makes generation intractable for anything but trivial inputs. However, since we are only interested in derivations that realize (a part of) the input, the generator can be made much more goal-driven. The chart generators described by Shieber [1988], Kay [1996], and Carroll and Oepen [2005] improve performance substantially through top-down guidance provided by the input. In Section 3.3, we propose another method for top-down guidance that is efficient and practical.

3.1 Abstract dependency structures

Section 2.3 discussed the use of dependency structure as the abstract representation of sentences in Alpino. These dependency structures are, however, not directly usable as the input of generation, since they specify the word order and contain inflected words. The dependency structures that are the input to generation differ from those that are the output of parsing in the following aspects:

- They do not contain information about word or constituent adjacency. In parsing, words are annotated by their sentence position to uniquely identify the word in the sentence. In generation, this information would not be used, since it is the task of the generator and fluency ranker to find realizations of the dependency structure that are grammatical and fluent.

- Words are represented by their stem and lexical information. In contrast to parsing, the inflected word is omitted, since it will be constructed by the generator. For instance, if the input specifies that the verb with the stem *loop* 'walk' should be used as third person, singular and present tense, the generator will construct the inflection *loopt*.

  If more than one inflection can be constructed using the lexical information that is available, the generator will do so. For example, if the input specifies that the verb with the stem *schuil* 'shelter' should be used as third person, singular, and past tense, the generator will construct the inflections *school* and *schuilde*.

The lexical information for a word can also be underspecified. For instance, if the tense of a verb is removed, the generator will attempt to realize sentences in the present and past tense.

- Separable particles are not specified as dependents of a verb. Some Dutch verbs have particles that can be connected to or separated from that verb. For example, the particle *weg* in the verb *weggooien* is separable. Alpino dependency structures specify the particle in the lexical information of the verb. If the particle is separated from the verb, it is also a separate dependent of the verb, so that every word that occurred in a sentence has a relation in the dependency structure.

  The choice of whether a particle should be separate or connected to the verb is usually a matter of fluency. Therefore, such choices should be made by the fluency ranking component (Chapter 4), rather than putting it on the plate of the component that creates the input of the generator.

  Since separable particles are not specified in the input of generation, the generator is allowed to realize variants that have the particle connected to a verb as well as those where the particle is separated. Consequently, the question *Zou ik dat mogen weggooien?* 'May I throw that away?' can also be realized as *Zou ik dat weg mogen gooien?*.

- Punctuation is not specified in the dependency structure. For historical reasons, in parsing, nodes representing punctuation are attached to the top category of the dependency structures. As such, they provide no hints to the generator on how the punctuation should be used and whether the use of punctuation is optional.

  Since the specification of punctuation in dependency structures would not provide any useful information to the generator, we leave it to the generator to introduce punctuation.

We call the resulting structures *abstract dependency structures*.

Abstract dependency structures are a linguistically rich input, since they specify the grammatical relations between lexical attribute-value structures in the input. Nonetheless, we believe that the use of abstract dependency structures is justified for different reasons:

- We believe that abstract dependency structure provides enough abstraction for generation tasks such as paraphrasing and machine translation. For instance, Marsi and Krahmer [2005] describes a system that uses Alpino dependency structures to merge information of multiple sentences

and to generate a new 'fused' sentence. Another example is the Paco-MT translation system, which generates target language sentences from Alpino dependency structures using statistical generation [Vandeghinste, 2009].

These tasks require the system to choose a word order that is considered fluent and do not fundamentally alter grammatical relations.

- In other applications, it may be useful to have less linguistically-rich input. However, as described in the generator architecture that was proposed by Reiter and Dale [1997] and discussed in the previous section, sentence realization is only a sub-task in a complete generation system. In such an architecture, the realizer is provided with input that is already linguistically rich, since lexicalization and referring expression generation is already performed by other sub-tasks within the system.

- Dependency structures are the output of the Alpino parser. If the generator used a different representation as its input, applications that use the parser and generator, such as paraphrasing and machine translation, would be required to translate between representations.

- As discussed in Section 2.4.2, dependency structures are constructed as a side-effect of the application of grammar rules. However, the grammar itself is a unification-based phrase structure grammar. Consequently, there is no trivial mapping from abstract dependency structure to a phrase structure tree. In other words, our task is not significantly simpler than if we used an attribute-value grammar with another abstract representation.

- No other wide-coverage unification grammars are available for Dutch and modifying the Alpino grammar to use another representation would be an enormous task.

3.2 Algorithm

3.2.1 Generation as deduction

Parsing and generation can be viewed as deductive processes that respectively prove that a sentence can be analyzed with abstract structure and that a particular sentence is a realization of an abstract structure [Pereira and Warren,

1983, Shieber, 1988, Shieber et al., 1995]. Shieber et al. [1995] describe a system for generating parsers in this vein, where specific parsers are encoded by specifying their axioms, inference rules, and goals.

We will give a description of our bottom-up chart generator [de Kok and van Noord, 2010], which was strongly influenced by Shieber [1988] and Kay [1996], using the notation of Shieber et al. [1995].

Notation In this section, we use α, β, and γ for sequences of zero or more syntactic categories. Capital letters are used for one syntactic category. $A \rightarrow \alpha$ is a grammar rule that rewrites the category A to a sequence of zero or more categories α. Inference rules are written in the following form:

$$\frac{A_1 \cdots A_n}{B} \tag{3.1}$$

This rule states that the *consequent* B can be inferred from the *antecedents* $A_1 \cdots A_n$.

Items Theorems and partially proven theorems are represented as items. In our generator, an item is a grammar rule where zero or more slots are consumed. A dot represents what slots have been consumed - slots left of the dot are consumed, slots right of the dot are not. Items have the following form:

$$[A \rightarrow \alpha \bullet \beta] \tag{3.2}$$

Following Pereira and Warren [1983], we call items that have the dot in a non-final position *active* and items that have the dot in the final position *passive*. Active items have the following form:

$$[A \rightarrow \alpha \bullet B\beta] \tag{3.3}$$

Passive items have the following form:

$$[A \rightarrow \gamma\bullet] \tag{3.4}$$

Grammar rules, with the exception of ϵ-rules (such as the optional punctuation discussed in Section 2.4.4), are active items that have the dot in the initial position:

$$[A \rightarrow \bullet\gamma] \tag{3.5}$$

Epsilon rules are passive items of the following form:

$$[A \rightarrow \bullet] \tag{3.6}$$

Axioms In our generator, axioms are attribute-value structures that represent words and grammar rules. The lexical attribute-value structures are constructed from the dependency structure that is the input of generation (Section 2.4.3) in all possible ways. The axioms corresponding to lexical items have the following form:

$$[A \rightarrow w\bullet] \tag{3.7}$$

The axioms corresponding to grammar rules have the following form:

$$[A \rightarrow \bullet\gamma] \tag{3.8}$$

Goal The goal of generation is to find at least one passive item, where the right-hand side syntactic category is an attribute-value structure of the type *top_cat*. *top_cat* is the type used in Alpino for attribute-value structures that represent a sentence. Additionally, the dependency structure in this attribute-value structure should be subsumed by the dependency structure $\mathcal{D}$ that is the input to generation. The goal of generation is:

$$[S \rightarrow {}_{top\_cat}\left[\begin{array}{ll}\mathrm{DT} & \mathcal{D}'\end{array}\right] \bullet], D \sqsubseteq D'$$

Inference rule The generator uses one inference rule, *completion*:

$$\frac{[A \rightarrow \alpha \bullet B\beta] \quad [B \rightarrow \gamma\bullet]}{[\phi(A \rightarrow \alpha B \bullet \beta)]} \quad \phi = mgu(B, B') \tag{3.9}$$

where $mgu(B, B')$ is the most general unifier of B and B' (Section 2.2.1). This inference rule moves the dot in an active item by unifying the slot right of the dot with a passive item.

When a new item is created using the inference rule, its construction history is recorded.

Example Suppose that we have the following items:

$$\begin{array}{l}[NP \rightarrow \bullet Det\ N] \\ [Det \rightarrow de\bullet] \\ [N \rightarrow man\bullet]\end{array} \tag{3.10}$$

The first (active) item is an axiom corresponding to a grammar rule, the second and third (passive) items are lexical items for the words *de* 'the' and *man* 'man'. Using these axioms NP can be proven:

$$\frac{[NP \rightarrow \bullet Det\ N] \quad [Det' \rightarrow de\bullet]}{[\sigma(NP \rightarrow Det \bullet N)]} \quad \sigma = mgu(Det, Det') \tag{3.11}$$

$$\frac{[NP \rightarrow Det \bullet N] \quad [N' \rightarrow man\bullet]}{[\sigma(NP \rightarrow Det\ N\bullet)]} \quad \sigma = mgu(N, N') \tag{3.12}$$

In this state, an NP with certain characteristics was proven.

3.2.2 Theorem prover

The theorem prover attempts to prove all possible items (theorems) using the initial axioms. The state of the prover is stored in two data structures:

1. **Chart:** the chart stores items that were obtained through the application of inference rules.

2. **Agenda:** the agenda is a queue of new items, which the prover uses to make new inferences.

The flow of items through the prover is simple: new items are first added to the agenda. Once an item is processed, it is removed from the agenda and added to the chart. During the processing of an item on the agenda, the prover attempts to create new items in all possible ways using that item, items on the chart, and the inference rule. Once the agenda is empty, the chart contains all possible inferences that could be made, given the axioms.

The algorithm of the theorem prover is outlined in Algorithm 1. In chart generation, the agenda is initialized by putting all axioms (grammar rules, epsilon rules, and lexical items) on the agenda. In practice, the grammar rules are put on the chart immediately, since there are initially no passive items on the chart that they could interact with.

The `chart_contains` function has a special role. If an item is already on the chart, it is not useful to add it again, since it will interact exactly in the same manner as the existing item. However, the item might have been derived differently and thus represent a different phrase. A sophisticated version of `chart_contains` will modify the existing chart item and add a 'derivation history'. This is discussed in more detail in Section 3.4.

Algorithm 1 The theorem prover

$agenda \leftarrow axiom\_items$
$chart \leftarrow initial\_chart$
while $\neg is\_empty(agenda)$ **do**
  $item \leftarrow head(agenda)$
  if $\neg chart\_contains(item)$ **then**
    $append(agenda, inferences(item))$
    put_on_chart(item)
  end if
  remove_head(agenda)
end while
return $chart\_retrieve\_unifies(\ _{top\_cat}\left[\text{DT} \quad \mathcal{D}\right]\ )$

3.2.3 The complexity of chart generation

Since the theorem prover tries all possible interactions between items, and continues until no more interactions are possible, it is guaranteed to find all items that can be deduced with the axioms and inference rules.

Since words can be ordered arbitrarily in some cases, the time complexity of chart generation is $O(n!)$, where n is the number of passive items on the initial agenda. The worst-case scenario can be observed when a word has multiple modifiers. If a word has n modifiers, $n!$ orderings are often allowed by the grammar. For instance, the following phrases can be generated from the dependency structure in Figure 3.1:

(1) a. de chique warme gele trui
  b. de warme chique gele trui
  c. de chique gele warme trui
  d. de gele chique warme trui
  e. de gele warme chique trui
  f. de warme gele chique trui

Another construct where $n!$ orderings are sometimes problematic are conjunctions. For example, the conjunction *noten, rozijnen, en gember* 'nuts, raisins, and ginger' can be realized as:

(2) a. gember , noten en rozijnen

  b. noten , gember en rozijnen

$$
\begin{bmatrix}
\text{HD} & \left[\text{LEX} \quad \textit{trui}\right] & \\
\text{DET} & \left\langle {}_{dt}\left[\text{HD} \quad \left[\text{LEX} \quad \textit{de}\right]\right]\right\rangle & \\
\text{MOD} & \left\langle {}_{dt}\left[\text{HD} \quad \left[\text{LEX} \quad \textit{chic}\right]\right], {}_{dt}\left[\text{HD} \quad \left[\text{LEX} \quad \textit{geel}\right]\right], {}_{dt}\left[\text{HD} \quad \left[\text{LEX} \quad \textit{warm}\right]\right]\right\rangle &
\end{bmatrix}_{dt}
$$

Figure 3.1: Attribute-value structure for *de warme chique gele trui* 'the warm chic yellow pullover'.

c. noten , rozijnen en gember

d. rozijnen , noten en gember

e. gember , rozijnen en noten

f. rozijnen , gember en noten

The liberty that the grammar and input specification provide in ordering modifiers and conjuncts, is the source of this complexity. Any generation algorithm will suffer from such permissiveness. That said, in practice sentences contain plenty of material that is subject to stricter ordering constraints. So, normally, there is not a factorial number of realizations. Chart generation also eliminates a lot of potential complexity by constructing partial derivations only once and its memory use can be reduced easily through packing (Section 3.4).

3.2.4 Lexical items

As described in Section 3.1 the input to generation in Alpino does not contain word inflections, and may contain only partial lexical information. However, during generation, the initial agenda is populated with passive items having attribute-value structures with detailed subcategorization information.

A (passive) lexical item is constructed from lexical information in the dependency structure in the following manner: first the given stem is looked up in the dictionary. This gives a set of possible inflections and tags. Each tag that is found, is verified to correspond to the lexical information that is provided in the dependency structure. This gives a set of stem-tag-inflection pairings that

are in correspondence with the lexical information in the dependency structure. Then, the tags are mapped to attribute-value structures (Section 2.4.3).

Finally, the dependency structures of these attribute-value structures are unified with the corresponding part of the abstract dependency structure. This is done to provide top-down guidance (Section 3.3) and to weed out lexical items that have a configuration of dependents that does not correspond to the abstract dependency structure.

The resulting attribute-value structures form the passive items for that particular lexical node. If a stem-tag pair has multiple inflections, the inflections are stored as a list in the items to reduce the number of items.

Example Suppose that an abstract dependency structure states that the generator should realize a singular verb with the stem *praat* 'talk'. The abstract dependency structure also lists a prepositional complement (PC) with the preposition *over* 'about'. The first step is to retrieve the stem-tag-inflection tuples. Since the Alpino lexicon is extensive there many such tuples. We list a very small subset in Table 3.1.

The plural readings of *praat* (row three and four) will be eliminated, since the lexical information in the abstract dependency structure does not correspond to that in the tag — the tag indicates that the inflection is plural, while the abstract dependency structure requires a singular inflection.

Inflection	**Tag**
praat	verb(hebben,sg,pc_pp(over))
praat	verb(hebben,sg,intransitive)
praten	verb(hebben,inf,pc_pp(over))
praten	verb(hebben,inf,intransitive)
⋮	⋮

Table 3.1: Four possible inflections and tags for the stem *praat* 'talk'.

Consequently, the attribute-value structures of the first and second reading will be created. The dependency structure of the first reading is shown in 3.2(a) and the second in 3.2(b). The first structure indicates that this reading expects a prepositional complement, since its PC attribute has a value that is not nil. The second reading expects no prepositional complement, since the PC attribute has the value *nil*.

Since the abstract dependency structure that is the input to generation specifies that the verb *praat* has a dependency with the relation PC, it will not

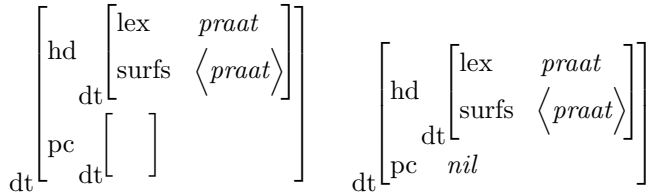


Figure 3.2: Dependency structures of (a) *verb(hebben,sg,pc_pp(over))* and (b) *verb(hebben,sg,intransitive)*.

unify with the dependency structure of the reading with the tag *verb(hebben,sg, intransitive)*, and this entry will also be excluded.

Lexical items with a complex dependency structure

As discussed in Section 2.4.3, the lexicon contains entries consisting of multiple words that have a syntactic structure (*with_dt* entries). Since the roots of such words occur separately in the abstract dependency structure, we cannot simply find the relevant entry by looking up the the roots of those words. For instance, the phrase *helemaal niemand* in the sentence *Helemaal niemand ziet ons* 'Nobody at all sees us' has the dependency structure shown in Figure 3.3.

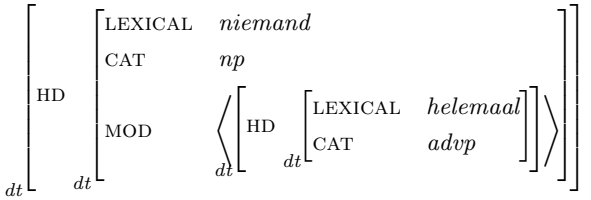


Figure 3.3: Dependency structure of the phrase *helemaal niemand* 'nobody at all'.

We follow a simple method to find such instances. First, we construct a special stem for each such lexical entry. This stem is the lexicographically sorted list of stems of words within that entry. For instance, in the case of *helemaal niemand*, the stem is 'helemaal niemand'. Then, when constructing lexical items for generation, we apply the following procedure: for each sub-structure in an abstract dependency structure, we collect the lists of stems within that structure. The stems are then sorted lexicographically to form a compound

stem. This stem is then looked up in the dictionary. If the stem occurs in a *with_dt* entry, we introduce the corresponding entry if its dependency structure unifies with the sub-structure that is under consideration.

3.2.5 Indexing of items

A grammar rule in the Alpino system consists of three elements (Section 2.4.4): (1) an atom that identifies the rule, such as `np_det_n`; (2) the attribute-value structure that represents the left-hand side of the rule; (3) the list of attribute-value structures that represents the right-hand side of the rule. Finding items for completion can be sped up tremendously if the active items on the chart are indexed by the next slot that should be completed.

Since items are Prolog facts in Alpino, we can rely on *first argument indexing* in Prolog to index items. Prolog interpreters calculate a hash of the first argument of a term that is asserted. When a goal is executed, the first argument of the goal is also hashed, and the interpreter will only consider facts with a matching hash. First argument indexing is only used when the first argument is not variable. If the argument is a term, only the functor of the term is hashed.

To be able to use first-argument indexing, we have to ensure that the next slot to be completed is the first argument in the term that represents an item. Grammar rules are pre-compiled to have this property. When constructing new active items using completion, the next (uncompleted) slot is made the first term argument. Since Alpino has a rich type hierarchy, and types are represented as term functors (Section 2.4.1), such indexing is very effective.

3.2.6 Head-first slot filling

Shieber et al. [1990] define the notion of a *chain rule*, which is a rule that has a right-hand side slot with semantics that is identical to that of the left-hand side. This slot, which we call the *semantic head* of the rule, is expected to carry the most detailed syntactic information. Consequently, if we require that the semantic head is completed before other right-hand side slots, a smaller number of active items will be created.

Top-down guidance (Section 3.3) adds further selection restrictions to the dependency structure of heads.

In the Alpino grammar, the semantic head of a rule is defined to be the first right-hand side slot that has a dependency structure that is identical to that of the left-hand slot. Consider the `np_det_n` rule in the Alpino grammar. The head of the grammar rule clause is as follows:

```
grammar_rule(np_det_n,NP,[Det,N])
```

By checking the right-hand side slots, we will find that $N \cdot \text{DT}$ is identical to $NP \cdot \text{DT}$. Using this information, we can construct a different representation of this rule that has the head as its first argument, to flag the slot that should be completed first and to make use of first-argument indexing:

```
headed_grammar_rule(N,np_det_n,NP,[Det,N]).
```

If a rule has two slots that can be considered the semantic head, we complete the leftmost head first. If a rule does not have a semantic head, the leftmost slot is considered to be the semantic head.

3.3 Top-down Guidance

When chart generation is finished, we are only interested in those derivations where the dependency structure of the root of the derivation tree is subsumed by the dependency structure that was the input to generation.[1] However, the dependency structure that forms the input is not only the goal of generation, but it also provides top-down information that can guide generation.

3.3.1 Semantic filtering

Shieber [1988] uses a filter that reduces the search space in a chart generator. This filter retains items that have semantics that subsume some portion of the goal semantics. Shieber [1988] argues that this can be seen analogous to indexing on string positions in parsing, since it avoids the creation of derivations that are in conflict with the input. It should be clear that this filter can also be applied if we use another abstract representation, such as (in our case) dependency structure. For example, consider the dependency structure in Figure 3.4, corresponding to the sentence *De wethouder berispt de burgermeester.*

At some point during theorem proving, the generator may have constructed a noun phrase corresponding to *de wethouder* and attempt to combine it with an item corresponding to *berispt* to form a verb phrase item with the dependency structure in Figure 3.5. However, the semantic filter would discard this item, since its dependency structure does not subsume (a part of) the dependency structure in Figure 3.4.

However, as Shieber [1988] points out, the use of a semantic filter can lead to incompleteness, unless the grammar is *semantically monotonic*. A grammar is semantically monotonic if "for every phrase admitted by the grammar, the

[1] Requiring the dependency structure of the root node to be identical to the input of generation, would rule out realizations of an underspecified dependency structure.

$$\begin{bmatrix} \text{HD} & \begin{bmatrix}\text{LEX} & \textit{berisp}\end{bmatrix} \\ \text{CAT} & \textit{smain} \\ \text{SU} & \begin{bmatrix} \text{HD} & \begin{bmatrix}\text{LEX} & \textit{wethouder}\end{bmatrix} \\ \text{CAT} & \textit{np} \\ \text{DET} & \langle \begin{bmatrix}\text{HD} & \begin{bmatrix}\text{LEX} & \textit{de}\end{bmatrix}\end{bmatrix} \rangle \end{bmatrix} \\ \text{OBJ1} & \begin{bmatrix} \text{HD} & \begin{bmatrix}\text{LEX} & \textit{burgermeester}\end{bmatrix} \\ \text{CAT} & \textit{np} \\ \text{DET} & \langle \begin{bmatrix}\text{HD} & \begin{bmatrix}\text{LEX} & \textit{de}\end{bmatrix}\end{bmatrix} \rangle \end{bmatrix} \end{bmatrix}$$

Figure 3.4: Attribute-value structure for *De wethouder berispt de burgermeester.*

semantic structure of each immediate subphrase subsumes some portion of the semantic structure of the entire phrase".[2]

Figure 3.6 gives an example of a unary AVG rule ($S \rightarrow A$) that is not semantically monotonic. This is because a portion of the semantics on the right-hand side of the rule, *sem:y*, does not subsume a portion of the semantics of the left-hand side. Assuming that $\mathcal{P}(sem : y) \neq some\_atom$ for at least some of the possible goal semantics, we can see that Shieber's semantic filter can lead to incompleteness. This particular rule can produce derivations that correspond to the goal semantics, even if the item used to fill the right-hand side slot does not have semantics that subsume a portion of the goal semantics. However, the item used to fill that slot would not pass the semantic filter in such cases.

Conversely, under the assumption of semantic monotonicity, we can safely ignore items with semantics that do not subsume part of the goal semantics, since their semantics would eventually end up as a part of the semantics of a derivation and the derivation would not correspond to the goal semantics.

While Shieber's semantic filter is effective in excluding items that do not correspond to the goal semantics, it is not completely satisfactory from the performance perspective. It requires the complete unification of a slot, before the filter is applied, and unification is a relatively expensive operation in a grammar that uses complex attribute-value structures. Afterwards, the filter

[2]Shieber [1988], pp. 617

$$\begin{bmatrix} \text{HD} & \left[\text{LEX} \quad \textit{berisp}\right] \\ \text{CAT} & \textit{smain} \\ \text{OBJ1} & \begin{bmatrix} \text{HD} & \left[\text{LEX} \quad \textit{wethouder}\right] \\ \text{CAT} & \textit{np} \\ \text{DET} & \langle\left[\text{HD} \quad \left[\text{LEX} \quad \textit{de}\right]\right]\rangle \end{bmatrix} \end{bmatrix}$$

Figure 3.5: The dependency structure of a verbal phrase where 'de wethouder' is the direct object of *berispt*.

$$_s\left[\text{SEM}_{sem}\left[\text{X} \quad \boxed{1}\right]\right] \rightarrow \quad _a\left[\text{SEM}_{sem}\begin{bmatrix} \text{X} & \boxed{1} \\ \text{Y} & \textit{some\_atom} \end{bmatrix}\right]$$

Figure 3.6: A grammar rule that violates semantic monotonicity — the semantics of the right-hand side slot do not subsume (a portion of) the left-hand side.

has to perform subsumption checks against all portions of the goal semantic, until the subsumption check succeeds against one portion or fails against all portions.

Another interesting question is how these subsumption checks should be performed in the presence of list-valued attributes (Section 2.4.2 and Section 2.4.4).

In the next section we present a mechanism for top-down guidance that makes the construction of items with unwanted dependency structure fail as early as possible, namely during slot unification. We then discuss the handling of list-valued attributes.

3.3.2 Semantic restriction

The mechanism that we use for top-down guidance is related to Shieber's semantic filter, but not identical. In our case, we do assume, as in Shieber [1988], that the grammar exhibits the monotonicity requirement with respect to semantics or dependency structure. But we exploit this requirement one step further: the dependency structure of a lexical attribute-value structure is *instantiated* with part of the abstract dependency structure that is the goal of generation. As a consequence, our bottom-up algorithm is more goal-directed.

When attribute-value structures are constructed for a stem in the abstract dependency structure that is the input to the generator (Section 3.2.4), we also unify their dependency structure attributes (DT) with the relevant part of the abstract dependency structure. This has two consequences: (1) lexical attribute-value structures that have a dependency structure that is incompatible with (the relevant part of) the abstract dependency structure are excluded; and (2) the dependents of a word are listed with their dependency relation in the dependency structures of the remaining attribute-value structures. Instantiating the dependency structure of a word with its dependents has the consequence that right-hand side slots are not only constrained by the syntactic restrictions of the grammar rule or the syntactic head, but also by semantic restrictions (in our case by specific dependents). If a candidate for completion would introduce a dependency relation that is not specified in the abstract dependency structure, and hence in the attribute-value structure of a word, it would simply not unify with the attribute-value structure of the right-hand side slot.

As an example, consider the dependency structure given in Figure 3.7. When the lexical items are created to initialize the agenda, this is done with the additional requirement that the value of the *dt* attribute unifies with this dependency structure or with a sub-part of it. This implies that only the words *de*, *advies*, *adviezen* and inflectional variants of the verb *beklijven* are selected during lexical lookup. Moreover, the dependency structure of those lexical entries is already instantiated with parts of the dependency structure of 3.7. For instance, the lexical attribute-value structure for *beklijfden* is given in Figure 3.8(a). Top-down guidance results in the structure given in figure 3.8(b).

$$\begin{bmatrix} \text{HD} & \begin{bmatrix} \text{LEX} & \textit{beklijf} \end{bmatrix} \\ \text{CAT} & \textit{smain} \\ \text{SU} & \begin{bmatrix} \text{HD} & \begin{bmatrix} \text{LEX} & \textit{advies} \end{bmatrix} \\ \text{CAT} & \textit{np} \\ \text{DET} & \langle \begin{bmatrix} \text{HD} & \begin{bmatrix} \text{LEX} & \textit{de} \end{bmatrix} \end{bmatrix} \rangle \end{bmatrix} \end{bmatrix}$$

Figure 3.7: Attribute-value structure for *De adviezen beklijven.*

In Figure 3.8(a), the attribute-value structure states that the subcat list (the attribute SC) of the verb contains a single entry which is identical to the subject (attribute SUBJ). The dependency structure associated with the subject

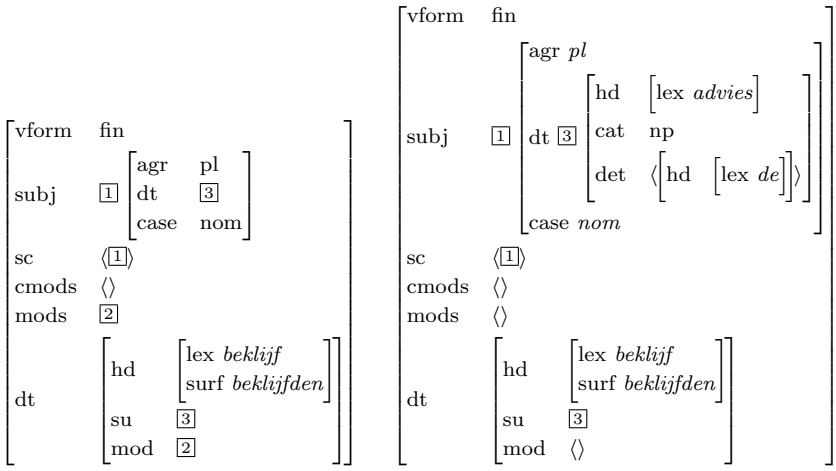


Figure 3.8: Attribute-value structure for *beklijfden* before (a) and after (b) top-down guidance.

is identical to the SU of the dependency structure of the verb. In addition, the verb can take an arbitrary number of modifiers. The pair of attribute CMODS and MODS are used to collect those modifiers in a derivation; CMODS contains the list of modifiers found so far, whereas MODS represents the complete list of modifiers (Section 2.4.4).

In Figure 3.8(b), the dependency structure has been instantiated as a result of top-down guidance. Therefore, the verb can only combine with a subject which has a dependency structure associated with *de adviezen*. Moreover, the attribute value structure indicates that there cannot be a single modifier attached to the verb, nor any other dependents.

Top-down guidance not only prevents the construction of items with a dependency structure that is not a portion of the abstract dependency structure, but it also enforces that all required dependencies are found. For instance, if the input contains a dependency structure of a verb with a subject and a direct object, then lexical lookup will typically not propose an attribute-value structure for the intransitive reading of that verb: that attribute-value structure will have, in the lexicon, the value *nil* for the attribute *obj1* which will not unify with the goal. Similarly, if the input contains a number of modifiers associated with a head, then, typically, maximal projections of that head with fewer modifiers are also ruled out, solving one of the problems raised in Kay

[1996].

Top-down guidance is thus achieved by instantiating the dependency structure of each lexical attribute-value structure with the corresponding part of the dependency structure of the input. In addition, Shieber's semantic monotonicity requirement is enforced for other items as well. A new item is constructed only in case its dependency structure is unifiable with any part of the dependency structure of the goal.

We discussed our method for top-down guidance in the context of abstract dependency structure, but it is applicable for the same grammars and abstract representations as Shieber's semantic filter.

3.3.3 List-valued attributes

As discussed in Section 2.4.4, modifiers are stored in lists in the Alpino grammar, since a head can take an arbitrary number of modifiers. This poses two problems for semantic restriction:

1. If a modifier list contains more than one element, we cannot rely on unification to check whether the necessary modifiers are present. Consider for instance the dependency structure in Figure 3.1. The modifiers *warme*, *chique*, and *gele* can be realized in any order. Consequently, the order of the modifier list can vary. Two lists with the same elements, but different orders, do not unify, where in our application we want to view them as equivalent.

   In other words, the modifiers of a head are stored as a list for practical reasons, though they actually form a multi-set.

2. Since the MODS attribute is unified with the collected modifiers list CMODS when maximal projection is reached (Section 2.4.4), the validity of adding a modifier is checked at a very late stage. As a result, the generator could construct items where a head has modifiers that were not specified in the input. Only at a later stage it would find these items to be useless.

To solve these problems, we added two refinements. (1) If a modifier list in a (constructed) dependency structure is list-valued and contains more than one element, we do not require this list to unify with the corresponding list in the input, but rather to be a permutation of it. (2) When adding a modifier to a collection list (such as CMODS), we require that this modifier is present in the corresponding list of modifiers of that head (or in the case of e.g. control verbs the relevant dependent). This assures that it is not possible to add the wrong modifiers to a head, but does not prevent that a valid modifier is used

more than once. To assure that modifiers are only added once, we also need to verify that the bag of modifiers is a sub-bag of the final bag of modifiers. Since this would be too expensive, we use *bit vector filtering* instead.

3.3.4 Bit vector filtering

Since the top-down guidance method that was proposed in this section specifies the dependencies of a head exactly, one would expect that a lexical item cannot be used more than once by accident. This is true with respect to non-list dependents, since a head will only take the dependents that are specified in its dependency structure. However, top-down guidance does not prevent the duplication of entries in collection lists. As discussed in the previous subsection, although it is verified that a modifier that is added to a modifier collection list is a valid modifier of the head, the modifier list is only checked to be a permutation of the intended list at the maximal projection. As a consequence, a head could collect a dependent more than once before the permutation check occurs. For instance, given a rule $N \rightarrow Adj\ N$ and the dependency structure in Figure 3.1, the generator would generate an infinite number of N categories, containing an arbitrary number of *warme*, *chique*, and *gele* adjectives. We will discuss a solution to this problem that is proposed in the literature and then show that this solution does indeed work for our grammar and generator.

The stated problem is very similar to a problem that occurs in the parsing of discontinuous constituents. If a grammar describes discontinuous constituents, one cannot simply rely on the span of an item to infer which words that item covers. This is problematic, because one cannot easily guarantee when combining items that each word in the spans of the items is analyzed once and only once. Johnson [1985] proposes to associate an index to each word (its position) and to store in each item the set of indices of words that were used in the derivation of that item. As Popowich [1996], we follow exactly the same approach in generation. If we give a unique index to each lexical item, we can store in each passive or active item the set of indices that identify the lexical items that were used directly or indirectly to infer that item. For instance, if we would generate the NP *de hond* 'the dog', its set would contain the indices of *de* and *hond*. When an inference rule is applied, it is guaranteed that no lexical item is used more than once if the intersection of the sets of the active and the passive items is empty. If $\mathcal{L}(i)$ is the set of indices of the lexical items that were used in the derivation of item i, then:

$$\mathcal{L}([A \rightarrow \alpha \bullet B\beta]) \ \cap \ \mathcal{L}([A \rightarrow \gamma\bullet] \equiv \emptyset \tag{3.13}$$

The lexical item set of the newly inferred item is then the union of both

sets:

$$\mathcal{L}([A \rightarrow \alpha B \bullet \beta]) = \mathcal{L}([A \rightarrow \alpha \bullet B\beta]) \ \cup \ \mathcal{L}([A \rightarrow \gamma\bullet] \tag{3.14}$$

It is well-known in set theory that, if the possible elements E in a set is finite and we have a function $f : e \rightarrow \mathbb{N}^0$ and $\forall_{e\in E}\forall_{e'\in E}[f(e) = f(e') \iff e = e']$, then we can represent sets as a bit vector:

- $\{\} \equiv 0$
- $\{e\} \equiv 1 \ll f(e)$, where $\ll$ is a bitwise left shift.
- $\{e\} \cup \{e'\} \equiv f(e) \mid f(e')$, where $\mid$ is bitwise-OR.
- $\{e\} \cap \{e'\} \equiv f(e)$ & $f(e')$, where & is bitwise-AND.

Bit vectors provide an efficient representation for finite sets such those for identifying lexical item coverage, since they are small and offer fast set difference and union operations.

As said, we do not require bit vector filtering to the same extend as Johnson [1985] and Popowich [1996], since our method for top-down guidance already guarantees uniqueness for single-valued dependencies. However, we will give two examples where bit vector filtering of lexical items reduces complexity in generation: in modifier collection and in the handling of extraposed phrases.

Modifier collection As noted in the previous section, a modifier could be used more than once in a modifier collection list. Bit vector filtering prevents this, since adding a modifier twice would fail the check that requires that the intersection of the sets of lexical items used by the active and passive items is empty.

For example, consider the dependency structure in Figure 3.1 once more. Suppose that we give lexical items in this dependency structure the bit vectors in Table 3.2 and that one particular item contains the realization *chique gele trui*. This item has the bit vector 1101 (0001 | 0100 | 1000). If the generator attempts to combine this item with *gele* once more, it will fail the bit vector check, since 1101 & 1000 $\not\equiv$ 0.

Extraposition Internally, the Alpino grammar also uses lists that do not directly correspond to lists in the dependency structure. One such list keeps track of extraposed phrases. Without bit vector filtering, the grammar would recursively try to add the same extraposition to the extraposition list over and over. As a result, the generator would not terminate.

Lexical item	bit vector
trui	0001
warm	0010
chique	0100
geel	1000

Table 3.2: One possible distribution of bit vectors among the lexical items in Figure 3.1.

By using bit vector filtering, any attempt to add an extraposition of the same dependent more than once will be blocked by the bit vector filter.

3.4 Packing

When complex dependency structures are used as the input to the sentence realizer, the chart can grow enormously. However, there may be a large number of items with the same attribute-value structure. For instance, the use of optional punctuation does not change an attribute-value structure, while it does introduce new items for all allowed configurations of punctuation. Another source of growth of the chart are lexical items that have more than one valid inflection.

To compress the chart, we apply *packing*. In packing one passive item can represent multiple derivation histories that share the same attribute-value structure. A derivation history is represented by simple items that contain a number that maps the item one-to-one to a passive item and the identifier of the rule or lexical item that was used to construct the item. In the case of a rule, we also include a list of pointers to passive items that were used to complete the right-hand side slots of that rule.

For instance, consider the passive items $[A \rightarrow \alpha\bullet]$, $[A \rightarrow \beta\bullet]$, and $[A \rightarrow \gamma\bullet]$, that have an identical left-hand side attribute-value structure A and were constructed with the *top_start* rule. Since these three items represent three different manners in which the same attribute-value structure can be created, we could represent them using simple history items. For instance, if this was the 31st unique attribute-value item that was constructed during generation, we can make the history items that correspond to these passive items:

`his(31,r(top_start,`α`))`
`his(31,r(top_start,`β`))`
`his(31,r(top_start,`γ`))`

However, since the right-hand side consists of attribute-value structures that were also packed using this representation, we can replace the attribute-value structures by the history identifiers for these attribute-value structures. For example:[3]

```
his(31,r(top_start,[20,25]))
his(31,r(top_start,[23,29]))
his(31,r(top_start,[23,30]))
```

In other words, the history `his(31,r(top_start,[23,30]))` represents one particular way the attribute-value structure of passive item *31* was constructed, namely, by completing the *top_start* grammar rule using passive items *23* and *30*. The derivation histories of lexical nodes contain lexical information, such as the stem and the Alpino part of speech tag, rather than a list of passive item pointers.

Packing is performed when a passive item is created. If a passive item is found on the chart with the same attribute-value structure, the history of the new passive item is added to the existing passive item. We have also experimented with forward-packing (where packing is applied when the new item is subsumed by an passive item on the chart) and backward-packing (where the new item subsumes an passive item on the chart). However, the benefits of both forms of packing did not outweigh the decreased performance caused applying subsumption checking: (1) we subsumption checking had to be applied twice (for backward and forward packing); and (2) we look up identical attribute-value structures by hash code, after freezing blocked goals (Section 2.4.1) and numbering variables.

3.4.1 Unpacking

After chart generation, full and partial realizations can be retrieved (unpacked) from the 'packed forest'. Unpacking creates a derivation tree for these realizations. A (fully) realized sentence is represented by items with a top category and dependency structures that unify with the dependency structure that was the input to sentence realization. Derivation trees are constructed by expanding histories top-down. Algorithm 2 summarizes how a derivation tree with a particular identifier (*id*) is unpacked - all the histories with the identifier are retrieved and unpacked with a rule or lexical item unpacking function, depending on the type of history.

[3] In the actual generator, this replacement is already performed after the unification of a slot. The identifiers were made up for this example.

Algorithm 2 Unpacking

```
function UNPACK(id)
    histories ← RETRIEVE_HISTORIES(id)
    unpacked ← []
    for all his ∈ histories do
        if IS_RULE_HISTORY(his) then
            APPEND(unpacked,UNPACK_RULE(his))
        else
            APPEND(unpacked,UNPACK_LEX(his))
        end if
    end for
    return unpacked
end function
```

Most of the actual work is done in the function that unpacks histories of items that were constructed using grammar rules. The pseudo-code of this function is shown in Algorithm 3. As discussed in the previous section, the history of such an item contains the identifier of the grammar rule, as well as list of pointers to histories that complete the right-hand side slots. UNPACK_RULE unpacks the derivations of the right-hand side slots, giving a list where the n-th element is a list of derivation trees that completed the n-th slot on the right-hand side in this item. It then takes the Cartesian product of these 'sets' to obtain all possible combinations of derivations to complete the right-hand side slots. For each such combination, the attribute value structure is constructed by completing the right-hand side slots with the attribute-value structures in the derivation trees of each combination. Finally, each possible derivation of the item is wrapped in a special *tree* term.

The unpacking function for lexical items is shown in Algorithm 4. This function simply gets the lexical information in the history item, retrieves the attribute-value structure of the item, and returns the item in tree representation.

Since the packed forest can contain an enormous number of realizations, depending on the application we may want to unpack all or a specific number of realizations. In chapter 4 we describe how we apply *N-best* unpacking to retrieve the N most fluent realizations.

Algorithm 3 Unpacking of non-terminal items.

```
function UNPACK_RULE(history)
    rule_id, daughter_ids ← history
    lhs, rhs ← GRAMMAR_RULE(rule_id)
    unpacked_daughters ← []
    for all id ∈ daughter_ids do
        PUSH_BACK(unpacked_daughters,UNPACK(id))
    end for
    trees ← []
    daughter_combs ← CARTESIAN_PRODUCT(unpacked_daughters)
    for all comb ∈ daughter_combs do
        fresh_lhs ← UNIFY_RHS(lhs,rhs,comb)
        APPEND(trees,tree(rule_id, fresh_lhs, comb)
    end for
end function
```

Algorithm 4 Unpacking of terminal items.

```
function UNPACK_LEX(history)
    lex_info ← history
    attr_val ← GET_LEX_ITEM(lex_info)
    return [tree(lex_info, attr_val, [])]
end function
```

3.4.2 Punctuation

Although punctuation is not specified in an abstract dependency structure (Section 3.1), the use of punctuation can improve fluency or may even be required. The Alpino grammar uses different lexical types for optional and required punctuation. Optionality is achieved by providing an ϵ punctuation sign of the optional punctuation type.

Since optional punctuation can be used in many configurations, mostly to be robust in parsing, a tremendous number of competing derivations can be unpacked from the packed forest. Initially it seemed appropriate to let the fluency ranking component decide on such variations. However, unpacking all derivations was unacceptably slow, even when applying beam search (Chapter 4). For this reason, we eventually disabled the use of optional punctuation in the generator.

This problem also surfaced, although less profoundly, when using only non-

optional punctuation. However, we cannot exclude punctuation completely, since the use of a rule that has mandatory punctuation is sometimes required to realize an abstract dependency structure. For this reason, we introduce as little punctuation as possible to retrieve realizations of the input from the forest.

Extraction of realizations using the smallest amount of punctuation is achieved through *iterative deepening*. First we attempt to unpack realizations with no punctuation token. When this fails, we attempt to unpack realizations with one punctuation token. Et cetera. The algorithm is summarized in Algorithm 5, and uses a special version of UNPACK that unpacks trees such that the given number of punctuation signs is used.

Algorithm 5 Iterative deepening

```
derivations ← []
n_punct ← -1
repeat
    n_punct ← n_punct + 1
    derivations ← UNPACK_PUNCT(id,n_punct)
until ¬EMPTY(derivations) or n_punct = max_punct
```

In the future it would be worthwhile to explore methods to introduce optional punctuation judiciously to improve fluency.

3.5 Evaluation

3.5.1 Experimental setup

A simple experimental setup was used to monitor coverage and performance of the generator during its development. In this setup, we keep track of the number of dependency structures for which realizations could be constructed (coverage), the number of realizations, and the time required for constructing all realizations for a given dependency structure. Using this information, we can extract various interesting characteristics, such as the coverage of the realizer, and the average generation time for a dependency tree of a certain complexity.

The realizer was tested using three Alpino test suites (the so-called *g*, *h*, and *i* suites). These suites where created during the development of the grammar, are generally of increasing complexity, and cover a wide array of lexical and syntactic phenomena.

The abstract dependency structures that are the input to the generator are constructed by parsing each sentence in a suite, extracting the dependency structure that resembles the manual annotation the most. We then remove information to form an abstract dependency structure (Section 2.3).

It may seem more natural to form abstract dependency structures based on the manually corrected dependency structures. However, the problem with the gold standard structures is that the grammar may not be able to construct them. As a result, we would not be able to generate from those structures. The goal of the generator is to be able to construct sentences for abstract dependency structures that are valid in the grammar. For this reason, we construct the abstract dependency structure from the best parse of a sentence. It is the best dependency structure that is also valid according to the grammar.

To select the best parse of a sentence, we choose the parse with the dependency structure that is the most similar to the annotation. The similarity of a dependency structure of a parse and an annotation is estimated using concept accuracy (CA). Concept accuracy is a parsing accuracy measure that is normally applied to a complete evaluation corpus. Let D_p^i be the number of dependencies produced by the parser for sentence i, D_g^i is the number of dependencies in the manual annotation of i, and D_o^i is the number of correct dependencies produced by the parser. *Concept accuracy* (CA) is defined as:

$$\text{CA} = \frac{\sum_i D_o^i}{\sum_i \max(D_g^i, D_p^i)} \tag{3.15}$$

In this case we select the best annotation on a per-sentence basis. Consequently, we calculate the concept accuracy on a per-sentence basis:

$$CA_i = \frac{D_o^i}{\max(D_g^i, D_p^i)} \tag{3.16}$$

3.6 Results

Table 3.3 shows the coverage of the sentence realizer on the *g*, *h*, and *i* suites. As we can see in this table, coverage is good for the Alpino test suites. Most of the remaining problems are caused by components of the productive lexicon that are not truly reversible or post-processing that the parser applies to the dependency structures. There are also some less interesting cases, such as empty dependency structures, derived from 'sentences' that only contain punctuation. We give examples of two sources of typical errors.

Suite	Inputs	$\geq$ 1 realization	Coverage (%)
g_suite	996	995	99.9
h_suite	990	960	97.0
i_suite	271	260	95.9

Table 3.3: Coverage of the chart generator on various test suites.

Productive lexicon

One fairly common reason the generator could not produce a realization is that parts of the dependency structure are constructed by components of the productive lexicon that are currently only available during parsing. For example, the lexicon contains a definite clause grammar for analysing phrases such as times, dates, numbers, and amounts. This definite clause grammar makes it possible to analyze phrases such as those underlined in the following sentences:

- *De winst bedroeg tussen de 3 en de 3.5 miljoen gulden.*
  (The profit amounted between the 3 and the 3.5 million guilders.)
- *We moesten een boek of zes lezen. (We had to read a book or six.)*
- *De twee na laatste ploegen verdwijnen. (The second to last teams disappear.)*

Unfortunately, we do not have an efficient method to use this definite clause grammar in the opposite direction yet.

Post-processed dependency structure

The parser applies transformations to some dependency structures as a post-processing step. There are cases where annotation standards require a structure that the current grammar cannot create. For instance, the sentence *Ik heb platen gedraaid en films gekeken.* has the dependency structure shown in Figure 3.9, but is then transformed to the structure in Figure 3.10 that is according the annotation standard.

The first dependency structure provides an analysis as if there were two sentences (*ik heb platen gedraaid* and *ik heb films gekeken*), using reentrancy to indicate that the subject and main verb are shared between both conjuncts. The dependency structure derived after post-processing shows more clearly that the *VC* is actually a conjunction. The generator is currently not equipped to reverse these post-processing steps.

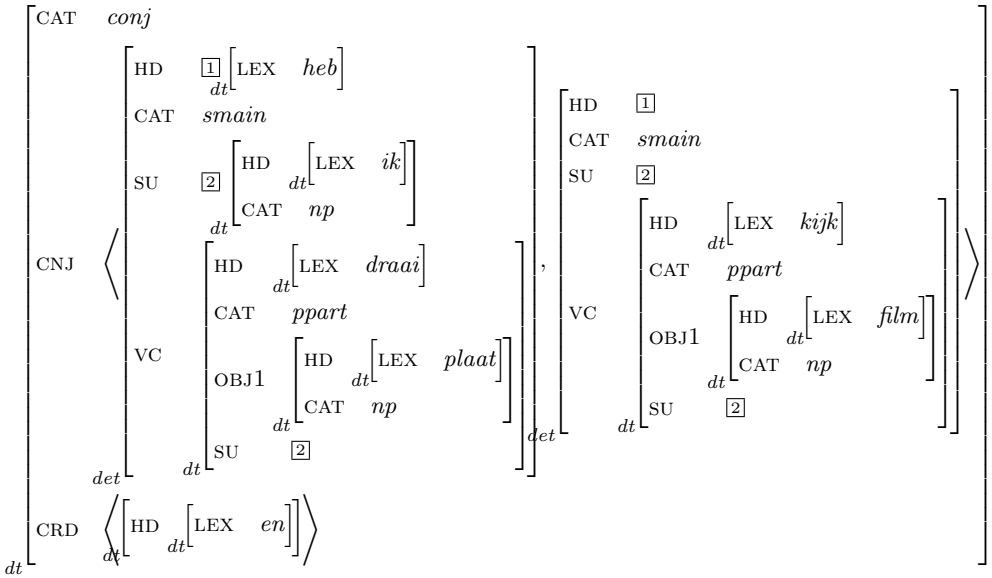


Figure 3.9: Dependency structure for *Ik heb platen gedraaid en films gekeken.* before post-processing.

3.7 Conclusion

In this chapter, we gave an overview of the chart generator in the Alpino system. Although chart generation itself is a simple and elegant algorithm, effective top-down guidance is required to make chart generation efficient. Our method for top-down guidance unifies the dependency structure of the initial lexical items with parts of the abstract dependency structure that was the input to generation. In this manner we assure that items are combined in such a manner that only the correct dependencies are introduced.

Since the usefulness of a generator is highly dependent on the coverage of its grammar and lexicon, the next chapter makes a short detour in techniques to improve the coverage of a grammar and lexicon. Chapter 8 introduces methods to improve computational lexicons and grammars. In Chapter 4 we introduce a stochastic component to rank realizations by fluency.

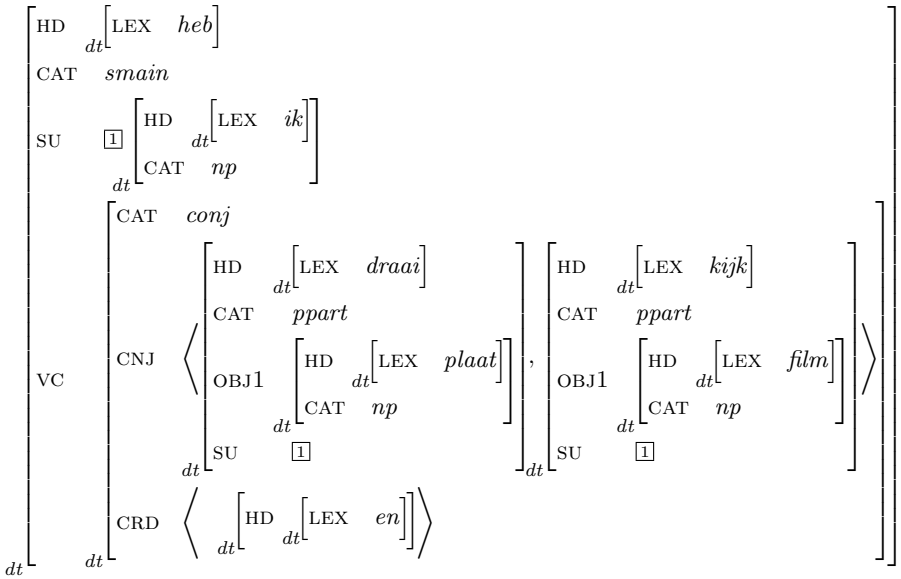


Figure 3.10: Dependency structure for *Ik heb platen gedraaid en films gekeken.* after post-processing.

Chapter 4

Fluency ranking

4.1 Introduction

Chapter 3 describes a chart generator that uses the grammar and lexicon of the Alpino system. This generator generates sentences that are supposed to be syntactically correct. However, many of these so-called realizations of the input would not be used by a native speaker of a language. For instance, generating from the abstract dependency structure corresponding to *Harm luisterde gisteren met Daniël naar Miles Davis* 'Harm listened yesterday with Daniël to Miles Davis', the generator produces 24 different realization, such as:

(1) a. Harm luisterde gisteren met Daniël naar Miles Davis
b. gisteren luisterde Harm met Daniël naar Miles Davis
c. met Daniël luisterde Harm gisteren naar Miles Davis
d. Harm luisterde met Daniël naar Miles Davis gisteren
e. naar Miles Davis luisterde Harm met Daniël gisteren

Although these realizations are syntactically correct realizations of the input, they differ highly in what is called *fluency*. A sentence is said to be fluent if it has a mix of a comprehensible syntactic structure, frequent words and obeys conventions. As we will discuss in Chapter 5, fluency has a strong relation with sentence comprehension — we often consider those sentences that are most likely to be understood correctly, to be the most fluent. Since we expect a generation system to produce a sentence that is comprehensible and conventional, we also need a *fluency ranker* to choose a realization that is not only grammatical, but also perceived by a human as fluent.

In this chapter, we will first analyze this problem, by discussing various aspects which play a role in the fluency of a sentence. We will argue that these aspects are so varied in nature that a model is required that allows us to consider potentially any characteristic of the derivation. We will first discuss n-gram language models, which have traditionally been used to rank sentences. It will then be shown that n-gram models cannot model all the different aspects of fluency. We will then discuss maximum entropy modeling, which allows us to construct models incorporating any characteristic of a derivation as a feature. We will then discuss the features that we use in our fluency ranking model. Finally, we discuss how we evaluate our model and provide the results of this evaluation.

4.1.1 Different aspects of fluency

In this section, we will introduce the problem of fluency ranking more extensively, by discussing different aspects or preferences that influence the fluency of a sentence, using examples. In some examples, one realization is clearly preferred over another. In other examples, this is not as clear-cut — the preference may depend on the domain, or the realizations may even be considered equally fluent. Aspects of fluency that will be described are: syntactic preferences, fixed expressions, prenominal modifiers, and conjunctions. This description is by no means complete, but aims to show that most of these aspects are of a very different nature — some of them can be described using straightforward rules, others require knowledge of semantics, pragmatics, tradition, and custom.

Syntactic preferences One of the most important classes of preferences in fluency ranking are syntactic preferences. This is expected, since sentences that do not have a conventional syntactic structure to realize an abstract structure can be misinterpreted or are hard to grasp.

The preference for subject fronting is one of the simplest and foremost syntactic preferences in Dutch. In Dutch, both the subject and the direct object can be fronted in declarative main clauses. Consider the following (syntactically correct) realizations of an abstract dependency structure:

(2) a. **De baseliner** speelde ter voorbereiding op het Grand Slam **één grastoernooi**.
The baseliner played for preparation of the Grand Slam one grass-tournament.

b. **Eén grastoernooi** speelde **de baseliner** ter voorbereiding op het Grand Slam.

Although both realizations are permitted in Dutch, the first realization, which fronts the subject, is clearly preferred. The second realization would only be used to emphasize that the baseliner only played 'one grass tournament'.

In the preceding example, fronting of the direct object does not lead to misinterpretation, since we know that a grass tournament cannot play, hence it cannot be the subject of *speelde*. Direct object fronting becomes outright confusing for a reader if it is not possible to discern the subject and the direct object, other than by their position in the sentence. For example, the second realization in the following example will give a reader the impression that 'Van 't Klooster' and 'Milliard' hit a ball, rather than 'Gumbs' and 'Balentina':

(3) a. **Gumbs en Balentina** sloegen **Van 't Klooster en Milliard** over de thuisplaat (5-3).
Gumbs and Belentina hit Van 't Klooster and Milliard over the home-plate (5-3).
Gumbs and Belentina's hits made it possible for Van 't Klooster and Milliard to pass the home plate. (5-3)

b. **Van 't Klooster en Milliard** sloegen **Gumbs en Balentina** over de thuisplaat (5-3).

Another example of a syntactic preference in fluency is modifier adjunction. If a verb in a verb cluster is modified, we generally prefer left-adjoining modification. For instance, the first of the following realizations is preferred, because *in de vermakelijke en chaotische eindfase* is on the left side of the verb *voorkomen* in the *VC*.

(4) a. Ze konden de gevreesde massasprint in de vermakelijke en chaotische eindfase niet voorkomen.
They could the feared mass-sprint in the enjoyable and chaotic end-phase not prevent.

b. Ze konden de gevreesde massasprint niet voorkomen in de vermakelijke en chaotische eindfase.

The last syntactic preference in fluency that we discuss is ordering in the *middle field*. Dutch is a language with a verb-second word order, where the second constituent of declarative main clauses is always a finite verb. For example:

(5) a. Ik **heb** Jan gisteren gesproken
I have Jan yesterday spoken-to

b. Gisteren **heb** ik Jan gesproken

c. Jan **heb** ik gisteren gesproken

The usual analysis of this phenomenon is that Dutch sentences have an SOV (subject-object-verb) order, where the finite verb moves to the second position in declarative main clauses.

The constituents between the verb in the second position and the remaining verbs in the *verb cluster* in the final position are said to form the *middle field*. We repeat the examples above, marking the middle field:

(6) a. Ik heb **Jan gisteren** gesproken
b. Gisteren heb **ik Jan** gesproken
c. Jan heb **ik gisteren** gesproken

If the verb in the second constituent position is not an auxiliary verb, the *VC* at the end of the sentence is empty. Still, the material between the verb and the *VC* gap is considered to be the middle field. For instance:

(7) a. De twee giganten schudden **elkaar glimlachend de hand**.
The two giants shake each-other smiling the hand.

There is some freeness in the ordering of constituents in the middle field. For instance, the following realizations use different orderings of the middle field that are considered syntactically valid:

(8) a. De Amerikaanse vereniging van psychologen had **het een jaar daarvoor aan de leden** gevraagd.
The American association of psychologists had it a year before to the members asked.
The American association had asked its members a year before.
b. De Amerikaanse vereniging van psychologen had **een jaar daarvoor het aan de leden** gevraagd.
c. De Amerikaanse vereniging van psychologen had **het aan de leden een jaar daarvoor** gevraagd.

The second and third realizations would be considered less fluent than the first. In other words, there are ordering preferences in the middle field: the direct object should be positioned in front of the indirect object and modifiers and it is preferred to put a modifier such as *een jaar daarvoor* before the indirect object.

In the following example, the verb in the verb-second position is not an auxiliary verb. Again, the middle-field in the first realization is the most fluent:

(9) a. De twee giganten schudden **elkaar glimlachend de hand**.
b. De twee giganten schudden **elkaar de hand glimlachend**.

c. De twee giganten schudden **glimlachend elkaar de hand**.

Fixed expressions For the syntactic aspects of fluency, it is conceivable to pick the best realization using a set of rules that encode these preferences. However, the other aspects that we discuss cannot be captured with relatively straightforward rules. For instance, consider the following sentence:

(10) a. In plaats daarvan verdween Locci's echtgenoot Stefano Mele
In place thereof disappeared Locci's husband Stefano Mele
voor veertien jaar achter **slot en grendel**.
for fourteen years behind lock and latch.
(Instead, Locci's husband Stefano Mele dissapeared behind bars for fourteen years.)

Another realization that the Alpino generator produces is:

(11) a. In plaats daarvan verdween Locci's echtgenoot Stefano Mele voor veertien jaar achter **grendel en slot**.

However, this would not considered be fluent in Dutch — although *slot en grendel* may look like an ordinary conjunction, *achter slot en grendel* is a fixed expression for (something or someone) being locked up. Since fixed expressions are frequent in language, a component that makes decisions about fluency should not only take syntactic preferences into account, but should also have knowledge of a wide variety of fixed expressions.

Prenominal modifiers Given that fixed expressions do not follow general rules, but occur relatively frequently, we could simply collect a list of fixed expressions and use that list to wield out incorrectly realized variants of fixed expressions. However, there are also surface-oriented preferences that we cannot judge in such a manner. For instance, the ordering of prenominal modifiers affects fluency, but cannot be described using relatively simple rules, nor can we enumerate every possible ordering. For instance, the prenominal modifiers marked in the following sentences cannot be swapped without impacting fluency:

(12) a. Jean Paul de Bruijn is de **nieuwe Europese** kampioen
Jean Paul de Bruijn is the new European champion
bandstoten.
cushion-caroms.

b. Djindjic zei dat dit weekeinde in reactie op de **ernstige**
Djindjic said that this weekend in reaction to the severe
politieke crisis in Joegoslavië.
political crisis in Yugoslavia.

c. Ook **verschillende regionale** ministers uit Vlaanderen,
Also several regional ministers from Flanders,
Wallonië en Brussel mogen meedraaien in het circus.
Wallonia, and Brussels may participate in the circus.

It could be argued that the adjective describing the most defining property tends to be the closest to the noun. For instance, *politieke crisis* provides more information than *ernstige crisis*. However, this does not always seem to be the case. For instance, the first of the following two realizations would be considered more fluent:

(13) a. Ik zag gisteren de **snelle gele** auto.
I saw yesterday the fast yellow car.
b. Ik zag gisteren de **gele snelle** auto.

In this case, *snelle* and *gele* could be equally defining properties. However, we prefer to have the color closer to the noun. It is also possible that a different order has a different meaning, that would ideally have different input representations:

(14) a. Zijn **twintigste fantastische** goal.
His twentieth fantastic goal.
b. Zijn **fantastische twintigste** goal.

The first realization means that someone scored his twentieth fantastic goal (out of more) and the second realization means that he scored his twentieth goal, which was fantastic.

Given the examples above, it is not surprising that most theories that have attempted to describe the ordering of prenominal modifiers assume a relationship between semantics and position. However, it is not inconceivable that phonetic or phonological aspects also play a role in the ordering of prenominal modifiers.

But even if we did have an exact set of rules for ordering modifiers, based on semantic, phonetic, and phonological features, it would be difficult to integrate such rules in the Alpino system. Alpino's dependency structures do not allow us to specify the intended meaning of a noun phrase, nor does the lexicon provide a detailed semantic description of modifiers that could help in making ordering decisions.

Even though we do not know the exact underlying rules for the ordering of prenominal modifiers and dependency structures may not provide enough information if such rules did exist, we can attempt to deduce the best ordering via other means. For instance, modifier-noun combinations are collocations, such as *politieke partij* 'political party', *Europese kampioen'* 'European champion', or *regenachtige dag* 'rainy day'. Such combinations can be prioritized based on corpus statistics. The same thing applies to combinations of prenominal modifiers.

Conjunctions Another construction where fluency is not determined syntactically are conjunctions. In many conjunctions, there is no clear preference for a specific ordering of conjuncts. In the following sentence, we can swap the conjuncts *Zenit Sint Petersburg* and *Spartak Moskou* without a loss of fluency.

(15) a. Tijdens de competitiewedstrijd tussen **Zenit Sint Petersburg en Spartak Moskou** liep het uit de hand.
During the competition-game between Zenit Saint Petersburg and Spartak Moscow went it out of hand
(Things got out of hand during the competition game between Zenit St. Petersburg and Spartak Moscow.)

However, often the order of conjuncts is important. For instance, the following sentence has two time spans described using the conjunctions *28 uur en 45 minuten* '28 hours and 45 minutes' and *27 uur en 30 minuten*:

(16) a. Sterry wist het preken **28 uur en 45 minuten** vol te houden, het oude record staat op **27 uur en 30 minuten**.
Sterry knew the speaking 28 hour and 45 minutes persist, the old record is 27 hour and 30 minutes.
(Sterry was able to persist in speaking for **28 hours and 45 minutes**, the previous record was **27 hours and 30 minutes**.)

The following realization of the same abstract dependency structure would be considered odd:

(17) Sterry wist het preken **45 minuten en 28 uur** vol te houden, het oude record staat op **30 minuten en 27 uur**.

The rule in such conjunctions is that we prefer to see larger units before smaller units (ie. 'ten meters and five centimeters'). Usually, a comparable rule applies

to conjunctions where the conjuncts are numbers such as years or quantities. For example, it would be odd to realize *zestig en zeventig* as *zeventig en zestig* in the following sentence:

(18) a. In de jaren **zestig en zeventig** maakte Roos naam met een
In the years sixty and seventy made Roos name with a
persoonlijk getinte human-interest-pagina.
personally tinted human-interest-page.
(In the sixties and seventies Roos became known for a personal human-interest page.)

Another situation where conjuncts cannot be ordered randomly, is when there is a relation between the conjuncts that requires the conjuncts to have a particular order. For instance, if a conjunction contains co-references, the referent should be introduced first. In other words,

(19) a. Hij plukt een handvol graan, bekijkt het met een zuur gezicht
He reaps a handful cereal, looks-at it with a sour face
en legt uit waarom de oogst dit jaar slecht zal zijn.
and explains why the harvest this year bad will be.

is a fluent sentence, while

(20) a. Hij bekijkt het met een zuur gezicht, plukt een handvol graan en legt uit waarom de oogst dit jaar slecht zal zijn.

is not.

The relation between the conjuncts can also be temporal. In such cases there are often no direct indicators that could help deciding on an order. For example:

(21) a. Ze pakt een pen, schrijft een notitie en roept een
She takes a pencil, writes a note and calls a
bode.
messenger.

In many such cases, the input representation used by the generator should provide the means to encode phenomena such as co-references, temporal relations, or conjunction order.

4.1.2 Stochastic models for fluency ranking

As we can see in the examples in the previous section, a practical generation system cannot present a random realization. Instead, realizations should be ranked by fluency, and the most fluent realization(s) should be presented to the user of the system. Such a component should be of a stochastic nature — although the most likely realization may be the most fluent, in some circumstances another realization is picked by the speaker. In other words, a fluency ranking component should provide a probability distribution over all possible outcomes.

In the next section, we will discuss n-gram language models, which estimate the probability of a realization based on the sentence or a simple abstraction thereof, such as the sequence of part-of-speech tags. As we will argue, n-gram models can 'eliminate' many realizations that are not fluent. However, some other aspects, such as syntactic preferences, cannot be judged adequately by n-gram models. We will discuss conditional maximum entropy models, which allow us to integrate arbitrary characteristics of a realization, in Section 4.3.

4.2 N-gram language models

4.2.1 Model

One approach to choose the best of a set of generated realizations is choosing the realization that occurs most frequently within human-written texts in the domain that is targeted by the generator (e.g. newspaper text). If we assume that human writers attempt to produce sentences that are as fluent as possible for that domain and ignore the relation between the input and the generated sentence, this approach would select a fluent sentence.

Obviously, this method is also naive. Since any sample of language that is used for such counting is finite and the number of possible realizations is infinite, most realizations will never occur in a particular sample. In other words, the data is too sparse to make a good estimation of the relative frequencies of realizations. To give a simple example, the sentence *Komt deze zin voor in Wikipedia?* (*Does this sentence occur in Wikipedia?*), does not occur in the Dutch Wikipedia encyclopedia. Consequently, we are not able to estimate the fluency of this perfectly fluent sentence.

The intuition that sentences or fragments that occur frequently are likely to be fluent, however, has been used quite effectively in fluency ranking. Knight and Hatzivassiloglou [1995] proposes to use language models in generation to select the most fluent sentence from a set of candidate realizations. Language

models estimate the probability of a sentence based on n-grams.[1]

A language model model can be obtained as follows. The relative frequency of a sentence in a corpus is an estimate of the probability of that outcome of the random variable.

$$p(w_1^n) = \frac{C(w_1^n)}{N} \tag{4.1}$$

Where w_1^n are the tokens $(w_1, w_2 ... w_n)$ in a sentence. The chain rule allows us to calculate the joint probabilities from conditional probabilities:

$$p(w_1^n) = p(w_1) \prod_{i=2}^{n} p(w_i | w_1^{i-1}) \tag{4.2}$$

Obviously, this step alone will not solve the problem of data sparseness, since the frequency of a sentence in the sample is still required to estimate $p(w_n | w_1^{n-1})$. If the distribution of a word w_{n+1} only depended on the preceding word w_n, then sentence generation would be a Markov chain [Markov, 1954]. We could then calculate the probability of a sentence as follows:

$$p(w_1^n) = p(w_1) \prod_{i=2}^{n} p(w_i | w_{n-1}) \tag{4.3}$$

$$\hat{p}(w_n | w_{n-1}) = \frac{C(w_{n-1}, w_n)}{C(w_{n-1})} \tag{4.4}$$

where $\hat{p}(w_n | w_{n-1})$ is a maximum likelihood estimation based on the training sample. However, since the distribution of a word generally depends on other factors, treating sentence generation as a Markov chain can (only) give an approximation of the probability of a sentence.

Models that estimate the probability of the current word given the previous word are called *bigram models*. Models with a larger context are often used to obtain better estimations of sentence probabilities. However, increasing the context size is a trade-off, since the data to estimate the probability of an n-gram also becomes sparser. Hence, the length of n-grams should be chosen by taking these factors into account. Many existing generation systems have settled on the use of trigrams [Velldal and Oepen, 2006, White et al., 2007, Cahill, 2009]

[1] Language models have a long history in computational linguistics. Goodman [2001] provides a good survey of language modeling techniques.

Even if the length of the context is relatively short, some n-grams may not occur in the sample. It may be that a particular n-gram is so infrequent that it did not occur, or worse, the n-gram contains one or more unknown words. This is problematic, since if the n-gram has a frequency of zero, the conditional probability of the word given its context will be zero, and finally the estimated probability of the sentence will be zero.

Smoothing methods, which were pioneered by Lidstone [1920] and Good [1953], attempt to solve the problem of unseen events by taking away some probability mass from known events and reserving it for unseen events. In other words, for any event y, $p(y) \neq 0$, in a smoothed probability distribution.

In the probability estimation of $p(y|x)$, x, y might not occur in the training sample. In such cases, it is inadequate to use the same smoothed probability for $p(y|x)$. For example, if a language model is used to estimate $p(c|a, b)$ and $C(a, b, c) = 0$, $C(b, c)$ and $C(c)$ may be non-zero. By exploiting probability estimations of shorter contexts, a language model can be more effective in estimating the conditional probability of its word.

Smoothing methods that take smaller contexts (so-called *lower-order distributions*) into account can be divided in two families: *back-off smoothing* and *interpolation smoothing*. Chen and Goodman [1999] shows that both families can be described generically, so we use their notation below. Back-off models revert to a lower-order distribution if an n-gram did not occur in the training sample. If τ is the computed distribution of trigrams, then back-off models calculate the smoothed probability $\dot{p}$ in the following manner:

$$\dot{p}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \tau(w_i|w_{i-n+1}^{i-1}) & \text{if } C(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1})\dot{p}(w_i|w_{i-n+2}^{i-1}) & \text{if } C(w_{i-n+1}^{i}) = 0 \end{cases} \tag{4.5}$$

The factor $\gamma(w_{i-n+1}^{i-1})$ is used to scale a lower-order n-gram probability into the higher-order n-gram probability, to ensure that the conditional probabilities of trigrams sum to one. Interpolation smoothing, on the other hand, always uses the distribution of lower-order n-grams. A scaling factor is used to weigh n-grams of different orders.

$$\dot{p}(w_i|w_{i-n+1}^{i-1}) = \tau(w_i|w_{i-n+1}^{i-1}) + \gamma(w_{i-n+1}^{i-1})\dot{p}(w_i|w_{i-n+2}^{i-1}) \tag{4.6}$$

Since n-gram language models have been shown to work quite effectively [Knight and Hatzivassiloglou, 1995], they have been used in other work exclusively as a fluency ranking components [Langkilde and Knight, 1998, Langkilde, 2000, Langkilde-Geary, 2002, White, 2004].

4.2.2 Implementation

Our n-gram language models for fluency ranking use trigrams, which provide a good trade-off between data availability and use of contextual information. For smoothing, we use a very straightforward version of linear interpolation smoothing:

$$\dot{p}(w_3|w_1,w_2) = \tau(w_3) + \tau(w_3|w_2) + \tau(w_3|w_1,w_2) \tag{4.7}$$

Each distribution is estimated using the empirical distribution in the training sample and a context-independent weight:

$$\begin{array}{lcl} \tau(w_3) & = & \lambda_1 \cdot p(w_3) \\ \tau(w_3|w_2) & = & \lambda_2 \cdot p(w_3|w_2) \\ \tau(w_3|w_1,w_2) & = & \lambda_3 \cdot p(w_3|w_1,w_2) \end{array}$$

The weights are estimated using deleted interpolation [Jelinek, 1980, Brants, 2000]. Deleted interpolation successively removes each trigram from the corpus, and chooses λs such that the likelihood of the training data is maximized. Algorithm 6 describes how the parameters are estimated from the training sample, following Brants [2000].

Algorithm 6 Algorithm for estimating the parameters in linear interpolation smoothing. Here N is the size of the corpus.

$\lambda_1,\ \lambda_2,\ \lambda_3 \leftarrow 0$
for all $w_{n-2}^n \in Corpus$ **do**
  $p_3 \leftarrow \frac{C(w_{n-2}^n)-1}{C(w_{n-2}^{n-1})-1}$
  $p_2 \leftarrow \frac{C(w_{n-1}^n)-1}{C(w_{n-1})-1}$
  $p_1 \leftarrow \frac{C(w_n)-1}{N-1}$
  if $p_3 > p_1$ **and** $p_3 > p_2$ **then**
    $\lambda_3 \leftarrow \lambda_3 + C(w_{n-2}^n)$
  else if $p_2 > p1$ **then**
    $\lambda_2 \leftarrow \lambda_2 + C(w_{n-2}^n)$
  else
    $\lambda_1 \leftarrow \lambda_1 + C(w_{n-2}^n)$
  end if
end for
normalize$(\lambda_1, \lambda_2, \lambda_3)$

If a word does not occur in the training sample, then the interpolated

probabilities are all zero. We have experimented with three different approaches to estimating the probability of such words:

1. The probability of an unknown word estimated using Laplace smoothing: $\frac{C(w_i)+\alpha}{N+d}$, where N is the size of the training sample and d the number of types (we use $\alpha = 0.5$).

2. The probability is estimated using linear interpolation, where the word is replaced by a generic unknown word tag in the estimation of $p(w3|w2)$ and $p(w_3|w_1, w_2)$. The probability of $p(w_3)$ is estimated using Laplace smoothing, since the generic unknown word tag overestimates the probability of an unknown word.

3. As (2) with the addition that the word is also replaced by the generic unknown word tags when the word occurs in the context of known words.

For (2) and (3) we replace low-frequent words by a generic unknown word tag to obtain the distribution of unknown words.

We report on experiments with these approaches in Section 4.7.1.

We also experimented with the modified Kneser-Ney (linear interpolation) smoothing [Chen and Goodman, 1999]. However, since this method did not perform better than linear interpolation smoothing with deleted interpolation in our setup (as reported in Section 4.7.1) and we already used linear interpolation smoothing in our system, we do not use it in later experiments.

4.2.3 Disadvantages

While n-gram models are simple and fast, they have some deficiencies that make it impossible to pick up some important cues that indicate fluency. First of all, n-gram models cannot capture dependencies that go beyond an $n - 1$ span of history. Moreover, even dependencies within this span are often not taken into account due to data sparseness. A more general, related, problem is that n-gram models are purely surface-based. They cannot integrate structural information about the realization process, such as characteristics of the derivations that are built up during sentence realization. Such structural information is required to learn the syntactic preferences that were discussed in Section 4.1.1. For instance, to express the preference for object fronting in the general case, we need access to the attribute-value structures that were constructed by the grammar during parsing to get information about the dependents of the finite verb. Another, less obvious example are the conjunctions that were also described in that section. In the sentence,

(22) a. Sterry wist het preken 28 uur en 45 minuten vol te houden, het oude record staat op 27 uur en 30 minuten.

it helps tremendously if we can recognize the conjunction and its conjuncts.

While n-gram language models are clearly too limited to effectively perform fluency ranking on their own, they do settle a lot of local choices effectively. For example:

- Given a pair of synonyms, a language model will prefer the word that is more frequent within the domain of the training sample.
- A language model will often eliminate incorrect realizations of fixed expressions. For instance, the fixed expression *appels en peren vergelijken* (lit: 'apples and pears comparing', 'comparing apples and oranges') will be preferred over *peren en appels vergelijken*.
- Language models can settle superficial syntactic choices. For instance, while they cannot settle subject-fronting in general, it does prefer subject fronting in cases when pronouns are used. The n-gram model will prefer realizations starting with a pronoun in the nominative case over sentences starting with a pronoun in the accusative case.

Given that n-gram models do settle many interesting local choices, we use two n-gram models as features in the maximum entropy model that we discuss in the next session.

4.3 Maximum entropy modeling

4.3.1 Introduction

As discussed in the previous section, a serious drawback of n-gram language models is that they cannot incorporate structural information to judge the fluency of a realization. Also, language models attempt to estimate the probability of a sentence in a language using a sample. However, in fluency ranking we are actually more interested in the best realization of the abstract dependency structure, which may not be the same. For instance, the first of the following two sentences may be the most likely according to a language model:

(23) a. De burgemeester maande de SP tot spoed.
The mayor urged the SP to hurry.
b. De SP maande de burgemeester tot spoed.

However, if the input specifies that *de SP* is the subject of the sentence, the second sentence is a more fluent realization of the input. If l is the input to generation, we are interested in the sentence s that maximizes $p(s|l)$. For simplicity and efficiency, we use an approximation (similar to *Viterbi approximation* in speech recognition); if $\Omega(l)$ is the yield of l, we want to find the derivation $d \in \Omega(l)$ that maximizes $p(d|l)$.

We cannot directly estimate the conditional probability $p(d|l)$, since most inputs and derivations do not occur in the training data. Instead, derivations can be decomposed in the form of potentially interesting *features*.[2] Each feature is associated with a real-numbered *value* that can encode a boolean, frequency, probability, etc. by convention. Some examples of value types are shown in table 4.1.

Type	**Feature**	**Value**	**Description**
Boolean	syntactic(subj_np_topic)	1	The realization has a topicalized NP subject.
Frequency	rule(np_det_n)	2	The grammar rule with identifier *np_det_n* is used twice in this derivation.
Probability	word_trigram_distribution	64.3	$-log\ p(w_1^n)$ equals 64.3

Table 4.1: Examples of different types of feature-value pairs.

If we want a model for $p(d|l)$ to be able to use arbitrary features, it is difficult to guarantee that features are independent. For instance, a syntactic phenomenon that signals that a phrase is fluent could be modeled concretely through n-grams, or more abstractly by the use of certain grammar rules. Consider the following realizations of a dependency structure:

(24) a. Het is zeker dat hij komt.
It is certain that he comes.
b. Zeker is het dat hij komt.

The fact that the first realization is more fluent than the second could be described syntactically: fronting of the subject is preferred over fronting of the predicative complement. However, the language model might also encode such preferences indirectly: it could prefer *Het* at the start of the sentence over *Zeker* at the start of the sentence. Also, *Het is* and *is zeker* are more frequent than *Zeker is* and *is het* respectively.

[2]The term *feature* in this context should not be confused with the term *feature* in attribute-value grammars, which is sometimes used as a synonym of *attribute*.

Since it is hard to craft complex models while maintaining feature independence, it is strongly preferable to use a feature-based model that does not assume feature independence. One such class of models are *maximum entropy models*. Maximum entropy models have been shown to be effective for fluency ranking by Velldal et al. [2004]. In Velldal [2008], maximum entropy models are compared with support vector machines (SVM). However, it is shown that the best SVM model does not outperform the best maximum entropy model for fluency ranking.

The following sections give a short description of conditional maximum entropy models, roughly following Berger et al. [1996]. First, we look at how feature-value pairs in the training sample are used in the form of constraints during the optimization of the model, making the model mirror the training sample as much as possible. Then we will see how the principle of maximum entropy can be used to pick the model with the least assumptions.

4.3.2 Feature value constraints

If feature-value pairs are used to characterize derivations, we are interested in the distribution of features in fluent and non-fluent sentences. If L is the set of abstract dependency structures in the training sample, $f_i(l,d)$ the value of the feature f_i in derivation d of the abstract dependency structure l, and $\tilde{p}(l,d)$ the joint probability of l and d, then the value of f_i in the training sample is:

$$E_{\tilde{p}}(f_i) = \sum_{l \in L} \sum_{d \in \Omega(l)} \tilde{p}(l,d) f_i(l,d) \tag{4.8}$$

We want to construct a model such that the model generates the training sample. In other words, for a given feature, its expected value should equal its value in the training data,

$$E_p(f_i) = E_{\tilde{p}}(f_i) \tag{4.9}$$

where the expected value of f_i, $E_p(f_i)$, is defined as

$$E_{p(f_i)} = \sum_{l \in L} \sum_{d \in \Omega(l)} \tilde{p}(l) p(d|l) f_i(l,d) \tag{4.10}$$

Note that a conditional model only estimates the probability $p(d|l)$, so $p(d|l)$ is multiplied by the empirical $\tilde{p}(l)$ to obtain the joint probability of l and d (given that $p(l,d) = p(l)p(d|l)$).

During the optimization of the model, the constraint in Equation 4.9 is applied to each feature f_i in the set of all features F.

If $\mathcal{P}$ is the set of all possible models, F the set of features, we can define a restricted set of models $\mathcal{C}$ containing only those models wherein feature constraints are satisfied:

$$\mathcal{C} \equiv \{p \in \mathcal{P} \mid \forall_{f_i \in F}[E_p(f_i) = E_{\tilde{p}}(f_i)]\} \tag{4.11}$$

Section 4.3.6 discusses how the empirical probabilities $\tilde{p}(l)$ and $\tilde{p}(l, d)$ are calculated.

4.3.3 The principle of maximum entropy

Since the set $\mathcal{C}$ normally consists of an infinite number of models, an additional criterion is required to select the best model. This is where the principle of maximum entropy [Jaynes, 1957a,b] becomes useful. This principle states that of the models that are possible given a set of constraints, the model that is the most uniform ought to be selected. The most uniform model is of particular interest, because it does not make any additional assumptions besides those that are implied by the training data.

The uniformity of a model is measured by its entropy - the entropy reaches its maximum in the uniform model. Consequently, of all the models in $\mathcal{C}$, we are interested in the model that maximizes entropy. If $H(p)$ is the entropy of model p, we pick p such that:

$$\underset{p \in \mathcal{C}}{\operatorname{argmax}}\, H(p) \tag{4.12}$$

4.3.4 Parametric form

Picking the model in $\mathcal{C}$ that maximizes entropy is a constrained optimization problem. Such problems can be solved by introducing Lagrange multipliers to transform them into unconstrained optimization problems. Using this transformation we can derive the parametric form of maximum entropy models [Berger et al., 1996]:[3]

$$p(d|l) = \frac{1}{Z(l)} exp \sum_{i=1}^{|F|} \theta_i f_i(l, d) \tag{4.13}$$

Here $\frac{1}{Z(l)}$ is a normalizer over the yield of l to ensure that $\sum_{d \in \Omega(l)} p(d|l) = 1$:

[3] Plank [2011] provides a derivation of the parametric form.

$$Z(l) = \sum_{d \in \Omega(l)} exp \sum_{i=1}^{|F|} \theta_i f_i(l, d) \tag{4.14}$$

The corresponding dual function and the unconstrained dual optimization problem are:

$$\Psi(\theta) = -\sum_{l \in L} \tilde{p}(l) \log Z(l) + \sum_{f_i \in F} \theta_i \tilde{p}(f_i) \tag{4.15}$$

$$\theta* = \underset{\theta}{\operatorname{argmax}}\, \Psi(\theta) \tag{4.16}$$

4.3.5 Maximum likelihood estimation

Optimization of the unconstrained optimization problem can also be seen as maximum likelihood estimation. The log-likelihood of the distribution $\tilde{p}$ as predicted by the model p is:

$$L(p) \equiv \sum_{l \in L} \sum_{d \in \Omega(l)} \tilde{p}(l, d) \log p(d|l) \tag{4.17}$$

It can be shown [Berger et al., 1996] that the dual function $\Psi(\theta)$ is in fact the log-likelihood of the model p.

4.3.6 Empirical probabilities

Calculating $E_p(f_i)$ and $E_{\tilde{p}}(f_i)$ (Section 4.3.2) requires the estimation of $\tilde{p}(l)$ and $\tilde{p}(l, d)$ with respect to the training sample. The methods that have been described in the literature can be divided broadly in two approaches: (1) methods that divide the probability mass of the joint distribution between all derivations, proportionally to a quality metric; and (2) methods that put all the probability mass of the joint distribution on the correct derivation(s). Since the *correct* derivation may not always be available in the training data, the methods that follow the latter approach can be relaxed such that they put the probability mass on the best available derivation(s) according to a quality metric. In the literature, at least three methods are described for estimating $p(l)$ and $p(l, d)$:

1. **Uniform inputs:** Each input is equally probable. Each derivation takes a part of the input probability, proportionally to its quality score [Osborne, 2000]:

$$\tilde{p}(l) = \frac{1}{|L|} \qquad \tilde{p}(l,d) = \tilde{p}(l)\frac{score(l,d)}{\sum_{d' \in \Omega(l)} score(l,d')} \tag{4.18}$$

2. **Weighted inputs:** The probability of an input is proportional to the sum of the score of its derivations [van Noord and Malouf, 2005]:

$$\tilde{p}(l) = \frac{\sum_{d \in \Omega(l)} score(l,d)}{\sum_{l' \in L} \sum_{d \in \Omega(l')} score(l',d)}$$
$$\tilde{p}(l,d) = \tilde{p}(l)\frac{score(l,d)}{\sum_{d\ \in \Omega(l)} score(l,d')} \tag{4.19}$$

   This is a variation of the first method. The motivation of [van Noord and Malouf, 2005] is to give more weight to inputs with higher quality derivations.

3. **Uniform inputs with binary events:** Each input is equally probable. The input probability is shared between the highest-scoring derivations [de Kok et al., 2011]. Let $j(l,d)$ be a function which returns 1 if the derivation d is the best-scoring derivation of l, and 0 if it is not. Then:

$$\tilde{p}(l) = \frac{1}{|L|} \qquad \tilde{p}(l,d) = \tilde{p}(l)\frac{j(l,d)}{\Sigma_{d' \in \Omega(l)} j(l,d')} \tag{4.20}$$

Another natural variation on these three methods is:

4. **Weighted inputs with binary events:** The probability of an input is weighted by the number of derivations with the highest score. Let $j(l,d)$ again be a function that gives binary scores to derivations, then:

$$\tilde{p}(l) = \frac{\sum_{d \in \Omega(l)} j(l,d)}{\sum_{l' \in L} \sum_{d \in \Omega(l)} j(l,d)} \tag{4.21}$$
$$\tilde{p}(l,d) = \tilde{p}(l)\frac{j(l,d)}{\Sigma_{d' \in \Omega(l)} j(l,d')} \tag{4.22}$$

A different approach, that we did not evaluate, is to optimize directly over the set of correct (or best) derivations, as proposed by Riezler et al. [2002]. In that work, a model is trained to maximize the function

$$\sum_{l \in L} \log(p(\forall_{d \in \Omega(l)} j(l,d) \equiv 1 | l)) \tag{4.23}$$

This is similar to the third method above, in that it assumes a uniform distribution of inputs and puts all the probability on the correct derivations.

In Section 4.7.3 we evaluate the practical impact of these four methods for estimating empirical probabilities.

4.3.7 Parameter estimation

To find the model that maximizes the log-likelihood with respect to the training data, an optimization method is required. Malouf [2002] provides a survey of six commonly used methods, applied to natural language processing tasks: *Generalized Iterative Scaling* [Darroch and Ratcliff, 1972], *Improved Iterative Scaling* [Berger et al., 1996, Della Pietra et al., 1997], *steepest ascent* [Cauchy, 1847], two variants of *conjugate gradient* [Fletcher and Reeves, 1964, Polak and Ribiere, 1969], and *limited memory variable metric* [Benson and Moré, 2001]. Malouf finds that the limited memory variable metric outperforms the other methods by a substantial margin.

In this work, we use the L-BFGS optimization method [Nocedal, 1980, Liu and Nocedal, 1989]. L-BFGS, like the limited memory variable metric, belongs to the family of limited-memory quasi-Newton methods. Since L-BFGS cannot apply ℓ_1 regularization (Chapter 6), we use an extension of L-BFGS named OrthantWise Limited-memory Quasi-Newton (OWL-QN) that also supports ℓ_1 regularization [Andrew and Gao, 2007].

L-BFGS attempts to solve the minimization problem $F(\theta)$, and requires that $F(\theta)$ and its gradients $G(\theta)$ are computable. As discussed in Section 4.3.5, the model in $\mathcal{C}$ that maximizes entropy is a model in the parametric family $p(d|l)$ that maximizes the likelihood of the training sample. Consequently, when using a minimization algorithm, we have to minimize the negative log likelihood:

$$F(\theta) = -L_{\tilde{p}}(p) \tag{4.24}$$

L-BFGS also requires the gradient of the objective function. The gradient of $L(p)$ with respect to the parameter θ_i is [Malouf, 2002]:

$$G(\theta_i) = E_{\tilde{p}}(f_i) - E_p(f_i) \tag{4.25}$$

4.3.8 Regularization

As discussed in Section 4.3.5, optimization of the parametric maximum entropy model can be seen as maximum likelihood estimation. As other maximum likelihood estimators, parameter estimation of maximum entropy models is prone to overfitting to the training data [Chen and Goodman, 1999]. This overfitting results in extreme positive or negative weights, that negatively impact the performance of the model.

Regularization is often applied during parameter estimation to avoid overfitting. If $F(\theta)$ is the objective function that is minimized during training, a regularizer $\omega_q(\theta)$ is added as a penalty for extreme weights [Tibshirani, 1996]:

$$C(\theta) = F(\theta) + \omega_q(\theta) \tag{4.26}$$

The regularizer has the following form, where $q \geq 0$:

$$\omega_q(\theta) = \lambda \sum_{i=1}^{n} |\theta_i|^q \tag{4.27}$$

Setting $q = 2$ in the regularizer gives a so-called ℓ_2 regularizer:

$$\omega_2(\theta) = \frac{1}{2\sigma^2} \sum_{i=1}^{F} \theta_i^2 \tag{4.28}$$

This regularizer amounts to imposing a Guassian prior distribution on the parameters with a mean of zero and a variance of σ^2 [Chen and Goodman, 1999]. Since the Gaussian distribution only has very little of its probability mass in the tails, the ℓ_2 regularizer protects against extreme parameters.

Regularization with $q = 1$ will be discussed in Chapter 6.

4.3.9 Application

In order to estimate the parameters of the model, we use maximum entropy modeling in the parametric form. However, in the application of maximum entropy models a simpler equation is commonly used (e.g. Geman and Johnson [2002] and Velldal and Oepen [2005]). Observe that the normalizer $Z(l)$ is constant for each $d \in \Omega(l)$, and also $a > b \iff e^a > e^b$. So, if we are only interested in the ordering of derivations, rather than their actual probabilities, we can use a simpler scoring function:

$$score(d|l) = \sum_{i=1}^{|F|} \theta_i f_i(l, d) \tag{4.29}$$

4.4 Features

Since a maximum entropy model ranks realizations based on feature values, we have to settle on a set of features that can adequately describe characteristics of derivations of fluent sentences. Features for fluency ranking can be divided into two classes [de Kok and van Noord, 2010]: (1) *Output features* that describe aspects of the produced sentence, such as the probability of a sentence according to a language model. (2) *Construction features* that describe aspects of the derivation that constructed the sentence, such as how often a particular grammar rule was used in the derivation.

While features can be hand-crafted, they are usually extracted automatically by applying a template to the training or evaluation data. Such a template can be seen as a function that takes a derivation as its input and results in a set of features with corresponding values. As in other fluency rankers [Velldal et al., 2004, Velldal and Oepen, 2006, Cahill et al., 2007], (nearly) all construction features were inherited from an existing parse disambiguation component; in our case, the parse disambiguation component of the Alpino parser [van Noord, 2006].

One important difference between our ranker and the ranker described by Velldal et al. [2004] and Velldal and Oepen [2006] is abstraction level of the construction features that are produced. The former works use features that enumerate local derivation subtrees. However, as in Cahill et al. [2007], we complement local derivation subtrees with deeper and more global syntactic features from the attribute-value structures in the derivation. This has the effect that we can improve the model by making informed decisions about global syntactic phenomena, such as subject fronting, ordering in the middle field, and long distance dependencies. This difference is not accidental, since Velldal's realizer was developed for English, while our realizer and that of Cahill et al. [2007] were developed for other Germanic languages, which have a freer word order. While such features provide more information about the fluency of a sentence, they reduce the effectiveness of best-N unpacking (Section 4.5).

The performance of a fluency ranking model can be improved considerably if generation-specific output features are added, as shown by Velldal et al. [2004]. For this reason, we also include surface-oriented features in our model.

In this section, we first describe the features that are used in the Alpino fluency ranking model. We will then give a short comparison with the set of features described in detail in Velldal and Oepen [2006].

Language models A language model over word trigrams is used as an auxiliary distribution [Johnson and Riezler, 2000] in the model. The advantage of

using an auxiliary distribution over using trigrams directly as features, is that in the case of an auxiliary distribution a large (unannotated) training corpus can be used. If trigrams were integrated as separate features with their value being the frequency of that trigram within a sentence, the weights of individual trigram features would have to be estimated within the maximum entropy framework. Consequently, the training data would would be restricted to the training data for the maximum entropy model.

In addition to the word trigrams language model, an auxiliary distribution of Alpino part-of-speech tag trigrams is added as a feature. Since part-of-speech tags provide an abstraction over words and the number of part-of-speech tags is finite, a tag trigram distribution can ofter improve the estimation of fluency in the presence of unknown words. Consider the following realizations:

(25) a. Barack Obama gebruikt een tabletcomputer
Barack Obama uses a tablet-computer
b. Een tabletcomputer gebruikt Barack Obama

If 'Barack Obama' and 'tabletcomputer' were unseen in the training sample, it would be possible that a word trigram model has no (informed) preference for either realization. Now suppose that 'Obama' was tagged as a proper noun and 'tabletcomputer' as a noun. Since the sequence *proper name - finite verb - article* is more likely than *noun - finite verb - proper name*, the tag trigram model has a preference for the first realization.

Again, the use of an auxiliary distribution gives us access to far more training data. For instance, to train the tag trigram model, we used a large corpus that was automatically annotated using Alpino's parser.

We will now give a short example of how these features are used, and combined in the model. Consider the following sentence

(26) a. De optische astronomie maakt gebruik van zichtbaar licht.
The optical astronomy makes use of visible light.

Alpino assigns the following part-of-speech tags in Table 4.2.

We use the word trigram model to estimate the probability of the sentence and the tag trigram model to estimate the probability of the trigram sequence *determiner(de)..punct(punt)*. The logarithms of these probabilities then become the values of the *word_trigram_distribution* and *tag_trigram_distribution* features, respectively. As shown in Table 4.3, these values are multiplied by the weights of the features (Section 4.3.9) that were found during the training of the maximum entropy model. If we had no other features, the score of this realization would be the sum of the weighted scores.

Word	**Tag**
De	*determiner(de)*
optische	*adjective(e)*
astronomie	*noun(de,sg,[])*
maakt	*verb(sg3)*
gebruik	*particle*
van	*preposition(van)*
zichtbaar	*adjective(no_e(adv))*
licht	*noun(het,sg,[])*
.	*punct(punt)*

Table 4.2: Part-of-speech tags assigned to the sentence *De optische astronomie maakt gebruik van zichtbaar licht.*

Feature	**Weight (θ_i)**	**Value (f_i)**	$\theta_i \cdot f_i$
word_trigram_distribution	0.0158	-62.70	-0.9907
tag_trigram_distribution	0.0115	-24.05	-0.2766
Score ($\sum_{i=1}^{n} \theta_i f_i$)			-1.2673

Table 4.3: Example output feature values for the sentence *de optische astronomie maakt gebruik van zichtbaar licht* 'the optical astronomy makes use of visible light'. Each value is multiplied by the feature weight that was found during training of the maximum entropy model. The score of the realization is obtained by summing the weighted feature values.

As described in Section 4.2.2, both models use linear interpolation smoothing to handle unknown trigrams. Laplace smoothing [Lidstone, 1920] is applied for estimating the probability of unknown words.

Grammar rule identifiers Since some grammar rules are more likely to be used in fluent realizations than others, the derivation tree features create a distribution of grammar rule applications. Two types of features are used: (1) a *rule* feature simply records how often a particular grammar rule is used in a derivation, (2) a *rule_in_context* feature records how often a rule is used with its parent. For instance, consider the small tree fragment shown in Figure 4.1. The *rule* and *rule_in_context* features that have a non-zero value/frequency in this fragment are enumerated in Table 4.4. The *rule_in_context* feature is represented by a term that has three arguments: (1) the identifier of the rule that was used to construct the parent; (2) the index of the parent slot that the

rule filled, starting at 1; and (3) the rule identifier.

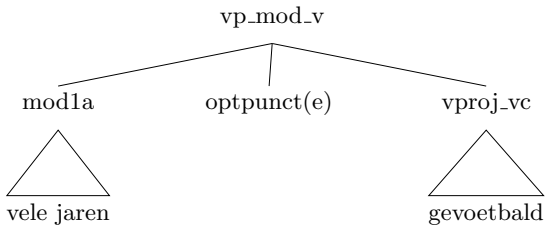


Figure 4.1: A small tree fragment dominating the phrase *vele jaren gevoetbald* 'many years played-soccer'.

Feature	**Value**
rule(vp_mod_v)	1
rule(mod1a)	1
rule(optpunct(e))	1
rule(vproj_vc)	1
rule_in_context(vp_mod_v,1,mod1a)	1
rule_in_context(vp_mod_v,2,optpunct(e))	1
rule_in_context(vp_mod_v,2,vproj_vc)	1

Table 4.4: Rule features extracted from the tree fragment in Figure 4.1.

The *rule* and *rule_in_context* features are similar to the local derivation subtree features used by Velldal et al. [2004] and Velldal and Oepen [2006]

Syntactic features The fluency ranking model also uses more abstract syntactic features that originated in the parse disambiguation model. These features have a non-zero value if a certain syntactic phenomenon occurs, by using the relevant information in the attribute-value structure. For instance, one such feature, *syntactic(subj_np_topic)*, records the presence of a topicalized NP subject. This feature 'fires' when a category in the derivation unifies with the attribute-value structure in Figure 4.2. This structure states that if a category has the type *sv1* (a verb-first structure) and takes a nominative NP as its slash-value, then it is a fragment with a topicalized NP subject.

Another syntactic feature template records orderings in the middle field. As discussed in Section 4.1.1, Dutch provides some freeness in ordering the middle field. For instance, for the examples given in that section,

$$_{sv1}\left[\text{SLASH} \left\langle {}_{np}\left[\text{CASE} \quad nom\right]\right\rangle\right]$$

Figure 4.2: Attribute-value structure that is used for recording a topicalized NP subject.

Realization	Middle field features
1	*middle_field(np(dat,pron(nwh)),mcat_adv),* *middle_field(np(dat,pron(nwh)),np(acc,noun)),* *middle_field(mcat_adv,np(acc,noun))*
2	*middle_field(np(dat,pron(nwh)),np(acc,noun)),* *middle_field(np(dat,pron(nwh)),mcat_adv),* *middle_field(np(acc,noun),mcat_adv)*
3	*middle_field(mcat_adv,np(dat,pron(nwh))),* *middle_field(mcat_adv,np(acc,noun)),* *middle_field(np(dat,pron(nwh)),np(acc,noun))*

Table 4.5: Examples of *mf* features that capture ordering in the middle field in three realizations.

1. De twee giganten schudden **elkaar glimlachend de hand**.
2. De twee giganten schudden **elkaar de hand glimlachend**.
3. De twee giganten schudden **glimlachend elkaar de hand**.

the fluency ranker extracts the *middle_field* features in Table 4.5. We can see that each *middle_field* feature describes the order of a pair of categories in the middle field. Where necessary, the category is augmented with more information. For instance, the case of the noun phrases (accusative or dative) is included, as well as a boolean to distinguish pronouns.

The *mf* feature is another example where deep and global syntactic knowledge is required. We need a verb phrase to collect and provide its middle field; we need detailed information about the categories in the middle field, such as their case; and we need all the pairwise categories in the middle field to get a proper estimation of the fluency of the middle field.

We discussed two syntactic phenomena that are described by features that are more abstract than local derivation subtrees. Other syntactic phenomena for which features are constructed are:

- Other types of fronting (NP/non-NP, subject/non-subject)

- The presence or absence of long-distance dependencies.
- Parallelism of conjuncts in coordinations.

'Velldal features' Initially, we also used the feature templates that were described by Velldal and Oepen [2006], which consist of construction features capturing:

1. Local derivation subtrees with optional grand-parenting, with a maximum of three parents.
2. Local derivation subtrees with back-off and optional grand-parenting, with a maximum of three parents.
3. Binned word domination frequencies of the daughters of a node.
4. Binned standard deviation of word domination of node daughters.

However, the use of these features did not improve performance. Perhaps this is not surprising, since templates (1) and (2) overlap with the *rule* and *rule_in_context* features described above. Grand-parenting adds more context, at the cost of data sparseness. Global preferences are captured more effectively using abstract syntactic features.

Velldal and Oepen [2006] also describes two output templates:

1. N-grams of lexical types ($n \leq 4$).
2. N-grams of lexical types ($n \leq 4$), where the word corresponding to the last type is also included.

Again, the addition of these templates did not give an improvement. This is expected, since these templates overlap with the trigram models. Additionally, since the trigram models are used as an auxiliary distribution, they have access to a much larger amount of training data.

4.5 N-best unpacking

As discussed in Section 3.4, edges that share the same attribute-value structure are stored only once with their derivation histories, to reduce memory use and increase performance. The derivations can be unpacked when required, for instance to apply the fluency ranking model or to extract realizations from the

derivations. Unpacking can be relatively time consuming — every attribute-value structure in a derivation is reconstructed. If we have access to information with respect to fluency of a sentence, it can be used to extract the most fluent realizations.

Such selective unpacking has been in wide use in parsing the last ten years. For instance, Miyao and Tsujii [2002], Geman and Johnson [2002], and Clark and Curran [2003] propose methods for unpacking the best derivations in parsing using a dynamic programming algorithm. However, as van Noord and Malouf [2005] points out, such approaches require monotonicity, in that a subtree that has the highest probability locally should also be the most probable globally. Exactly this assumption of monotonicity is not warranted in our case, since many of our (syntactic) features are global.

As a solution to this problem, van Noord and Malouf [2005] propose to unpack derivations using a beam instead. In this approach, the derivations of a given item are unpacked, but only the N most probable derivations are retained. Since this method is applied recursively, only a small subset of derivations is constructed. The use of the N best realizations, rather than only the best, increases the likelihood that the best parse is included. We follow this proposal by using beam search to unpack realizations. However, in our case even local information may not yet be available in an unpacked subtree. As discussed in Section 2.4.4 some dependents, such as modifiers, are not immediately unified in the attribute-value structure of a head. For instance, a head might collect modifiers that are eventually transferred to another head. Given that such local information is not finalized yet, purging derivations for such items is premature. So, instead, we apply beam search only for items that are maximal projections to assure that such local choices are settled. Algorithm 8 extends the pseudo-code in Algorithm 2 with beam search.

4.6 Evaluation methodology

4.6.1 Treebanks

Throughout this thesis, two treebanks are used for training and evaluating models. The newspaper part of the Eindhoven corpus,[4] named *cdb* (*corpus dagbladen*), is used for training models. This corpus consists of 7136 sentences of 1 to 74 tokens, with an average length of 19.73 tokens. Syntactic annotations are part of the Alpino Treebank.[5]

[4] `http://www.inl.nl/tst-centrale/nl/producten/corpora/eindhoven-corpus/6-27`
[5] `http://www.let.rug.nl/vannoord/trees/`

Algorithm 8 Algorithm for unpacking the N best derivations for a given item. The search beam is only applied at maximal projections, to ensure e.g. that a head was chosen for modifiers.

```
function UNPACK(id)
    histories ← RETRIEVE_HISTORIES(id)
    unpacked ← []
    for all his ∈ histories do
        if IS_RULE_HISTORY(his) then
            unpacked′ ← UNPACK_RULE(his)
            if MAXIMAL_PROJECTION(his) then
                APPEND(unpacked,N_BEST(unpacked′))
            else
                APPEND(unpacked,unpacked′)
            end if
        else
            APPEND(unpacked,UNPACK_LEX(his))
        end if
    end for
    return unpacked
end function
```

For evaluation, we use the *PPH* corpus. This corpus consists of sentences from the Trouw 2001 newspaper, that are originally from the Twente Nieuws Corpus (TwNC).[6] The *PPH* corpus consists of 2267 sentences of 1 to 63 tokens, with an average length of 16.43 tokens. Syntactic annotations are part of the Lassy Small treebank.[7]

During the development of the fluency ranking component, we used a different corpus for training and evaluation. For development purposes, we created a corpus of 20,000 sentences of 5 to 25 tokens that were randomly selected from the Dutch Wikipedia of August 2008.[8] Since no syntactic annotations were available for this corpus, we used the best parse constructed by the Alpino parser to find the corresponding abstract dependency structure. This provided access to a relatively large test set of training and testing at the cost of annotation errors. Since we do not use this corpus in the evaluation of the fluency ranking component described in this thesis, this has no effect on its evaluation.

The trigram language language models are trained using the Twente Nieuws-

[6] http://wwwhome.cs.utwente.nl/~druid/TwNC/TwNC-main.html
[7] http://www.inl.nl/tst-centrale/nl/producten/corpora/lassy-klein-corpus/6-66
[8] http://ilps.science.uva.nl/WikiXML/

corpus,[9] with the exception of the Trouw 2001 newspaper (approximately 110 million words). The training data of the tag trigram model is generated by applying parsing the data and extracting the part-of-speech tags.

4.6.2 Preparation for fluency ranking

Each treebank consists of sentences with annotations in the form of dependency structures. To train and evaluate a fluency ranking model, we need access to the derivation of that sentence. We could generate directly from the dependency structure to obtain the corresponding derivations. However, it is not always possible to construct a given dependency structure using the grammar. Conversely, it may not always be possible to generate from the gold standard dependency structures. For this reason, the derivations that are used for training and evaluating the fluency ranking model are created in two steps:

- Each sentence in the treebank is parsed using Alpino. The parse that has the dependency structure with the highest correspondence to the dependency structure in the treebank is selected.
- The best dependency structure that was found in the previous step is used as the input to the generator. The derivation of each realization is stored, along with a quality score that will be discussed in the next section.

In this procedure for preparing training and testing data, we assume that the sentence that occurs in the treebank is the most fluent realization of the corresponding dependency structure. This may not be true, or there may be multiple realizations that are equally fluent. Nonetheless, we expect that writers (in this case newspaper journalists and editors) attempt to express their thoughts as fluently as possible for the given domain.

Another assumption is that the dependency structure in the best parse of Alpino is similar enough to the gold standard dependency structure that the gold standard sentence can be generated. We believe that this is a reasonable approach: if the correct dependency structure is among the parses it will be used. If it is not, the gold standard dependency structure can probably not be derived using the grammar, conversely it is not possible to generate from that dependency structure. In such cases, we are more interested in obtaining training instances that are nearly correct, than in having nothing at all.

Producing all parses and realizations is expensive in time and space for complex inputs. For these reasons, the amount of memory that can be used

[9] `http://wwwhome.cs.utwente.nl/~druid/TwNC/TwNC-main.html`

is set to 10GB and parsing and generation time is limited to one minute on a Xeon E5410 2.33 GHz core. With these limits, we could extract 4187 training instances from the *cdb* treebank and 1622 evaluation instances from the *PPH* treebank. Table 4.6 provides more information about both data sets.

Corpus	Inputs	Avg. realizations	Avg. realization length
cdb	4187	414.6	18.0
PPH	1622	330.5	17.1

Table 4.6: Characteristics of the *cdb* training set and the *PPH* evaluation set.

4.6.3 Realization quality

During training and evaluation, a quality estimation of each realization is required. In training, this estimation is used to select the best realization(s) when the correct realization(s) are not available (methods (3) and (4) in Section 4.3.6) or to estimate the empirical probability $\tilde{p}(x, y)$ of a realization (methods (1) and (2) in Section 4.3.6). In the evaluation, the quality score of the best realization according to the fluency ranker is used to evaluate the ranker. Since quality scores are an important factor during training and evaluation, it is important to select a good scoring metric.

One of the simplest metrics for the evaluation of a fluency-ranker is the *exact match accuracy*. This is simply the percentage of evaluation instances where a fluency ranker chose the derivation with the correct sentence. It is possible that the generator could not construct a derivation that contains the correct sentence. In such a case, the fluency ranker would be penalized for a deficiency in another component by the best match accuracy. The *best match accuracy* is the percentage of evaluation instances where the fluency ranker chose the derivation with the best available realization, according to some metric.

An important disadvantage of exact and best match accuracies, is that they penalize equally the choice of a realization that is almost correct and a realization that is outright wrong. It is better to have a metric that can distinguish the choice of realizations in a more fine-grained manner.

Cahill [2009] examines such fine-grained metrics (BLEU, ROUGE-L, GTM, SED, WER, and TER) in the context of a sentence generation system for German. The metrics are compared in their correlation with human judgements. Overall, the General Text Matcher (GTM) [Melamed et al., 2003] was found to be the metric with the highest correlation to human judgments. Since German is fairly similar to Dutch and the system used by Cahill [2009] uses a grammar

formalism that is comparable to stochastic attribute-value grammar, we believe that the outcome is applicable for our experiments.

The GTM method can best be explained by considering the relationship between a realization and the treebank sentence in a bitext grid. Figure 4.3 shows such a grid for the imaginary realization *C B A I C D E* and the reference sentence *A B C D E F B A I C*. The tokens that match between the strings are marked by placing a bullet in the corresponding cell, this is called a *hit*. To avoid double counting, the GTM method uses the concept of a *matching*. A matching is a subset of hits in the grid, such that no hits are in the same row or column. A *maximum matching* is a matching of the maximum possible size for that bitext. The size of the maximum matching of the bitext in Figure 4.3 is 7.

The use of maximum matchings allows one to find the configuration where as many tokens match as possible, while not double counting a token. However, in generation, we not only want a realization to contain the tokens of the reference sentence, it should also have those tokens in the same order. Sequences of tokens in the realization that match with those in the reference text can easily be detected in the bitext grid. They are characterized by diagonally adjacent hits that run parallel to the main diagonal. Melamed et al. [2003] refers to such diagonals as *runs*. Since longer common substrings are preferred over shorter ones, GTM puts a heavier weight on longer runs, by calculating the match size as follows:

$$size(M) = \sqrt{\sum_{r \in M} length(r)^2} \tag{4.30}$$

where r is a run in the matching M. Longer runs get a heavier weight due to squaring. The score is normalized using a square root, making the maximum possible score equal to length of the shorter of the realization and the reference.

It should be emphasized that only those hits in a run should be counted that are in the maximum matching. So, for instance, in Figure 4.3 there are runs (from left to right) of length 1, 2, and 4. In this case the resulting size is $\sqrt{1^2 + 2^2 + 4^2}$.

The match size can be used to calculate the precision, recall, and f-score. The General Text Matcher score is then the f-score. If C is the candidate realization and R the reference sentence, then:

$$precision = \frac{size(mm(C,R))}{|C|} \quad recall = \frac{size(mm(C,R))}{|R|} \tag{4.31}$$

where $mm(A, B)$ is the maximum match defined above.

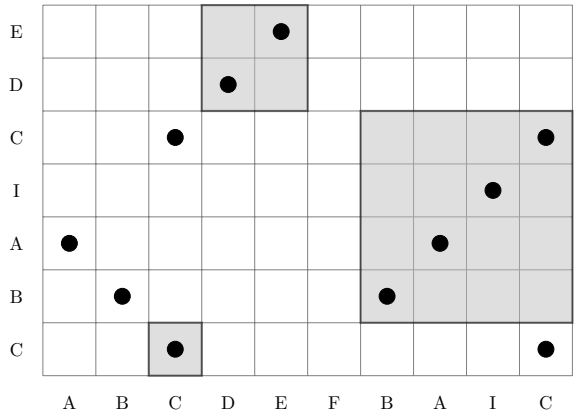


Figure 4.3: The maximum matching of a realization and a reference sentence. This example is from Melamed et al. [2003].

4.6.4 Statistical significance

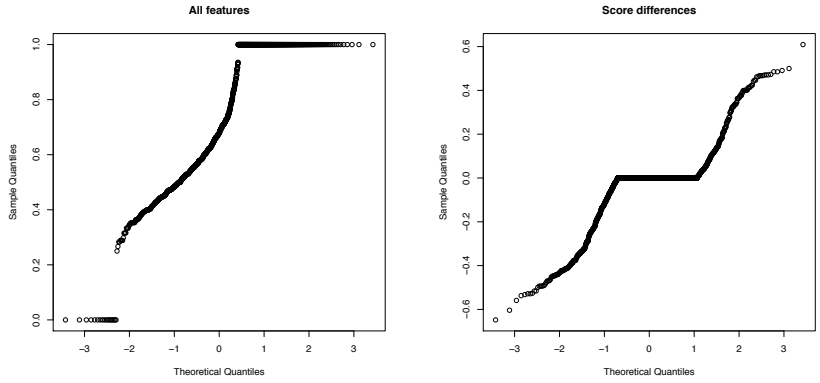
Quality scores such as GTM scores in fluency ranking or CA scores in parse disambiguation do not have a simple distribution. Usually, there are two clusters of scores, one of which is nearly normally distributed and another which has scores that are exactly 1. This can be observed in the normal QQ-plot of the GTM scores of the model with all features in Figure 4.4(a). Since the evaluation scores of all our models exhibit this property, the differences between scores of models usually have a large number of zeros, as seen in Figure 4.4(b).

Since the scores do not have a distribution that is appropriate for a parametric test, we use the non-parametric *approximate randomization test* [Noreen, 1989]. Cohen [1995] provides a practical introduction to randomization tests. In this section, we give a more formal description of *exact* and *approximate pair-wise randomization* tests.

Exact pair-wise randomization test

The intuition behind the exact pair-wise randomization test is fairly simple. Suppose that we have a pseudo-statistic t that compares two samples. The null hypothesis in our case is that two models perform equally. If the null hypothesis is true, we could simply swap pairs of scores from both samples without a drastic effect on the statistic t.

Suppose that S_m is the sample of evaluation scores of model m and S_n



(a) (b)

Figure 4.4: Normal Q-Q plots for (a) the GTM scores of the model with all features and (b) the differences in GTM scores between the model with all features and the model with only the ngram auxiliary distributions.

that of n. If $t(S_m, S_n)$ calculates the pseudo-statistic for S_m and S_n, we can compute the probability $p(t(S_m, S_n))$ that such a score would occur. First, we create all possible permutations of pairs in the samples S_m and S_n. If $|S_m|$ is the size of the samples, then the number of possible pair-wise permutations is $2^{|S_m|}$.

For a one-tailed test, we can then partition the sample permutations such that all sample permutations $(S_{m'}, S_{n'})$ for which $t(S_{m'}, S_{n'}) \geq t(S_m, S_n)$ applies belong to P, while the rest belongs to Q. We can then estimate the probability that the value of the test statistic, or a larger value, occurs by chance:

$$p = \frac{|P| + 1}{|P \cup Q| + 1} \tag{4.32}$$

For a two-tailed test, we define the sets P and R: if $t(S_{m'}, S_{n'}) \geq t(S_m, S_n)$ then the permutation is in P, and if $t(S_{m'}, S_{n'}) \leq t(S_m, S_n)$ then the permutation is in R. The p-value is then:

$$p = \frac{\min(|P|, |R|) + 1}{|P \cup R| + 1} \tag{4.33}$$

where the desired significance level should be divided by two (to account for both tails).

Approximate pair-wise randomization test

Unfortunately, the exact randomization test is computationally intractable for most larger sample pairs. The *approximate pair-wise randomization test* applies the same methodology to a (relatively) small sample of all possible permutations. Algorithm 9 shows the pseudo-code of this test (wherein N is the size of the sample of permutations).

Algorithm 9 Approximate randomization test

```
leftCount ← 0
rightCount ← 0
for i = 1...N do
    (S_m', S_n') ← ([], [])
    for all pair ∈ (S_m, S_n) do
        (x',y') ← shuffle(pair)
        append(S_m', x')
        append(S_n', y')
    end for
    if t(S_m', S_n') ≥ t(S_m, S_n) then
        rightCount ← rightCount + 1
    end if
    if t(S_m', S_n') ≤ t(S_m, S_n) then
        leftCount ← leftCount + 1
    end if
end for
p ← (min(leftCount, rightCount) + 1) / (N + 1)
```

In this thesis, we apply approximate randomization tests with the mean difference of two samples as the test statistic t:

$$\text{mean\_diff}(S_m, S_n) = \frac{S_m - S_n}{|S_m|} \tag{4.34}$$

A note on some previous descriptions

Pair-wise approximate randomization tests have been used before to test significance of models in related tasks, such as parse disambiguation [Riezler and

Maxwell, 2005, Cahill et al., 2008, Plank, 2011]. We would like to point out that the algorithms described in these works do not explicitly check whether $t(S_m, S_n)$ is in the left tail or right tail. Instead, they take the absolute value of the pseudo-statistic. This has the effect that if the original test statistic had the value β, that randomizations with a test statistic value that is at least as extreme as β and $-\beta$ are counted together. This overestimates the tailed p-value by a factor approaching two. But since these works do not divide the test p-value by two to obtain the critical p-value for each tail, their algorithm accepts or rejects the null hypothesis correctly for a two-tailed test.

The problem, however, is that this approach assumes that the pseudo-statistic has a mean $\mu \approx 0$ for repeated randomizations. Their algorithms will give incorrect results when this assumption does not hold. An example of a pseudo-statistic where $\mu \not\approx 0$ for repeated randomizations, is the ratio of variance [Cohen, 1995] of two samples:

$$t_{var}(S_m, S_n) = \frac{var(S_m)}{var(S_n)} \tag{4.35}$$

This statistic is used to test whether two models have equally consistent performance.

Correction for comparisons with repeated tests

The performance of multiple models is often compared pairwise. If we check the significance at a particular level, probability of an erroneous rejection of the null-hypothesis increases. If α_c is the error per-comparison and k the number of pairwise comparisons, then the experiment-wise error is $\alpha_e = 1 - (1 - \alpha_c)^k$ [Cohen, 1995]. To test significance that is experiment-wise at the 99% level, we apply the Šidák correction [Šidák, 1967]. This correction calculates the allowable per-comparison error as: $\alpha_c = 1 - (1 - \alpha_e)^{\frac{1}{k}}$. The Šidák test differs from the more widely-used Bonferroni test [Dunn, 1961] in that it assumes that individual tests are independent, but gives a stronger bound.

4.6.5 Training of the models

Language models

The language models used in our system are trained using deleted interpolation (Section 4.2.2). Since each model is very large, we compress the models by storing them in tuple automata [Daciuk and van Noord, 2004].

For our experiments with modified Kneser-Ney smoothing, we used MITLM[10]

[10] `http://code.google.com/p/mitlm/`

to train the model. The Kneser-Ney model was compressed and applied using KenLM.[11]

In both models, we discard words that occur fewer than eight times in the training sample, and use them to model the distribution of unknown words.

Maximum entropy models

The maximum entropy models are trained by using the *cdb* corpus as the training set. Each dependency structure corresponds to a training input, each realization of that dependency structure is an event. A dependency structure can have an enormous number of realizations. However, it turns out that parameter estimation can be performed reliably by using an informative sample [Osborne, 2000]. We make such a sample by randomly selecting 100 realizations if $|\Omega(x)| > 100$ for a particular $x \in X$.

We train each model using the TinyEst[12] maximum entropy parameter estimator, that was specifically developed for use in the Alpino system. The experiments that are described in this chapter were conducted using TinyEst with a ℓ_2 prior where $\sigma^2 = 1000$. This provides smoothing to prevent overfitting of the model. Chapter 6 describes further experiments with the ℓ_1 prior and feature selection techniques.

4.6.6 Evaluation

The fluency ranking models are evaluated by applying the rankers to the *PPH* corpus. For each abstract dependency structure in the evaluation data, we let a ranker choose the best realization. Table 4.7 shows an example of such a ranking decision. The table contains the realizations of a dependency structure obtained by parsing the phrase *de geletterdheid van de volledige bevolking wordt geschat op 36%* (*the literacy of the full population is estimated to be 36%*), along with their GTM scores, and the fluency scores assigned by a fluency ranking model (Section 4.3.9). Here, the fluency ranking model will choose the first realization, since it has the highest score. This realization is also the best realization, since it has the highest GTM score.

For each ranker, we calculate the average of the GTM scores of the realizations that were chosen by that ranker to determine its performance. We also report the *best match* accuracy. The realization with the highest GTM score is considered to be the best realization.

[11] `http://kheafield.com/code/kenlm/`
[12] `http://github.com/danieldk/tinyest`

Realization	GTM score	Ranker score
de geletterdheid van de volledige bevolking wordt geschat op 36%	**1.00**	**-1.20**
de geletterdheid van de volledige bevolking wordt op 36% geschat	0.74	-1.56
op 36% wordt de geletterdheid van de volledige bevolking geschat	0.65	-1.66
op 36% wordt de geletterdheid geschat van de volledige bevolking	0.51	-1.81

Table 4.7: A fluency evaluation example for a dependency structure with four realizations. In this case, the ranker picks the most fluent realization.

4.7 Results

4.7.1 Language models

Section 4.2.2 provided a description of the n-gram language model that is used in the Alpino fluency ranker. We evaluate the effectiveness of three variants of the language model described in that section, as well as the widely used modified Kneser-Ney method.

Table 4.8 shows the performance of these four word trigram language models. As we can see, there is quite a discrepancy in the performance of the three models that use linear interpolation. The first model uses only Laplace smoothing for estimating the probability of unknown words, the second model also attempts to model contextual probabilities by replacing an unknown word by a generic unknown word marker, and the third model also uses this generic marker when an unknown word occurs in the context of a known word. Perhaps surprisingly, among the three models the works the best. One possible explanation would be that the more sophisticated models attribute too much probability to unknown words.

The differences between the best linear interpolation model and Modified Kneser-Ney smoothing are less profound: linear interpolation smoothing achieves a higher best match accuracy, Kneser-Ney smoothing, on the other hand, fares better in the evaluation using the GTM method.

Since the differences between the best model using linear interpolation smoothing and that using modified Kneser-Ney smoothing are not conclusive, and linear interpolation smoothing had previously been used in our system [de Kok and van Noord, 2010], we continue to use n-gram models with linear

Model	Best match (%)	GTM
Random	17.64	0.5519
Linear interpolation	**41.52**	0.6864
Linear interpolation (generic marker)	38.68	0.6761
Linear interpolation (unknowns in context)	38.19	0.6743
Modified Kneser-Ney	41.15	**0.6867**

Table 4.8: Performance of the word n-gram language model using linear interpolation smoothing with deleted interpolation and back-off smoothing using the modified Kneser-Ney method [Chen and Goodman, 1999]. Linear interpolation smoothing is used in three variants: the first only uses Laplace smoothing to estimate the probability of an unknown word, the second integrates contextual probabilities by replacing the unknown word by a generic unknown word tag, the third also replaces the word by a generic unknown word tag when estimating the probability of a known word.

interpolation smoothing in other experiments in this chapter.

We also experimented with different language models for tag trigrams, but the differences in accuracy are much smaller. This is expected, since the tag set is a closed and much smaller class.

4.7.2 Fluency rankers

We have evaluated three surface models: word trigrams, tag trigrams, and a maximum entropy model that uses the n-gram models as the only features. In the evaluation of the model of structural information, we first evaluate a model that only uses structural features. Then, we combine both surface and structural features. The results are shown in Table 4.9. To show the range of possible scores, we also give the performance of random selection and the oracle, which always selects the best realization.

Of the n-gram models, the word n-gram model outperforms the tag ngram model (significant at $p < 0.001$). Training a maximum entropy model using both models as auxiliary distributions improved performance over the word n-gram model, although not significantly. The syntactic features by themselves provide a fairly weak model - it only performs insignificantly better than the tag n-gram model and is outperformed significantly by the word n-gram model. However, adding syntactic features to a model with the n-gram models as auxiliary distributions (*all features*) improved performance significantly when comparing to the model using the n-gram distributions.

The best match accuracies for each model are also shown in Table 4.9.

Model	**Best match (%)**	**GTM**
Random	17.64	0.5519
Word trigrams	41.52	0.6864
Tag trigrams	31.28	0.6336
Word and tag trigrams	43.62	0.6945
Syntactic features	33.50	0.6439
All features	51.33	0.7219
Oracle	100.0	0.8662

Table 4.9: Best match accuracies and GTM scores for fluency models incorporating n-gram language models and syntactic features from parse disambiguation. The model that combines syntactic features with language models (*all features*) outperforms the other models.

	word trigrams	tag tigrams	trigrams	syntactic
Word trigrams				
Tag trigrams	$\mathbf{p < 0.0001}$			
Both trigrams	$p = 0.0045$	$\mathbf{p < 0.0001}$		
Syntactic	$\mathbf{p < 0.0001}$	$p = 0.0262$	$\mathbf{p < 0.0001}$	
All	$\mathbf{p < 0.0001}$	$\mathbf{p < 0.0001}$	$\mathbf{p < 0.0001}$	$\mathbf{p < 0.0001}$

Table 4.10: Experiment-wise significance at 99% level, $p < 0.001$.

These scores may seem relatively low, but they should be seen as an indication of the relative performance of each model. Since we created the training and testing data automatically from treebanks, there was no manual verification that the dependency structure used was the correct reading of the original sentence. Additionally, we only have the treebank sentences as an annotation, while there can be more than one fluent realization.

4.7.3 Empirical probabilities

Section 4.3.6 discussed four different methods to estimate the empirical probabilities $\tilde{p}(l)$ and $\tilde{p}(l, d)$: (1) uniform inputs [Osborne, 2000]; (2) weighted inputs [van Noord and Malouf, 2005]; (3) uniform inputs with binary events [de Kok et al., 2011]; and (4) weighted inputs with binary events. We experimented with these four methods to train a model with all available features. The results of this experiment are shown in Table 4.11. As we can see, there is some benefit in carefully selecting a method. Method (3) outperforms the other methods substantially when comparing on best match accuracy, while the overall GTM

score improves slightly. However, the differences in performance are not significant (at $p < 0.001$).

Input probabilities	**Best match (%)**	**GTM**
Uniform inputs	50.40	0.7178
Weighted inputs	50.46	0.7205
Uniform inputs with binary events	**51.33**	**0.7219**
Weighted inputs with binary events	51.08	0.7211

Table 4.11: Best match accuracies and GTM scores for fluency models trained using four different methods for estimating the input probability. The model using uniform inputs with binary events outperforms the other models.

	Uniform	Weighted	Uniform (binary evts)
Uniform			
Weighted	$p = 0.0842$		
Uniform, binary evts	$p = 0.0228$	$p = 0.3580$	
Weigted, binary evts	$p = 0.0245$	$p = 0.3905$	$p = 0.9530$

Table 4.12: Experiment-wise significance at 99% level, $p < 0.0017$.

As we will see in the next chapter, methods (3) and (4) have another benefit, in that they are more generic. Since the derivations are just marked as 1 (correct) or 0 (incorrect), the training data can be mixed with training data that used another scoring function. This will be taken advantage of in Section 5.5.1.

4.7.4 Error analysis

In the previous section, we evaluated fluency ranking models using automated methods. They give a good indication of the performance of a fluency ranker, but do not provide an analysis of the types of errors that the ranker makes. In this section, we discuss some of the differences between the realizations chosen by the ranker, and the realizations that resemble the gold standard the most. As we will see, some of the differences are errors made by the fluency ranker, some differences are caused by limitations of the input representation, and finally there are cases where more than one sentence can be considered fluent.

Punctuation One minor source of differences is punctuation. As we discussed in Section 3.4.2, punctuation is not specified in abstract dependency

structures. In some cases, it is necessary to introduce punctuation to successfully construct a realization. Often different punctuation tokens are allowed, leading to a situation where many realizations only differ in the punctuation sign(s) being used. In such cases, the realization with the highest n-gram language model score is used. In the following sentence we see that the best realization according to the ranker (*a*) uses a comma as the non-optional punctuation character, while the best realization (*b*) with the highest correspondence to the gold standard uses a hyphen:[13]

(27) a. Hij stond voor de naastenliefde maar werd toch de klos **,**
He stood for the benevolence but became still the victim ,
zulke dingen gebeuren nog dagelijks.
such things happen still daily.
b. Hij stond voor de naastenliefde maar werd toch de klos **-** zulke dingen gebeuren nog dagelijks.

In the following example, the tendency of the generator to introduce as little punctuation as possible reduces readability:

(28) a. Enig beroep is niet mogelijk en zelfs het parlement mag niet
Any appeal is not possible and even the parliament may not
weten welke bedrijven steun ontvangen**,** aldus
know which companies support receive, according-to
mr.-Polak, lid van de Eerste-Kamer.
mr.-Polak, member of the First-Chamber.
b. Enig beroep is niet mogelijk en zelfs het parlement mag niet weten welke bedrijven steun ontvangen**.** aldus mr.-Polak, lid van de Eerste-Kamer.

When reading this sentence, it is not immediately clear that it starts with a quote. In a fluent realization, punctuation would be used to mark the quote:

(29) **“**Enig beroep is niet mogelijk en zelfs het parlement mag niet weten welke bedrijven steun ontvangen**.”**, aldus mr.-Polak, lid van de Eerste-Kamer.

Realization of words Sometimes there are multiple realizations of lexical attribute-value structures, even if their lexical information is not underspecified. For instance, the word *vóór* has an alternative inflection in the Alpino lexicon, namely *voor*:

[13]The same order is used in all the examples that follow in this section.

(30) a. Maar het pleidooi **voor** heeft de tijd niet mee.
But the plea in-favor has the time not with-it.
(But the plea was not made in a favorable climate.)

b. Maar het pleidooi **vóór** heeft de tijd niet mee.

The following realizations contain a compound that was unknown to the lexicon. In such cases, the productive lexicon (Section 2.4.3) creates one variant with and one without a hyphen that connects the compounded words:

(31) a. Eind 2003 zouden alle 800 **Laurus-winkels** zijn omgebouwd.
End 2003 would all 800 Laurus-shops be rebuilt.
b. Eind 2003 zouden alle 800 **Lauruswinkels** zijn omgebouwd.

Conjunctions As expected, there is a lot of variety in conjunctions, since they are fairly frequent, and their conjuncts can assume any order. In many cases, the ordering does not have a notable impact on fluency. We give two examples:

(32) a. **Graven, koningen en prinsen** hadden een orkestje
Counts, kings and princes had a small-orchestra
tot hun beschikking en een kapelmeester die het muzikale
to their availability and a chapel-master who the musical
leven aan het hof organiseerde.
life of the royal-household organized.
Counts, kings, and princes had a small orchestra available to them, as well as a chapel master who managed music in the royal household.)
b. **Graven, prinsen en koningen** hadden een orkestje tot hun beschikking en een kapelmeester die het muzikale leven aan het hof organiseerde.

Another example where the conjuncts can be ordered freely is:

(33) a. Pronk is **al zestien jaar minister en zit al**
Pronk is already sixteen years minister and sits already
dertig jaar in de nationale politiek.
thirty years in the national politics.
b. Pronk zit **al dertig jaar in de nationale politiek en is al zestien jaar minister**.

A difference in emphasis can be observed in these realizations: the first emphasizes that Pronk has been a minister for sixteen years, the second that he has been active in politics for thirty years.

In the following example, proper ordering is required since the sentence uses antithesis for rhetorical effect. The writer is an adult, but is put in a situation where he has to do childish things. The rhetoric effect is (unfortunately) mostly absent in the realization that the fluency ranker proposes:

(34) a. Ik loop ineens met een molentje dat in de wind draait te
I am suddenly with a little-mill that in the wind turns to
zwaaien en ik ben allang volwassen.
wave and I am so-long adult.
(I have been adult for a long time and suddenly I am waving with a small mill that spins in the wind.)

b. Ik ben allang volwassen en ineens loop ik met een molentje te zwaaien dat in de wind draait.

There are almost no cues in the sentence that could help the ranker to give the correct order of conjuncts, except perhaps the word *ineens* 'suddenly'.

Equally difficult are conjunctions that form a sequence of instructions. The following realization is part of a recipe, although the fluency ranker chose a realization with an incorrect order of instructions:[14]

(35) a. Dek af, laat anderhalf uur pruttelen en giet er dan
Cover, let one-and-a-half hour simmer and pour there then
genoeg water zodat alles net onder staat bij.
enough water such-that everything just under water into.
(Cover, let it simmer for one and a half hour and pour enough water such that everything is drenched.)

b. Giet er dan genoeg water bij zodat alles net onder staat, dek af en laat anderhalf uur pruttelen.

In this example, a human would be able to deduce the correct order of instructions — we know that you should not cook something for one and a half hour before adding water. However, such knowledge is not available to the fluency ranker. It is not hard to think up comparable examples where even humans cannot deduce the correct ordering of conjuncts.

Many of these 'errors' are not choices that the ranker could realistically decide on. The abstract representation, on the other hand, currently lacks the

[14]The word *bij* should also be placed after *water*.

possibility to impose a particular order in conjunctions.

Modifier fronting As we discussed previously, subject fronting in Dutch is preferred over direct object fronting. However, it turns out that in many cases the fronting of a modifier is considered to be fluent as well. In Table 4.13 we list the five most frequent dependency relations in the vorfeld (the constituent positioned before the inflected verb in a main clause) in the Eindhoven corpus. This shows that direct object fronting is indeed rare, but also that modifier fronting occurs fairly frequently.

Relation	**Frequency**	**%**
Subject	4310	60.52
Modifier	2150	30.19
Direct object	161	2.26
Predicative complement	124	1.74
PP of a prepositional verb	98	1.38

Table 4.13: Dependency relation of constituents in the vorfeld in the Eindhoven corpus.

The consequence is that the fluency ranking model, besides the preference for subject fronting, learns a conflicting preference for modifier fronting. This is not an error, since modifier fronting is considered to be fluent in many cases, but does decrease accuracy in the automatic evaluation. In some cases the ranker will pick the modifier-fronting realization and the gold standard the subject fronting realization, and vice versa. We give two examples of both scenarios:

(36) a. **Misschien** was **Nevill** toch niet de arrogante cynicus
Maybe was Nevill nevertheless not the arrogant cynical
of de lolbroek voor wie zijn mede-officieren hem hielden.
or the joker for which his fellow-officers him took.
b. **Nevill** was **misschien** toch niet de arrogante cynicus of de lolbroek voor wie zijn mede-officieren hem hielden.

(37) a. **Zoals verwacht** ging **de strijd** in de koningsklasse tussen
As expected was the battle in the highest-class between
de Italiaanse kemphanen Biaggi, Rossi en Capirossi.
the Italian scrappers Rossi, Biaggi, and Capirossi.
b. **De strijd** in de koningsklasse ging **zoals verwacht** tussen de

Italiaanse kemphanen Rossi, Biaggi en Capirossi.

In both examples, the realizations are (according to our judgment) equally fluent, but the ranker obtained a lower GTM score for not choosing the best match.

Coreferencing Coreferencing still poses a problem for the current ranker. For instance, in the following realizations, the ranker preferred a realization where the referring expression is introduced before the referent:

(38) a. Anders moeten ze stoppen en de boeren moeten
Otherwise have-to they stop and the farmers have-to
kunnen concurreren met de rest van de wereld.
be-able-to compete with the rest of the world.
b. De boeren moeten kunnen concurreren met de rest van de wereld
en anders moeten ze stoppen.

In some cases, it may be feasible to improve ordering when co-referencing is used. For example, if the referent and the referring expression are both subjects in a conjunction of sentences, we could try to model the ordering of e.g. proper names and pronouns. For instance, such a strategy could work for realizing an abstract dependency structure that corresponds to the following sentence:

(39) Zo hanteerde **Scheffer** toevalstechnieken bij
In-that-manner used Scheffer coincidence-techniques during
de montage van zijn film over John Cage en speelde **hij** met
the editing of his movie about John Cage and played he with
verschillende tempi in het portret over Elliott Carter.
different tempos in the portret about Elliot Carter.

However, in other cases this does not work, since the subjects do not always refer to the same entity. For example:

(40) **Ze** wilde niet trouwen en **Bratt** zou daarom hebben
She wanted not marry and Brett would for-that-reason have
aangestuurd op het verbreken van de relatie.
steered towards the break-up of the relationship.

To address this issue properly, co-references should be marked a such in abstract dependency structure. This would allow for the training of a ranker, such that a referent is introduced first in a realization.

Miscellaneous Finally, there is a long tail of less frequent types of errors and differences that do not influence fluency. We will give two examples. The first example shows that a separable verb particle can be placed before and after a prepositional complement:

(41) a. Bij het ingaan van de finale maakte hij deel van een
During the start of the final made he part of a
kopgroep van negen man **uit**.
front-line of nine men out
(During the start of the final he was part of a nine men front-line.)
b. Bij het ingaan van de finale maakte hij deel **uit** van een kopgroep van negen man.

It is not clear that one of these realizations is more fluent than the other.

The second example is the discontinuous realization of direct objects with modifying phrases. For instance, consider the following realizations:

(42) a. Joegoslavië moet meer aan onderwijs besteden, aldus
Yugoslavia should more on education spend, according-to
de Wereldbank die echter ook **mogelijkheden voor**
the World-bank who however also possibilities for
opmerkelijke bezuinigingen ziet.
remarkable cost-cutting sees.
b. Joegoslavië moet meer besteden aan onderwijs, aldus de Wereldbank die echter ook **mogelijkheden ziet voor opmerkelijke bezuinigingen**.

In the realization that resembles the gold standard the most, the direct object *mogelijkheden voor opmerkelijke bezuinigingen* is split to place the verb *ziet* after the head of the direct object *mogelijkheden*. Such discontinuous direct objects also occur in main clauses. For example:

(43) a. Ik heb het recht **de beste optie voor mezelf en mijn**
I have the right the best option for myself and my
familie te kiezen.
family to choose.
b. Ik heb het recht **de beste optie te kiezen voor mezelf en mijn familie**.

Here the direct object *de beste optie voor mezelf en mijn familie* is split in the gold standard to place *te kiezen* between *het recht* and the modifying phrase *voor mezelf en mijn familie*. Again, none of these realisations would be consid-

ered more fluent than the other.

4.7.5 A note on prenominal modifiers

We have discussed the role of ordering prenominal modifiers on fluency in Section 4.1.1. It is clear that the trigram model that we have discussed in this chapter provides a probability distribution over such modifiers. It can be argued though, that a word trigram model may not be the best estimator for such long sequences of modifiers. Given the factorial number of possible orderings, many orderings will not occur in the training sample. Consider a sentence, such as

(44) een groot wit Nederlands vliegtuig
a big white Dutch airplane

Even if *groot wit opstijgend Nederlands* does not occur in the training sample, the training sample could still settle relevant questions with respect to the ordering of these modifiers, such as: is *groot* followed by *Nederlands* more frequent than the opposite? Does *groot* tend to be far from its head?

Given the opportunity to exploit corpora further, it is not surprising that more elaborate corpus statistics have been used in different works:

- Quirk and Greenbaum [1974], Dixon [1977], and Sproat and Shih [1991] have argued that modifiers can be ordered by their underlying semantic properties. For instance, Sproat and Shih [1991] proposed the ordering Quality > Size > Shape > Color > Provenance. Such ordering preferences could be extracted automatically from an annotated corpus.

- Mitchell [2009] applies a corpus-based approach, where modifiers are grouped in broad classes based on where they usually occur prenominally. A class can correspond to a specific position (e.g. immediately preceding the noun) or multiple consecutive positions. When realizing a noun phrase, the orderings are filtered such that they do not violate the (trained) class constraints.

- Shaw and Hatzivassiloglou [1999] uses a corpus to construct a $w \times w$ matrix, where w is the distinct number of prenominal modifiers, and the cell $[A, B]$ contains the number of occurrences where 'A' preceded 'B'. This information is then used to check modifier orderings, for if there is a preference for 'A' to occur before 'B', then $[A, B] \gg [B, A]$.

- Liu and Haghighi [2011] uses a maximum entropy model that incorporates many features, such as the 2 to 5-gram counts of the prenominal

modifiers, characteristics of the modifier at each position (such as the use of hyphenation, suffix, and length), and satisfaction of the Mitchell class ordering.

We did some preliminary experiments with a model that is comparable to that of Liu and Haghighi [2011]. However, this did not provide a worthwhile improvement. It turns out that in our data, nouns rarely have more than two prenominal modifiers. In Table 4.14 and Table 4.15 we provide counts of the number of modifiers head nouns have in the Eindhoven and PPH corpora. These frequencies were obtained using Dact [van Noord et al., 2012]. We can see that in both corpora, nouns rarely have more than two modifiers (0.16% and 0.04% respectively). Previous works on prenominal modifiers usually focused on lexical modifiers, such as adjectives. If we restrict ourselves to these cases (fourth and fifth columns in both tables), this number even decreases somewhat, as expected.

# Modifiers	**Modifiers**	**%**	**Lexical modifiers**	**%**
0	14298	68.76	15140	72.81
1	5994	28.83	5287	25.43
2	467	2.25	345	1.66
3	32	0.15	20	0.10
4	3	0.01	2	0.01
≥ 5	0	0.00	0	0.00

Table 4.14: Number of modifiers of noun heads in the Eindhoven corpus. The fourth and fifth columns only include lexical modifiers.

# Modifiers	**Modifiers**	**%**	**Lexical modifiers**	**%**
0	3784	69.47	3913	71.84
1	1541	28.29	1440	26.44
2	120	2.20	92	1.69
3	2	0.04	2	0.04
≥ 4	0	0.00	0	0.00

Table 4.15: Number of modifiers of noun heads in the PPH corpus. The fourth and fifth columns only include lexical modifiers.

Summarized, further investigation in the ordering of noun phrases with more than two prenominal modifiers is not worthwhile for the corpora that we use. It would be interesting to study whether this is also true for other domains.

Chapter 5

Reversible SAVG

5.1 Introduction

In the previous chapter, we augmented the Alpino attribute-value grammar with a model for fluency ranking. Together, these form a *Stochastic Attribute-Value Grammar* (SAVG). As a result, Alpino actually consists of two SAVGs, one for parsing and one for generation. In this chapter, we propose *Reversible Stochastic Attribute-Value Grammar* (RSAVG), a grammar that uses a single stochastic model for both fluency ranking and parse disambiguation.

5.2 Parse disambiguation

In Chapter 4, we described our fluency ranking model for the Alpino grammar and chart generator. This model ranks derivations by the fluency of their embedded realizations. Parsing, which uses a grammar to prove that a sentence is valid with a particular syntactic or semantic analysis, has a counterpart to fluency ranking, *parse disambiguation*. Since human language is ambiguous, parsing of a sentence often yields many derivations that correspond to different readings of that sentence. For instance, reconsider the sentence

(1) a. **De baseliner** speelde ter voorbereiding op het Grand Slam **één grastoernooi**.
The baseliner played for preparation of the Grand Slam one grass-tourney.

b. **Eén grastoernooi** speelde **de baseliner** ter voorbereiding op het Grand Slam.

The sentence can be read such that *de baseliner* is the subject and *één grasto-ernooi* the direct object, or vise versa.

Parse disambiguation attempts to solve ambiguity by selecting the most likely reading of a sentence. In the example above, a parse disambiguation component should normally choose the reading having the *de baseliner* as the subject, since in Dutch, subject fronting is preferred over direct object fronting (Section 4.1.1).

However, as implied by the word 'ambiguity', parse disambiguation is also a stochastic process. Although reading a fronted subject is preferred in general, sometimes there is reason to read the initial noun phrase of a sentence as a direct object. For instance, in a sentence such as

(2) a. **De resultaten van de berekeningen** toetsen **de**
The results of the calculations check the
wetenschappers aan de waarnemingen op zee op het ogenblik
scientists with the observations on sea at the moment
dat de satelliet voorbijkomt.
that the satellite passes by.
The results of the calculations are checked against observations at sea at the moment the satellite passes by.

the noun phrase *de resultaten van de berekeningen* 'the results of the calculations' is the direct object of the main verb *toetst* 'check', since calculations are something to be checked.

Chapter 4 discussed how an attribute-value grammar can be complemented by a discriminative model to estimate the fluency of a realization. Likewise, an attribute-value grammar can be complemented by a discriminative model for parse disambiguation. In fact, such combinations are successfully employed in a wide variety of systems [Toutanova et al., 2002, Riezler et al., 2002, Miyao and Tsujii, 2005, Clark and Curran, 2004, Forst, 2007], including Alpino [van Noord, 2006]. We call such discriminative models *directional*, since they either work in parse disambiguation or fluency ranking, but not both.

5.3 Limitations of directional models

The development of separate models for parse disambiguation and fluency ranking led to a situation in which an attribute-value grammar is coupled with two distinct stochastic components. One of these is employed during parsing, the other during generation. This is reasonable to some extent, because some features that can be employed by the stochastic model are only relevant in a

certain direction. For instance, in the fluency model described in Chapter 4, a trigram language model is included as an auxiliary distribution [Johnson and Riezler, 2000]. Obviously, such information is irrelevant for parsing since the sentence is given. On the other hand, the disambiguation model of Alpino contains features which judge aspects of the dependency structure of a derivation. Since the dependency structure is fixed in generation, these features are irrelevant in fluency ranking.

Yet, there are also many features that appear to be relevant in both directions. For instance, the feature discussed in Chapter 4 that signals topicalization of NP subjects (Section 4.4) is very effective in fluency ranking and parsing. The reason simply is that, in Dutch, the unmarked ordering is that subjects precede objects. This generalization is relevant both for parsing and generation. Such parallelism is not accidental, of course. Clearly, if surface realization favored direct object fronting, whereas disambiguation favored a subject reading in such cases, communication would become problematic.

In this chapter we propose the formalism of Reversible Stochastic Attribute-Value Grammars (RSAVG) in which we maintain only a *single* stochastic component which is used both for parsing and generation. As before, the stochastic component implements a conditional model, but we condition more abstractly over the set of constraints that the input specifies, rather than the sentence (for parsing) or the dependency structure (for generation).

There is also practical motivation for reversible stochastic attribute-value grammar. We provide experimental results indicating that, under certain conditions, fluency models and parse disambiguation models are out-performed by reversible models.

But before discussing RSAVG, we give a short overview of stochastic models that have been used previously in parse disambiguation and fluency ranking. Since the earliest models were reversible, our aim is to show the (historical) motivation for the introduction of directional models.

5.4 A history of SAVG

5.4.1 Stochastic context-free grammar

Stochastic context-free grammar (SCFG) has traditionally been and continues to be a popular formalism for estimating the probability of a derivation. An SCFG consists of context-free grammar rules, where each rule $V \rightarrow w$ is assigned a conditional probability $p(w|V)$, usually written as $p(V \rightarrow w)$. Since the probability is conditioned on the left-hand side of the rule, the conditional probabilities of all grammar rules with the same left-hand side symbol should

sum to one. Formally, if N is the set of non-terminal symbols and R the set of grammar rules, then:

$$\forall_{V \in N}[\sum_{V \rightarrow w \in R} p(V \rightarrow w)] = 1 \tag{5.1}$$

The relative frequency, in a treebank, of the rule $V \rightarrow w$ among all rules that have V on their left-hand side provides a maximum-likelihood estimate (MLE) of $p(V \rightarrow w)$:

$$p(V \rightarrow w) = \frac{C(V \rightarrow w)}{C(V)} \tag{5.2}$$

where $C(V)$ is the frequency of V in the treebank. The probability of a derivation is then calculated by multiplying the probability of the rules that were applied to construct the derivation. If $R(d)$ is the multi-set of the rules that were used in the construction of derivation d, then

$$p(d) = \prod_{V \rightarrow w \in R(d)} p(V \rightarrow w) \tag{5.3}$$

It can be shown that relative frequencies provide maximum likelihood estimates in SCFG. However, Abney [1997] shows that the use of relative frequencies does not yield maximum likelihood estimates in formalisms that are more powerful than context-free grammar. In other words, SCFG provides an optimal model for $\mathcal{G}_s$ (Section 2.2.2), but not $\mathcal{G}$. $\mathcal{G}$ is more powerful than context-free grammar, since it can define constraints across syntactic categories.

Following Abney [1997], we will show intuitively that empirical relative frequencies do not lead to an optimal solution for AVGs. Consider the attribute-value grammar fragment in Figure 5.1. Now, suppose that we have the treebank in Figure 5.2. Using this treebank, we can derive:

- $p(x \rightarrow y\ y) = \frac{C(x \rightarrow y\ y)}{C(x)} = 1.0$
- $p(y \rightarrow a) = \frac{C(y \rightarrow a)}{C(y)} = 0.5$
- $p(y \rightarrow b) = \frac{C(y \rightarrow b)}{C(y)} = 0.5$

Suppose that we want to estimate the probability of the only possible derivation of *aa*, we would obtain the probability $p(x \rightarrow y\ y) \cdot p(y \rightarrow a) \cdot p(y \rightarrow a) = 1 \cdot 0.5 \cdot 0.5$. However, this is an not an accurate estimation of the probability of this derivation. The rule $x \rightarrow y\ y$ requires the SURF attributes of both right-hand side slots to unify. The rule $y \rightarrow b$ is not an option anymore after

using $y \rightarrow a$ to rewrite the first y symbol. Consequently, the probability of completing the second slot with $y \rightarrow a$ is one. Since the grammar can only produce two possible trees, and both trees occur once in the treebank, the correct probability of the derivation of aa is 0.5.

A related issue with this model is that it does not even provide a correct probability distribution. The grammar can only generate the trees in Figure 5.2, however $p(tree(aa)) + p(tree(bb)) \neq 1$.

$$
\begin{array}{c}
{}_{x}\left[\text{SURF} \quad \left\langle \boxed{1}, \boxed{1} \right\rangle\right] \rightarrow {}_{y}\left[\text{SURF} \quad \boxed{1}\right] \quad {}_{y}\left[\text{SURF} \quad \boxed{1}\right] \\
{}_{y}\left[\text{SURF} \quad a\right] \rightarrow a \\
{}_{y}\left[\text{SURF} \quad b\right] \rightarrow b
\end{array}
$$

Figure 5.1: A grammar fragment with constraints on categories.

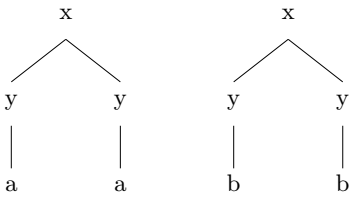


Figure 5.2: A simple treebank conformant to the grammar in Figure 5.2.

This example shows that once constraints are introduced between categories, there is a chance that relative frequencies do not provide an optimal estimation of the probability of applying a rule.

5.4.2 Maximum entropy modeling

Abney proposes to use a maximum entropy model to estimate the probability of a derivation instead. This (non-conditional) model estimates the probability of a derivation, $p(d)$:

$$p(d) = \frac{1}{Z} exp \sum_{i} \theta_i f_i(d) \tag{5.4}$$

Each derivation is characterized by a set of feature values. Thus, $f_i(d)$ is the value of feature f_i in derivation d. A parameter θ_i is associated with each feature f_i. In (5.4), Z is a normalizer which is defined as follows, where Ω is the set of derivations defined by the grammar:

$$Z = \sum_{d' \in \Omega} exp \sum_i \theta_i f_i(d') \tag{5.5}$$

Note that in the case we only want to find the best derivation, it suffices to compute $\sum_i \theta_i f_i(d)$ for each d. The resulting model is applicable both for parsing and generation. In contrast to the model that was described in Chapter 4, the probability of the derivation is not conditioned on an abstract representation or a sentence. As a consequence, this model can be used for parse disambiguation and fluency ranking.

The model is trained on the basis of a representative set of correct derivations: a treebank. Weights are estimated for each feature f_i such that the expected value of a feature in the model $E_p(f)$ is equal to its empirical feature expectation $E_{\tilde{p}}(f)$:

$$\begin{aligned} & E_p(f_i) = E_{\tilde{p}}(f_i) \\ \equiv \quad & E_p(f_i) - E_{\tilde{p}}(f_i) = 0 \\ \equiv \quad & \sum_{d \in D} p(d) f_i(d) - \tilde{p}(d) f_i(d) = 0 \end{aligned} \tag{5.6}$$

where D is the set of derivations in the treebank, $p(d)$ is the probability of derivation d according to the model, and $\tilde{p}(d)$ is the empirical distribution of d, in this case simply the relative frequency of d in the treebank.

Since it is necessary to calculate $p(d)$ during training, training requires access to *all* derivations (Ω) allowed by the grammar. Obviously, for any non-trivial grammar this set of derivations is infinite. Abney [1997] suggests to approximate the normalizer with random sampling using the Metropolis-Hastings algorithm, but the resulting procedure is believed to be impractical for realistically sized grammars.

5.4.3 Conditional maximum entropy models

Johnson et al. [1999] avoid the expensive computation of the normalizer Z by using a conditional maximum entropy model. In parse disambiguation, such a model estimates the probability of a derivation d given a sentence s, $p(d|s)$. Under the reasonable assumption that the number of derivations for a given

sentence s is finite, the number of derivations may still grow exponentially with the sentence length, but the calculation of the normalizer in a conditional model is less problematic. The model can be estimated using an informative sample of derivations [Osborne, 2000] or by using dynamic programming to calculate the feature expectations and normalizers over packed charts [Miyao and Tsujii, 2002, Geman and Johnson, 2002, Clark and Curran, 2003]. Conversely, as described in Chapter 4, the probability of a derivation d given the abstract representation l can also be estimated.

Since conditional maximum entropy models were discussed extensively in Chapter 4, we only repeat their parametric form here for brevity,

$$p(d|x) = \frac{1}{Z(x)} exp \sum_{i=1}^{|F|} \theta_i f_i(x, d) \tag{5.7}$$

$$Z(x) = \sum_{d \in \Omega(x)} exp \sum_{i=1}^{|F|} \theta_i f_i(x, d) \tag{5.8}$$

where x is either a sentence s or an abstract representation l.

The introduction of conditional models, which normalize over the yield of an input rather than all derivations generated by a grammar, made it computationally feasible to construct models for realistically sized grammars with a large number of features. However, the other side of the coin is that these conditional models are used directionally, since they estimate $p(d|s)$ or $p(d|l)$ respectively.

5.5 Reversible stochastic attribute-value grammars

5.5.1 Finding the best consistent derivation

In the bigger picture, parsing and generation can be seen as two instances of a more general task: given a set of constraints that the input specifies, give all possible derivations that are consistent with these constraints. In the case of parsing, the constraints restrict the set of possible derivations to those that have the same words and word order as the input sentence. In generation, the constraints restrict the set of possible derivations to those having a dependency structure corresponding to the input.

In reversible stochastic attribute-value grammars (RSAVG) we exploit this view to introduce one model for parse disambiguation and fluency ranking.

RSAVG estimates the probability of a derivation d given the set of input constraints c, $p(d|c)$. We use a conditional maximum entropy model to estimate $p(d|c)$:

$$p(d|c) = \frac{1}{Z(c)} exp \sum_i \theta_i f_i(c,d) \tag{5.9}$$

$$Z(c) = \sum_{d' \in \Omega(c)} exp \sum_i \theta_i f_i(c,d') \tag{5.10}$$

As discussed in the previous section, directional conditional models assume that the number of derivations of a sentence or abstract representation is finite. In RSAVG, we maintain this assumption – for each set of constraints that forms an input to the system, there should be a finite number of derivations that are consistent with the constraints.

We derive a reversible model by training on data for parse disambiguation and fluency ranking simultaneously (Section 5.5.2). During training, we impose constraints on the feature values with respect to the sentences S in the parse disambiguation treebank and the dependency structures L in the fluency ranking treebank:

$$\sum_{s \in S} \sum_{d \in \Omega(s)} \tilde{p}(s) p(d|c=s) f_i(s,d) - \tilde{p}(c=s,d) f_i(s,d) = 0 \tag{5.11}$$

$$\sum_{l \in L} \sum_{d \in \Omega(l)} \tilde{p}(l) p(d|c=l) f_i(l,d) - \tilde{p}(c=l,d) f_i(l,d) = 0$$

where $\Omega(s)$ is the set of derivations for sentence s and $\Omega(l)$ is the set of derivations for dependency structure l

If parse disambiguation and fluency ranking are governed by the same preferences (Section 5.3), enforcing constraints with respect to parse disambiguation and fluency ranking data simultaneously should should have no negative effect on estimating weights for features that are used in both directions. Estimation of weights for task-specific features should be unaffected, since they have constant values in the training instances of the opposite task.

Practically, it is also possible to use one type of constraint by concatenating the training data for fluency ranking and parse disambiguation:

$$\sum_{i \in S \cup L} \sum_{d \in \Omega(i)} \tilde{p}(i) p(d|c=i) f_i(i,d) - \tilde{p}(c=i,d) f_i(i,d) = 0 \tag{5.12}$$

However, with such an approach one should ensure that the empirical probabilities ($\tilde{p}(i)$ and $\tilde{p}(i,d)$) are estimated such that $\tilde{p}(i)$ is a uniform distribution. A uniform distribution of $\tilde{p}(i)$ implies that there is only competition between derivations for an input. If task-specific scoring methods (such as GTM and CA scores) are used intermixed using a non-uniform distribution $\tilde{p}(i)$, the parameter estimator may be biased on the task with the highest average scores or the highest average number of derivations per input. This does not turn out to be problematic in practice, since we have shown that the best-performing method for estimating $\tilde{p}(i,d)$ in fluency ranking uses binary derivation scores and a uniform distribution for $\tilde{p}(i)$ (Section 4.7.3). For completeness, we give the results of applying different methods for normalizing $\tilde{p}(s,d)$ in Table 5.1. It is not surprising that here, the methods that use binary events also outperform the methods that use the quality scores directly, although the differences are (again) not significant.

Input probabilities	**CA-score (%)**
Uniform inputs	90.93
Weighted inputs	90.91
Uniform inputs with binary events	91.14
Weighted inputs with binary events	91.18

Table 5.1: CA-scores for parse disambiguation models, using various methods to estimate $\tilde{p}(s)$ and $\tilde{p}(s,d)$.

5.5.2 Symmetric treebanks

For training reversible models, the previous section assumed separate parse disambiguation data and fluency ranking data. An interesting special case is the notion of symmetric treebanks [Velldal et al., 2004]. A symmetric treebank consists of pairs of abstract representations and sentences. The treebank is symmetric in the sense that for a pair of a sentence and an abstract representation, the abstract representation is an appropriate analysis of the sentence, and the sentence is a fluent realization of the abstract representation.

Conventional treebanks in which sentences are annotated with the corresponding abstract representations can be considered symmetric treebanks, under the assumption that the writers of the sentences are fluent in the language that is used and attempted to write as fluently as possible for the given domain. Since treebanks often use, as in our case, newspaper text, this assumption usually holds.

Models are trained and evaluated using competing derivations for each input (abstract representation or sentence). These derivations are obtained by creating all derivations that are consistent with the constraints in the input. Derivations that have a symmetric abstract representation and sentence according to the treebank are marked correct. All other competing derivations are marked incorrect.

The resulting training data contains for each abstract representation and sentence in the treebank a set of correct and incorrect derivations. Consequently, it can be used to train or evaluate a reversible model.

5.6 Evaluation methodology

We think that one reversible model is to be preferred over two distinct parsing and fluency models for theoretical reasons as well as for more practical reasons of simplicity, compactness, and maintainability. As an additional advantage, reversible models are applicable for tasks which combine aspects of parsing and generation, such as paraphrasing. Of course, these benefits are convincing only if the resulting reversible models achieve similar performance to their directional counterparts.

We are also interested in seeing if certain baseline models can be improved by adding derivations from the opposite direction to the training data (*cross-pollination*). For instance, if we had only the n-gram trigram auxiliary distributions available in the training data for fluency ranking, would the accuracy of the model improve if we train the weights of other features using derivations from parsing?

In this section, we first provide a description of how parse disambiguation models are trained and evaluated (the training and evaluation of fluency ranking models was already introduced in Section 4.6). We will then discuss how we train and evaluate reversible models. Finally, we will introduce some scenarios to test how effective the aforementioned *cross-pollination* is.

5.6.1 Training and evaluation of parse disambiguation

The parse disambiguation models are trained and tested on the same treebanks as fluency ranking (Section 4.6.1). However, since the Alpino parser provides an analysis for every sentence in these treebanks, all the sentences are available as training and evaluation instances.

Since the treebanks only contain an abstraction of a derivation in the form of a dependency structure, as in generation, we parse each sentence to construct the derivations of each sentence. In the current system, we create at most 3000

derivations for each sentence. The quality of each derivation is estimated using the concept accuracy score (CA-score) that was introduced in Section 3.5.1.

During training, we take a representative sample of at most 100 derivations per training input. The model is then trained using TinyEst with an ℓ_2 prior of $\sigma^2 = 1000$. The probability $p(s, d)$ is estimated using binary scores with uniform input probabilities.

Since we only discussed features that are active in fluency ranking in Section 4.4, we will now give a short description of the features that are active in parse disambiguation and have not yet been discussed in that section. These features can be divided in two sets, namely features modeling the distribution of *lexical frames* and *dependency relations*.

Lexical frames The parser applies lexical analysis to find all possible subcategorization frames for tokens in the input sentence. Since some frames occur more frequently in good parses than others, two feature templates record the use of frames in derivations. An additional feature implements an auxiliary distribution of frames, trained on a large corpus of automatically annotated sentences (436 million words). The values of lexical frame features are constant for all derivations in sentence realization, unless the frame is underspecified in the dependency structure.

Dependency relations Several templates describe aspects of the dependency structure. For each dependency relation multiple dependency features are extracted. These features list the dependency relation, and characteristics of the head and dependent, such as their roots or part of speech tags. Additionally, features are used to implement auxiliary distributions for selectional preferences [van Noord, 2007]. In generation, the values of these features are constant across derivations corresponding to a given dependency structure.

5.6.2 Baseline scenarios

Since preferences are shared between parse disambiguation and fluency ranking, it is conceivable that a weak model could benefit from the use of training data from the opposite direction. A model could be weak in two senses: (1) the amount of task-specific training data available is relatively small; and (2) the task-specific training data uses a smaller feature set.

To see if this cross-pollination from the opposite task occurs, we performed four experiments where we reduce the amount of training data or the number of features. The four experiments are:

1. Comparing the learning curve of a directional fluency ranking model to the learning curve of a reversible model that uses the given percentage of fluency ranking training data and is simultaneously trained on *all* parse disambiguation training data. In other words, the training data consists of (1) the derivations obtained by generating said percentage of the abstract dependency structures in the training data and (2) the derivations obtained by parsing all of the sentences in the training data.

2. Comparing the learning curve of a directional parse disambiguation model to the learning curve of a reversible model that uses the given percentage of parse disambiguation training data and is simultaneously trained on *all* fluency ranking training data.

3. Training models where the fluency ranking training data contains only n-gram features, with and without training data from parse disambiguation.

4. Training models where the parse disambiguation training data contains only rule identifier features, with and without training data from parse disambiguation.

5.7 Results

5.7.1 Parse disambiguation

Table 5.2 shows the performance of the reversible model compared to the directional parse disambiguation model. We also provide lower and upper bounds: the baseline model selects an arbitrary parse per sentence, while the oracle chooses the best available parse.

Model	**CA (%)**	**f-score (%)**
Baseline	75.63	76.11
Oracle	95.23	95.44
Parse model	91.14	91.46
Reversible	90.99	91.32

Table 5.2: Concept Accuracy scores and f-scores for the parsing-specific model versus the reversible model.

The results show that the performance of the general, reversible, model is very close to that of the dedicated, parsing-specific model. In fact, the difference is not significant.

Suppose we had a less informed parse disambiguation model, could it then be possible that adding fluency ranking data to a parse disambiguation model would improve parsing accuracy? To answer this question, we experimented with the baseline scenarios described in Section 5.6.2. The results of the experiment with fewer features is shown in Table 5.3, where we can see that adding fluency ranking training data to a parse disambiguation training data with only the *rule* or the *rule* and *rule_in_context* features improves accuracy. However, the improvement was only significant when features were added to the model with only *rule* features (Table 5.4) Thus, in this scenario it is possible to improve parse disambiguation with a reversible model which includes fluency ranking data.

Model	**CA (%)**	**f-score (%)**
rule	86.16	86.54
rule/fluency	86.95	87.31
rule/rule_in_context	88.33	88.69
rule/rule_in_context/fluency	88.51	88.88

Table 5.3: Concept Accuracy scores and F-Scores in terms of named dependency relations for the baseline models that use only the *rule* and *rule_in_context* features versus a model that uses data from fluency ranking.

	rule	rule/rule_in_context
rule/fluency	$\mathbf{p = 0.0023}$	·
rule/rule_in_context/fluency	·	$p = 0.0362$

Table 5.4: Significance of the improvement in accuracy. Experiment-wise significance at 99% level is $p < 0.005$.

Figure 5.3 provides the results of the other baseline experiment. Here we compare the learning curve of the parsing model to that of the reversible model which was trained using all derivations in the fluency ranking training data in addition to the given percentage of parse disambiguation training data. We see that the addition of training data from fluency ranking makes the reversible model more effective than the directional model when less than 20% of the parse disambiguation training data is available (20% amounts 1440 sentences with the corresponding sample of derivations). From this, we can conclude that training data from fluency ranking can be used to improve the weight estimation of features that are active in parsing and generation.

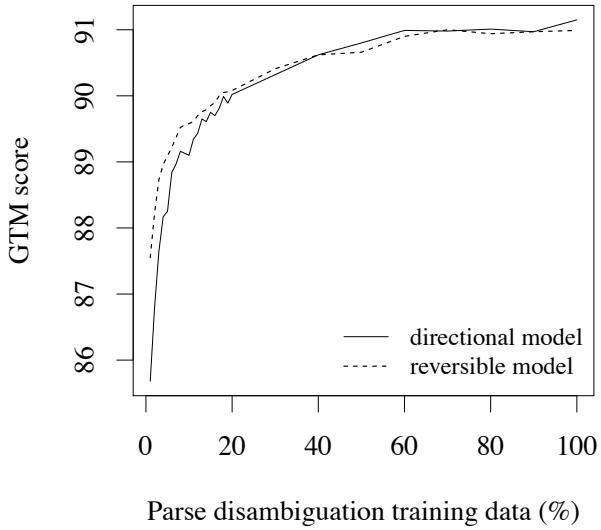


Figure 5.3: Learning curves of the parsing model and the reversible model when data from fluency ranking is immediately available.

5.7.2 Fluency ranking

Table 5.5 compares the reversible model with a directional fluency ranking model. As the table indicates, the reversible model outperforms the directional model, although the difference is not significant at $p < 0.001$.

Model	**Best match (%)**	**GTM**
Random	18.26	0.5539
Oracle	100.00	0.8662
Fluency	51.33	0.7219
Reversible	52.19	0.7252

Table 5.5: General Text Matcher scores and best match percentages for the fluency ranking-specific model versus the reversible model.

In Figure 5.4 we compare the learning curve of the directional fluency ranker to that of the reversible model which has the given percentage of fluency ranking training data available and all the training data from parse disambiguation. We can see that models that were trained using a small amount of training data perform better when training data for parse disambiguation is added. Also,

the addition of this training data never impacts performance negatively. On the contrary, we even see a mild improvement of performance when using a reversible model.

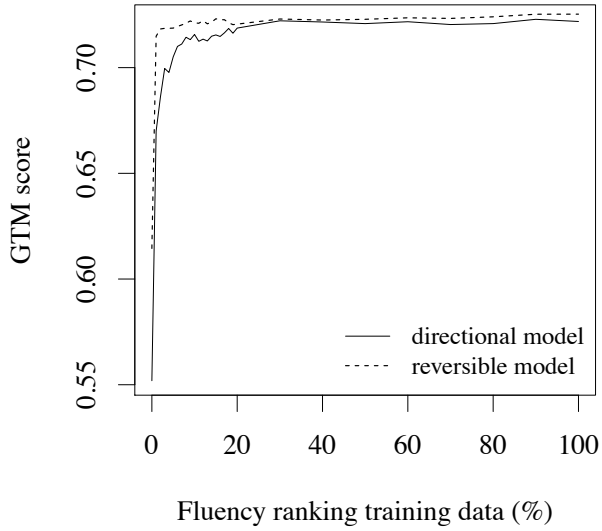


Figure 5.4: Learning curves of the fluency ranking model and the reversible model when data from parse disambiguation is immediately available.

To show that the reversible model actually profits from mutually shared features, we conducted an experiment where the fluency ranker only has access to the auxiliary trigram distributions as features in the training data. We train two models, the first using just this data, and the second adding parse disambiguation training data. The results for this experiment are shown in table 5.6. Adding information via parse disambiguation training data results in a considerable gain in accuracy (significant at $p < 0.01$). The resulting model performs almost as well as the directional fluency ranking model shown in table 5.5. This shows that preferences learned from parse disambiguation data are practically useful in fluency ranking.

5.8 Conclusion

We proposed reversible stochastic attribute-value grammars as an alternative to directional stochastic attribute-value grammars. This framework is based

Model	Best match (%)	GTM
word/tag trigrams	43.62	0.6945
word/tag trigrams + parse	48.74	0.7152

Table 5.6: General Text Matcher scores and best match percentages for the fluency ranking baseline model versus the baseline model with data from parse disambiguation.

on the observation that syntactic preferences are shared between parse disambiguation and fluency ranking. The integration of knowledge from parse disambiguation and fluency ranking data may be beneficial to tasks which combine aspects of parsing and generation, such as paraphrasing.

We have also shown that this framework is not purely of theoretical interest. In our experiments, reversible models obtain similar accuracy compared to directional models, and actually lead to increased accuracy in baseline models — small models in terms of features perform better if information from other tasks can be integrated. For instance, our baseline fluency ranking model had a considerably higher accuracy after complementing it with data from parse disambiguation.

However, there is an open question: given that reversible models also use features that are specific to parsing or generation, there is the possibility that the model is trained to rely on these directional features. If this is true, the premise that preferences are shared between parse disambiguation and fluency ranking does not hold, even though it may appear so in the results discussed in this chapter. In Chapter 7 we will revisit RSAVG and show using feature selection that reversible models do indeed rely on features that are active in both directions.

Chapter 6

Feature selection

6.1 Introduction

In Chapters 4 and 5 we discussed directional and reversible models for parse disambiguation and fluency ranking. In such models, the most interesting aspects of a derivation are described using features. A parameter estimator estimates a weight for each feature, so that we can estimate the probability of a derivation given the input of parsing or generation.

To achieve a high accuracy in such tasks, it is attractive to capture as many aspects of the derivation as possible. For instance, one could enumerate derivation tree fragments in varying sizes and forms, using very general templates. This path is followed in previous works, such as Velldal et al. [2004]. The advantage of this approach is that it requires little human labor and generally gives good ranking performance for languages with less free word orders, such as English. However, the generality of templates leads to huge models in terms of the number of features. For instance, in our experiments with the templates described by Velldal [2008], about half a million features were extracted using the training data described in previous chapters. Such models are very opaque, giving little understanding of strong discriminators in fluency ranking or parse disambiguation. Practically, the use of such large feature sets with no further selection may also be inconvenient in terms of storage space and training time. Feature selection can be applied to make such models compact and more understandable.

In this chapter, we discuss various feature selection methods that were (where necessary) modified for ranking tasks. These methods are then compared based on the performance of the features that they select. Finally, we

use the best method to give an analysis of the most effective features in fluency ranking and parse disambiguation.

6.2 Feature selection methods

Feature selection is a procedure that attempts to extract a subset of discriminative features $S \subset F$ from a set of features F, such that a model using S performs comparable to a model using F and $|S| \ll |F|$.

As discussed in de Kok [2010], a good feature selection method should reduce or eliminate three kinds of redundancies in feature sets:

- **Inactive:** Some features rarely change value within a context. Such features are not effective in general at discriminating good derivations from their competition.
- **Overlap:** Features can be overlapping. In that case, they may be effective in themselves, but using such functions in conjunction may not improve accuracy.
- **Noise:** Features may have values that do not correlate with the quality of the derivation. Such features are said to be *noisy*.

Also, since we want to analyze the relative effectiveness of features, a feature selection method should also provide a ranking of features by their discriminative power.

In the remainder of this section, we will describe six feature selection methods: (1) frequency-based selection; (2) correlation-based selection; (3) selection using an ℓ_1 prior; (4) grafting; (5) grafting-light; and (6) gain-informed selection. All of these selection methods, except the first two, are based on maximum entropy modeling. Since we augmented attribute-value grammar with conditional maximum entropy models, it is attractive to perform feature selection within that framework as well.

6.2.1 Frequency-based selection

Frequency-based selection counts for each feature f the number of inputs where a feature is active [van Noord, 2004]. In other words, it counts the number of inputs in the training data ($x \in X$) where there are at least two derivations d_1, d_2, such that $f(x, d_1) \neq f(x, d_2)$. The features are then sorted in descending order based on these counts. The resulting list provides the order in which features are selected.

6.2.2 Correlation-based selection

While frequency-based selection helps in selecting features that are active, it cannot account for feature overlap, nor does it exclude noisy features. Active features that have a strong correlation to features that were selected previously, are still added.

To detect feature overlap, we propose a simple correlation-based method that calculates the correlation of a candidate feature and a previously selected feature. If a candidate feature has a high-correlation with a previously selected feature, it is excluded. To estimate Pearson's correlation of two features, we calculate the sample correlation coefficient,

$$r_{f_1,f_2} = \frac{\sum_{x \in X, d \in \Omega(x)} (f_1(x,d) - \bar{f}_1)(f_2(x,d) - \bar{f}_2)}{(n-1) s_{f_1} s_{f_2}} \tag{6.1}$$

where $\bar{f}_x$ is the average feature value of f_x, and s_{f_x} is the sample standard deviation of f_x.

Of course, inter-feature correlation can only be used to detect overlap, since it does not provide a ranking of features by their effectiveness. Since frequency-based selection provides a ranking of how 'active' features are, we use frequency-based selection as described in the previous section to make an initial ranking of the features. We then use the correlation metric above to exclude features that overlap with features with a higher rank.

Since correlation-based selection compares features one by one, it cannot detect overlap in cases where some selected features jointly overlap strongly with a candidate feature. It is also not able to detect noisy features. It could, however, be extended to exclude noisy features by calculating the correlation between feature values and derivation scores — if such a correlation does not exist, the feature is noisy. Despite these disadvantages, correlation-based selection is a very simple technique.

6.2.3 $\ell 1$ regularization

During the training of maximum entropy models, regularization is often applied to avoid unconstrained feature weights and overfitting (Section 4.3.8). If $F(\theta)$ is the objective function that is minimized during training, a regularizer $\omega_q(\theta)$ is added as a penalty for extreme weights [Tibshirani, 1996]:

$$C(\theta) = F(\theta) + \omega_q(\theta) \tag{6.2}$$

Again, the regularizer has the following form, where q is a non-negative integer:

$$\omega_q(\theta) = \lambda \sum_{i=1}^{n} |\theta_i|^q \tag{6.3}$$

The ℓ_2 regularizer (q = 2) was discussed in Section 4.3.8. Another commonly-used setting is $q = 1$, which gives an ℓ_1 regularizer. This regularizer amounts to applying a double-exponential prior distribution with $\mu = 0$. Since the double-exponential puts much of its probability mass near its mean (which is $\mu = 0$ for feature weights), the ℓ_1 regularizer has a tendency to force weights towards zero, providing integral feature selection and while simultaneously preventing unbounded weights. Increasing λ strengthens the regularizer, and forces more feature weights to be zero.

Given an appropriate value for λ, ℓ_1 regularization can exclude features that change value infrequently, as well as noisy features. However, it does not guarantee the exclusion of overlapping features, since the weight mass can be distributed among overlapping features. ℓ_1 regularization also does not fulfill a necessary characteristic for the present task, in that it does not provide a ranking based on the discriminative power of features. However, ℓ_1 selection is the basis of the two other selection methods that we will discuss, *grafting* and *grafting-light*.

6.2.4 Grafting

Grafting [Perkins et al., 2003] adds incremental feature selection during the training of a maximum entropy model. The selection process is a repetition of two steps: 1. a gradient-based heuristic selects the most promising feature from the set of unselected features Z, adding it to the set of selected features S; and 2. a full optimization of weights is performed over all features in S. These steps are repeated until a stopping condition is triggered.

During the first step, the gradient of each unselected feature $f_i \in Z$ is calculated with respect to the model p_S, that was trained with the set of selected features, S:

$$\left| \frac{\partial L(\theta_S)}{\partial \theta_i} \right| = E_{p_S}(f_i) - E_{\tilde{p}}(f_i) \tag{6.4}$$

The feature with the largest gradient is removed from Z and added to S.

The stopping condition for grafting integrates the ℓ_1 regularizer in the grafting method. Note that when ℓ_1 regularization is applied, a feature is only

included (has a non-zero weight) if its penalty is outweighted by its contribution to the reduction of the objective function. Consequently, only features for which $\left|\frac{\partial L(w_S)}{\partial w_i}\right| > \lambda$ holds are eligible for selection. This is enforced by stopping selection if for all f_i in Z

$$\left|\frac{\partial L(\theta_S)}{\partial w_i}\right| \leq \lambda \tag{6.5}$$

The full grafting algorithm is shown in Algorithm 10.

Algorithm 10 The grafting selection algorithm. ESTIMATE is the parameter estimation function.

```
S ← {}
while do
    θ_S ← ESTIMATE(S)
    maxGradient ← 0
    maxF ← undefined
    for f ∈ Z do
        gradient ← E_pS(f) − E_p̃(f)
        if gradient > maxGradient then
            maxGradient ← gradient
            maxF ← f
        end if
    end for
    if maxGradient ≤ λ then
        BREAK
    end if
    S ← S ∪ maxF
    Z ← Z − maxF
end while
```

Although grafting uses ℓ_1 regularization, its iterative nature avoids selecting overlapping features. For instance, if f_1 and f_2 behave identically, and f_1 is added to the model p_S, $\left|\frac{\partial L(\theta_S)}{\partial \theta_2}\right|$ will amount to zero.

Performing a full optimization after each selected feature is computationally expensive. Riezler and Vasserman [2004] observe that during the feature step selection a larger number of features can be added to the model (n-best selection) without a loss of accuracy in the resulting model. However, this so-called *n-best grafting* may introduce overlapping features.

6.2.5 Grafting-light

The grafting-light method [Zhu et al., 2010] uses the same selection step as grafting, but improves grafting performance-wise by applying one iteration of gradient-descent during the optimization step rather than performing a full gradient-descent. As such, grafting-light gradually works towards the optimal weights, while grafting always finds the optimal weights for the features in S during each iteration.

Since grafting-light does not perform a complete gradient-descent, an additional stopping condition is used, because the model may still not be optimal, even if no features are eligible for selection. This condition requires that the change in value of the objective function incurred by the last gradient-descent is smaller than a predefined threshold.

6.2.6 Gain-informed selection

Grafting and grafting-light use the gradient of candidate features to estimate their effectiveness. From a theoretical perspective this is not completely satisfying. Gradient-based methods use the gradient as an indirect measure for the contribution that a feature would provide. It is certainly true that a feature with a gradient of zero would not contribute to a model, but for other features there may be factors that influence the gradient, such as the magnitude of the feature values.

Another class of methods aims to estimate the contribution of a feature more directly by calculating its effect on the objective function. In other words, the contribution (or so-called *gain*) of a candidate feature $f_i \in Z$ given the set of selected features is $L(\theta_S) - L(\theta_{S \cup f_i})$. The selection algorithm, which is outlined in Algorithm 11, is then very similar to grafting, except that the feature with the highest gain is chosen during a selection step, rather than the feature with the highest gradient.

The problem with the naive implementation of this selection method is that we have to compute $L(\theta_{S \cup f_i})$ for each $f_i \in Z$ during each selection step. Calculating $\theta_{S \cup f_i}$ requires the estimation of θ_{f_i}. In other words, during each selection step, we are required to do parameter estimation of $|Z|$ models. This is computationally intractable for any non-trivial feature or training set.

Berger et al. [1996] eliminate the aforementioned inefficiency by assuming that the weights of the features in S do not change as a result of adding a new feature. Under this assumption θ_i, the weight of f_i, can be estimated using a simple line search method. This improves the performance of the method considerably, such as Newton's iterative root finding method [Berger et al., 1996]. They modify the selection procedure so that the full parameter

Algorithm 11 General schemata of gain-informed selection algorithms. ESTIMATE is the parameter estimation function, $L(\theta)$ the log-likelihood of the model with feature weights θ, and Z is the set of unselected features.

```
S ← {}
while do
    θ_S ← ESTIMATE(S)
    maxGain ← 0
    maxF ← undefined
    for f ∈ Z do
        θ_{S∪f} ← ESTIMATE(S ∪ f)
        gain ← L(θ_S) − L(θ_{S∪f})
        if gain > maxGain then
            maxGain ← gain
            maxF ← f
        end if
    end for
    if maxGain < threshold then
        BREAK
    end if
    S ← S ∪ maxF
    Z ← Z − maxF
end while
```

estimation to obtain a feature's weight is replaced by a simple line search. This weight is used to calculate the gain of the feature and will become its weight if the feature is selected.

Zhou et al. [2003] point out that, despite this optimization, the weight of every candidate feature is still re-estimated during each selection step, although this may be irrelevant for many features. Zhou et al. [2003] observes that while the gains of candidate features may decrease as a result of the selection of a feature, they rarely increase. This observation can be exploited in the following manner: if we have a list of (remaining) candidate features ordered by their gains, we can select the topmost feature unless its gain decreased by such an amount that it is lower than the second feature on that list.

In our experiments, we apply the selection method of Zhou et al. [2003], with a small modification to allow for arbitrary feature values [de Kok, 2010]. In short, the recursive function used in the former work to update the vectors of unnormalized probabilities and normalizers assumed that features have a

binary value (0 or 1). This method is outlined in Algorithm 12.

Algorithm 12 Gain-informed selection algorithm using selective gain computation [Zhou et al., 2003]. ESTIMATE_SIMPLE performs a line search to estimate the weight of one feature, GAIN_SUCCEEDING(O, f) gives the gain of the feature succeeding f in O, and REINSERT$(O, f, gain)$ reinserts the feature f in O such that O is still sorted.

```
S ← {}
gains ← COMPUTE_GAINS(S, Z)
O ← SORT_DESCENDING(gains)
repeat
    for f ∈ O do
        θ_{S∪f} ← ESTIMATE_SIMPLE(S, f)
        gain ← L(θ_S) − L(θ_{S∪f})
        if gain > GAIN_SUCCEEDING(O, f) then
            S ← S ∪ maxF
            O ← O − maxF
            BREAK
        end if
        REINSERT(O, f, gain)
    end for
until FIRST(O)< threshold
```

Since this selection method uses the gain of a feature in its selection criterion, it excludes noisy and infrequent features. Overlapping features are also excluded since their gain diminishes after selecting one of the overlapping features.

6.3 Evaluation methodology

The goal of our evaluation of feature selection method is to find out which feature selection method performs the best when picking a very small subset of features. This method will be used in Section 6.5 to provide an analysis of the twenty most effective features in parse disambiguation and fluency ranking. Consequently, we are interested to see how good each method performs 'over time' as we allow the method to add features to the model. To this end, we let each method, with the exception of selection with an ℓ_1 prior, select 200 features. We will then use the ordered list of features provided by each method to train increasingly large models. At the highest granularity, this gives us 200

models per selection method with 1 to 200 features, that we can then evaluate. The result of this evaluation will give the performance of each method as it selects more features.

The training and evaluation sets will be familiar. We use the newspaper part of the Eindhoven corpus to let each method select the most discriminative features and to train the maximum entropy models using these features. The PPH corpus is again used for evaluation of the models. Both corpora were extensively discussed in Chapters 4 and 5. The methods are evaluated separately for parse disambiguation and fluency ranking.

The correlation selection method is used, so that features that have a correlation of 0.9 with a previously selected feature, are skipped. In de Kok [2011] we observed that gain-informed selection performed far worse in our setup than the grafting-based methods. This performance degradation compared to de Kok [2010] was caused by a change in the estimation of $\tilde{p}(x, y)$ from *weighted inputs* to *uniform inputs with binary events* (Section 4.3.6). For this reason, we use weighted inputs for feature selection using the gain-informed method. The feature weights in the resulting feature sets are estimated in the same manner as the other feature selection methods.

6.4 Results

6.4.1 Parsing

Figure 6.1 shows the performance of the five rankers in parse disambiguation. Figure 6.1(a) compares frequency, correlation and gain-informed selection. We can clearly see that gain-informed selection consistently outperforms the frequency and correlation selection methods. This clearly shows that excluding features in a smart manner, based on overlap and noisiness, pays off. Among the methods that are not based on maximum entropy modeling, the correlation method performs slightly better than frequency selection. But with its restriction to one-to-one overlap detection, it fails to impress. Figure 6.1(b) compares the maximum entropy-based methods. All three methods perform comparably, but overall, grafting results in the best models.

Besides comparing the selection methods individually, it is also interesting to observe the general trend — after selecting around 80 features the line flattens for the maximum entropy-based methods. After selecting 80 features, the model constructed using grafting has a concept accuracy of 89.59%. In comparison, the uniform model has an accuracy of 75.63% and the complete parse model 91.14%. In other words, a relatively small number of features settles most ambiguities. The long tail of remaining features are far less dis-

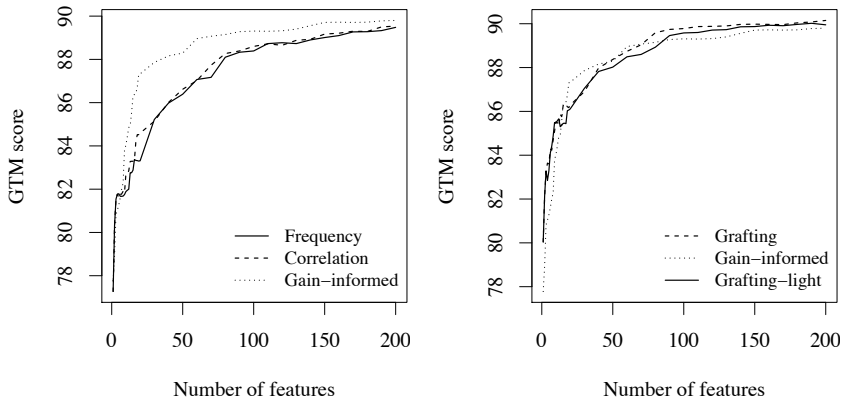


Figure 6.1: Performance of parse disambiguation models created with the (a) frequency, correlation, and gain-informed selection methods; and (b) maximum entropy, grafting, and grafting-light selection methods. The gain-informed, grafting, and grafting-light methods outperform the frequency and correlation based selection by a wide margin. Among the maximum entropy-based methods, the Grafting method performs the best overall.

criminative, but certainly important to the performance of the disambiguation component. However to get a grasp of the most discriminative features in parsing, we have to analyze a relatively small number of features.

6.4.2 Fluency ranking

The performance of selection methods in fluency ranking is shown in Figure 6.2. Figure 6.2(a) compares the frequency, gain-informed and grafting-light selection methods. Not much difference can be observed between the grafting-light and frequency selection methods. Both methods outperform gain-informed selection most of the time. Figure 6.2(b) compares the frequency, correlation, and grafting methods. Again, the grafting method performs the best, while the correlation method performs comparably to the frequency method.

In general, we see that there is much less differentiation between the selection methods than in parse disambiguation. We believe this is caused by the fact that the fluency ranking feature set contains fewer redundancies than that of parse disambiguation. For instance, parse disambiguation uses a large

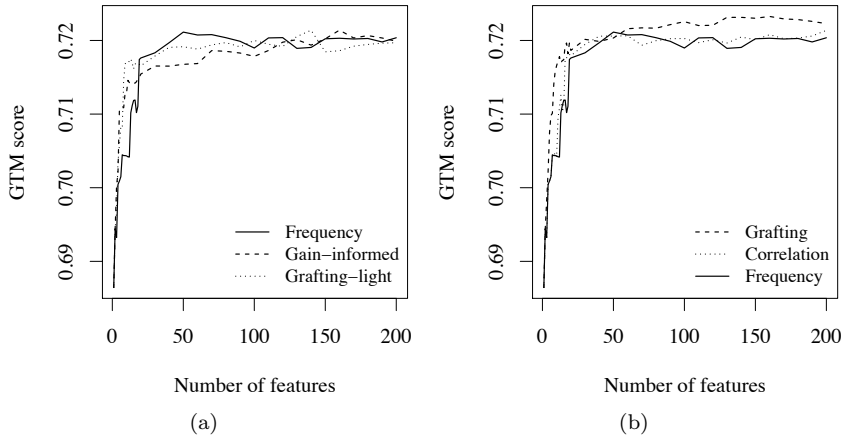


Figure 6.2: Performance of fluency ranking models created with the (a) frequency, gain-informed, and grafting-light selection methods; and (b) frequency, correlation, and grafting selection methods. Grafting outperforms the other selection methods.

number of lexical frame and dependency relation features (Section 5.6.1).

As in parse disambiguation, a very small number of discriminative features is required to construct an effective model. The lines of the selection methods flatten at around 50 to 60 features. For instance, the grafting model achieves a GTM score of 0.7203 after selecting 60 features, compared to 0.5519 for the uniform model and 0.7219 for the complete fluency ranking model (Section 4.7.2). Again, to get insights into the most discriminative features, one only needs to analyze a relatively small amount of features.

6.5 Discriminative features

In the previous section, we have established that grafting was the most effective feature selection method for our particular task and data sets. In this section, we use grafting to extract the twenty most discriminative features in parse disambiguation and fluency ranking, and provide an analysis of these features.

6.5.1 Parse disambiguation

Table 6.1 shows the twenty most discriminative features in parse disambiguation. The second column enumerates the polarities of the features, a polarity indicates whether a feature is an indicator for a good (+) or bad (-) derivation. We now provide a description of these features by category.

Rank	Polarity	Feature
1	+	word_frame_distribution
2	-	rule(np_n)
3	+	rule_in_context(np_det_n,2,n_n_pps)
4	-	parallel_conjunction_depth
5	-	rule_in_context(vp_mod_v,3,vproj_vc)
6	-	frame(adj)
7	-	rule_in_context(vp_arg_v(np),2,vproj_vc)
8	+	rule_in_context(vp_mod_v,2,optpunct(e))
9	+	rule(n_adj_n)
10	+	tag_dependency_tag(prep,hd/pc,verb)
11	+	syntactic(subj_np_topic)
12	+	word_tag_dependency_tag(van,prep,hd/mod,noun)
13	+	p1(par)
14	-	rule(optpunct(e))
15	+	rule_in_context(top_start_xp,1,max_xp(root))
16	+	rule_in_context(vp_arg_v(np),1,np_det_n)
17	-	frame(verb(intransitive))
18	+	tag_dependency_tag(noun,hd/su,verb)
19	+	tag_dependency_tag(prep,hd/mod,verb)
20	-	rule_in_context(np_n,1,l)

Table 6.1: The twenty most discriminative features in parse disambiguation.

Lexical frames Various features assist in the selection of word tags in parse disambiguation. Many words can be read in more than one manner, for instance, the word *kiezen* can be read as a finite or infinite verb ('to choose') that can be transitive or intransitive. It can also be a nominalized verb (*het kiezen*) or the plural of the noun *kies* 'molar'. The most discriminative feature in parse disambiguation, *word_frame_distribution*, provides an auxiliary distribution that estimates the probability of a tag given a word. Feature 6 penalized the reading of a word as an adjective, and feature 17 intransitive verb readings.

Topicalization As expected, one of the features (11) encodes the preference for topicalized NP subjects. It may perhaps be surprising that none of the features indicates the possibility of modifier topicalization (Section 4.7.4). However, in a sentence that starts with a modifier, followed by a verb, this ambiguity is not present. Feature 11 specifically settles ambiguity for the case where a sentence starts with an NP. We will see that in fluency ranking, where there is a choice of fronting the subject, direct object, and modifiers, that there is a discriminative feature describing modifier fronting.

Modification One frequent ambiguity in sentences is the attachment of prepositional phrases, which can often be attached to a noun or a verb. Feature 3 indicates a preference for noun attachment. Feature 10 states that if a preposition combines with a verb, it is preferred that it modify the verb with a *pc* (prepositional complement) relation. Feature 12 indicates the preference that modifiers that are headed by the word *van* 'of' should be attached to a noun. Feature 5 indicates readings with left-adjoining modifiers, unless the modifier and the verb are not separated by punctuation (Feature 8).

Parallelism in conjunctions In parse disambiguation, parallelism in conjunctions is preferred. Parallelism requires that the conjuncts in a conjunction are analyzed using the same grammar rule and that the conjuncts have the same (maximal) tree depth. The value of feature 4 is the difference of depth in conjunctions. This feature has a negative polarity, because it is indicative of a derivation that is not preferred. If a derivation has a high value for this feature, it means that the analysis has a conjunction that is not parallel with respect to the depth of conjuncts.

The second feature (13) has a non-zero value if the conjuncts of a conjunction are indeed constructed using the same grammar rule, or are all lexical. As such, it is a positive indicator of a good derivation.

Other Two features (2, 20) state the preference that analyses with bare noun NPs should not be preferred. Feature 18 prefers readings where verbs have a noun subject. This feature could reject readings with two characteristics: (1) sentences that have a subject that is not headed by a noun and (2) sentences that do not have a subject at all. For both cases, there are examples that they occur, although infrequently. For instance: (1) *Het beschamende daarbij is dat hij loog.* 'The embarasing about-that is that he lied.' and (2) *Voorgesteld wordt het assortiment vleeswaren uit te breiden.* 'Proposed is the collection meat to expand.' Finally, feature 9 states the preference for reading the word before the noun as an adjective.

6.5.2 Fluency ranking

Table 6.2 shows the twenty most discriminative features in fluency ranking. We will, again, discuss the features by category.

Rank	Polarity	Feature
1	+	word_trigram_distribution
2	+	tag_trigram_distribution
3	-	rule(vp_v_mod)
4	+	rule(optpunct(e))
5	+	syntactic(subj_np_topic)
6	+	rule_in_context(vp_mod_v,3,vproj_vc)
7	+	rule_in_context(vpx_vproj,1,vp_arg_v(np))
8	+	rule_in_context(non_wh_topicalization(modifier),3,imp)
9	+	rule_in_context(vp_arg_v(pp),2,vproj_vc)
10	+	rule_in_context(vp_arg_v(pred),2,vproj_vc)
11	+	rule_in_context(vp_arg_v(np),2,vproj_vc)
12	+	rule_in_context(vp_arg_v(np),2,vp_arg_v(np))
13	+	rule_in_context(non_wh_topicalization(modifier),1,mod2)
14	+	middle_field(mcat_adv,np(acc,noun))
15	+	rule_in_context(vp_arg_v(np),2,vp_mod_v)
16	+	middle_field(mcat_pp,np(acc,noun))
17	+	rule(n_adj_n)
18	+	rule_in_context(non_wh_topicalization(np),1,np_det_n)
19	-	syntactic(non_subj_np_topic)
20	+	rule_in_context(vp_mod_v,1,mod1a)

Table 6.2: The twenty most discriminative features in fluency ranking.

Trigram models The trigram models are the most important features (1, 2). This confirms the results outlined in Table 4.9 of Chapter 4 — language models form a solid baseline for fluency ranking models.

Topicalization Various features express preferences with respect to constituent fronting/topicalization. Foremost, feature 5 expresses a preference for realizations that have a topicalized NP subject. As discussed in Chapter 4, direct object fronting is allowed in Dutch, but not preferred. This is expressed by feature 19, which discredits realizations that have a topicalized NP that is not the subject. Modifier topicalization also occurs frequently in Dutch (Section 4.7.4), features 8 and 13 express this preference. Feature 18 is also related

to topicalization, it expresses the preference that fronted noun phrases were constructed using the $NP \rightarrow Det\ N$ rule.

Modifiers Four features (3, 6, 15, 20) are related to modifier adjoining. Feature 3 indicates that realizations with right-adjoining modifiers are not preferred. The other features describe instances of left-adjoining of modifiers, which are indicators of a good derivation. For instance, the model would prefer *zij heeft met de hond gelopen* 'she has with the dog walked' over *zij heeft gelopen met de hond* 'she has walked with the dog'.

Features 9 to 12 indicate the preferences of left-adjoining of phrasal arguments, such as prepositional phrases (9) and noun phrases (11).

Ordering in the middle field Two features (14, 16) describe ordering preferences in the middle field. Feature 14 prefers that an adverbial is positioned before an accusative noun phrase that is headed by a noun. For instance, *hij heeft gisteren kaas gekocht* 'he has yesterday cheese bought' is preferred over *hij heeft kaas gisteren gekocht*. Feature 16 prefers that a prepositional phrase is positioned before an accusative noun phrase headed by a noun. For instance, *wij hebben met plezier kaas gegeten* 'we have with pleasure cheese eaten' is preferred over *wij hebben kaas met plezier gegeten*.

Other Feature 4 expresses the preference to fill optional punctuation in rules with an empty punctuation token.

6.6 Conclusion

In this chapter, we have discussed and compared five different feature selection methods that provide an ordered list of features by their effectiveness. We saw that the techniques that used the gradients of candidate features performed the best. Although the gradient only gauges the contribution of a feature indirectly, gradient-based selection methods do not require additional assumptions to make them computationally tractable.

We then used a gradient-based grafting method, to select the twenty most discriminative features in parse disambiguation and fluency ranking. The role of many of these features can be understood well through simple inspection. What we see is that a small number of specific features such as fronting, word-level disambiguation, parallelism in conjunctions, specific orderings in the middle field, and modifier adjoining settle the most frequent ambiguities. Then, there

is a long tail of features that settle more specific cases. But we hope to have isolated the most distinctive phenomena in parse and realization selection.

In the next chapter, we will apply feature selection to reversible models, with a related goal: understanding what proportion of the features are truly reversible.

Chapter 7

Discriminative Features in RSAVG

7.1 Introduction

Reversible stochastic attribute-value grammars (Chapter 5) provide an elegant framework that fully integrates parsing and generation. The most important contribution of this framework is that it uses one conditional maximum entropy model for fluency ranking and parse disambiguation. In such a model, the probability of a derivation d is conditioned on a set of input constraints c that restrict the set of derivations allowed by a grammar to those corresponding to a particular sentence (parsing) or abstract representation (generation).

Reversible stochastic-attribute grammars rest on the premise that preferences are shared between language comprehension and production. For instance, in Dutch, subject fronting is preferred over direct object fronting (Chapter 4). If models for parse disambiguation and fluency ranking do not share preferences with respect to fronting, it would be difficult for a parser to recover the abstract representation that was the input to a generator.

Reversible models incorporate features that are specific to parse disambiguation and fluency ranking, as well as features that are active in both tasks (*task-independent features*). We have shown in the previous chapter, through feature selection and analysis, that task-independent features are indeed used in parse disambiguation and fluency ranking models. However, since reversible models assign just one weight to each feature regardless the task, one particular concern is that much of their discriminatory power is provided by task-specific

features. If this is true, the premise that similar preferences are used in parsing and generation does not hold.

In this chapter, we aim to verify that reversible models do rely on features used both in parsing and generation. We will do this by isolating the most discriminative features of reversible models through feature selection, and make a quantitative and qualitative analysis of these features.

To find the most effective features of a model, we use the grafting selection method that was described in the previous chapter. For the training and evaluation of the reversible model, we use exactly the same data as described in Chapter 5. In Section 7.2, we will introduce a quantitative method for comparing directional and reversible models through feature selection. We will then apply this method to show that reversible models indeed rely on features used in both directions. In Section 7.3 we provide a qualitative analysis of the twenty most discriminative features in our reversible model. This analysis shows that features used in both directions are also prominent among the most discriminative features in reversible models.

7.2 Quantitative analysis of reversible models

To gauge how important features are to a model, we have to measure their contribution to that model. In the general case, estimating the contribution of a feature is difficult. If we have a maximum entropy model with overlapping features, evaluating a model with and without that feature, does not provide a good estimation of its contribution. In such a case, the removal of a feature may barely impact the performance of the model. However, if its overlapping features are also removed, the feature could improve the model.

One of the goals of feature selection was to limit the set of features such that as many redundancies, including overlap, are removed. An iterative selection method can remove overlap, and even if there is overlap, we consider features that were selected earlier to be more important than features that were selected later. Using an iterative feature selection method, we can estimate the contribution of a feature to a model as follows:

$$c(f_i) = \frac{e(p_{0..i}) - e(p_{0..i-1})}{e(p_{0..n}) - e(p_0)} \tag{7.1}$$

Here, $p_{0..i}$ is a model trained with the i most discriminative features, p_0 is the uniform model, $e(p_{0..i})$ the effectiveness of $p_{0..i}$ measured via the average GTM score, and n is the eventual number of features that we will calculate the contribution for.

For the quantitative analyses of highly discriminative features, we extract the 300 most effective features of the fluency ranking, parse disambiguation, and reversible models using grafting. We choose this number, because we have seen in Section 4.7 that the improvement in performance of the rankers has essentially flattened out after selecting that number of features.

We cannot directly compare the feature lists of the directional models and the reversible model. The reason is that many of the candidate features are overlapping and show subtle interactions, features can easily be substituted by other features that have the same role. For example, in the set of most discriminative features in parse disambiguation, there is a feature that disprefers derivations that have an adjective modifying a verb. However, this feature does not occur at all in the set of most discriminative features in the reversible model. Instead, there is a feature that promotes the use of an adjective as the modifier of a noun.

Another question is if we could compare the set of selected features directly, whether this would be useful. By definition, the set of features selected for the reversible model will be different from that of the directional models, since the reversible model has to accommodate both parse disambiguation and fluency ranking. If the reversible model is truly reversible, the number and contribution of direction-specific features will be smaller. If the reversible model is not truly reversible, the number and contribution of features that are active in both directions will be smaller. Consequently, to test the hypothesis that the reversible model is truly reversible, we have to show that the latter is not true.[1]

We divide each feature into one of five classes: *dependency* (enumeration of dependency triples), *lexical* (readings of words), *n-gram* (word and tag trigram auxiliary distributions), *rule* (identifiers of grammar rules), and *syntactic* (abstract syntactic features). Of these classes, the *rule* and *syntactic* features are active during both parse disambiguation and fluency ranking.

For the quantitative analysis, we train a model for each selection step, for models having the 0 to 300 most discriminative features. Using these models we compute $c(f_i)$, for each feature f_i in among the 300 features. We then simply calculate the per-class distribution by summing the contributions of the features of each particular class.

[1] Computing a statistic such as the correlation of feature contributions of directional models and reversible models would show whether directional models and reversible models use the same features with approximately the same contribution, whereas we are interested in the actual per-class shift that occurs. In both extremes (the reversible model uses more reversible features or the reversible model uses more directional features), we would actually expect a low correlation.

7.2.1 Parse disambiguation

Table 7.1 provides class-based counts of the 300 most discriminative features for the parse disambiguation and reversible models. Since the n-gram features are not active during parse disambiguation, they are not selected for the parse disambiguation model. All other classes of features are used in the parse disambiguation model. The reversible model uses all classes of features.

Class	**Directional**	**Reversible**
Dependency	93	84
Lexical	24	24
N-gram	0	2
Rule	156	154
Syntactic	27	36

Table 7.1: Per-class counts of the best 300 features according to the grafting method, for the directional parsing model and the reversible model.

Contributions per feature class in parse disambiguation are shown in Table 7.2. In the directional parse disambiguation model, parsing-specific features (*dependency* and *lexical*) account for 55% of the improvement over the uniform model.

In the reversible model, there is a shift of contribution towards task-independent features. When applying this model, the contribution of parsing-specific features to the improvement over the uniform model is reduced to 45.79%.

We can conclude from the per-class feature contributions in the directional parse disambiguation model and the reversible model, that the reversible model does not put more emphasis on parsing-specific features. Instead, the opposite is true: task-independent features are more important in the reversible model than in the directional model.

Class	**Directional**	**Reversible**
Dependency	21.53	13.35
Lexical	33.68	32.62
N-gram	0.00	0.00
Rule	37.61	47.35
Syntactic	7.04	6.26

Table 7.2: Per-class contribution to the improvement of the model over the base baseline in parse disambiguation, for the directional fluency ranking model and the reversible model.

7.2.2 Fluency ranking

Table 7.3 provides class-based counts of the 300 most discriminative features of the fluency ranking and reversible models. During fluency ranking, dependency features and lexical features are not active.

Class	Directional	Reversible
Dependency	0	84
Lexical	0	24
N-gram	2	2
Rule	181	154
Syntactic	117	36

Table 7.3: Per-class counts of the best 300 features according to the grafting method.

Table 7.4 shows the per-class contribution to the improvement in accuracy for the directional and reversible models. Since the dependency and lexical features are not active during fluency ranking, it may come as a surprise that their contribution is negative in the reversible model. Since they are used for parse disambiguation, they have an effect on weights of task-independent features. This phenomenon did not occur when using the reversible model for parse disambiguation, because the features specific to fluency ranking (n-gram features) were selected as the most discriminative features in the reversible model. Consequently, the reversible models with one and two features were uniform models from the perspective of parse disambiguation.

Class	Directional	Reversible
Dependency	0.00	-4.21
Lexical	0.00	-1.49
N-gram	81.39	83.41
Rule	14.15	16.45
Syntactic	3.66	4.59

Table 7.4: Per-class contribution to the improvement of the model over the baseline in fluency ranking.

Since active features compensate for this loss in the reversible model, we cannot directly compare per-class contributions. To this end, we normalize the contribution of all positively contributing features, leading to table 7.5. Here, we can see that the reversible model does not shift more weight towards

task-specific features. On the contrary, there is a mild effect in the opposite direction here as well.

Class	Directional	Reversible
N-gram	81.39	79.89
Rule	14.15	15.75
Syntactic	3.66	4.39

Table 7.5: Classes giving a net positive distribution, with normalized contributions.

7.3 Qualitative analysis of reversible models

While the quantitative evaluation shows that task-independent features remain important in reversible models, we also want to get an insight into the actual features that were used. Since it is unfeasible to study the 300 best features in detail, we extract the 20 best features.

Grafting-10 is too course-grained for this task, since it selects the first 10 features solely by their gradients, while there may be overlap in those features. To get the most accurate list possible, we perform grafting-1 selection to extract the 20 most effective features. We show these features in table 7.6 with their polarities. The polarity indicates whether a feature is an indicator for a good (+) or bad (-) derivation.

We now provide a description of these features by category.

Word/tag trigrams The most effective features in fluency ranking are the n-gram auxiliary distributions (1, 3). The word n-gram model settles preferences with respect to fixed expressions and common word orders. It also functions as a (probabilistic) filter of archaic inflections and incorrect inflections that are not known to the Alpino lexicon. The tag n-gram model helps in picking out a sequence of part-of-speech tags that is plausible.

Frame selection Various features assist in the selection of proper subcategorization frames for words. This currently affects parse disambiguation mostly. There is virtually no ambiguity of frames during generation, and a stem/frame combination normally only selects one inflection. The most effective feature for frame selection is 2, which is an auxiliary distribution of words and corresponding frames based on a large automatically annotated corpus. Other effective

Rank	Polarity	Feature
1	+	word_trigram_distribution
2	+	word_frame_distribution
3	+	tag_trigram_distribution
4	-	rule(np_n)
5	+	rule_in_context(np_det_n,2,n_n_pps)
6	-	parallel_conjunction_depth
7	+	rule_in_context(vp_mod_v,3,vproj_vc)
8	-	rule_in_context(vp_arg_v(np),2,vproj_vc)
9	-	frame(adj)
10	+	rule_in_context(vp_mod_v,2,optpunct(e))
11	-	syntactic(non_subj_np_topic)
12	+	rule(n_adj_n)
13	+	tag_dependency_tag(prep,hd/pc,verb)
14	+	rule(optpunct(e))
15	+	word_tag_dependency_tag(van,prep,hd/mod,noun)
16	+	tag_dependency_tag(noun,hd/su,verb)
17	+	parallel_conjunction_construction
18	-	rule(vp_v_mod)
19	+	tag_dependency_tag(prep,hd/mod,verb)
20	-	frame(verb(intransitive))

Table 7.6: The twenty most discriminative features of the reversible model, and their polarities.

features indicate that readings as an adjective (9) and as an intransitive verb (20) are not preferred.

Modifiers Feature 5 indicates the preference to attach prepositional phrases to noun phrases. However, if a modifier is attached to a verb, we prefer readings and realizations where the modifier is left-adjoining rather than right-adjoining (7, 18, 19). For instance, *zij heeft met de hond gelopen* 'she has with the dog walked' is more fluent than *zij heeft gelopen met de hond* 'she has walked with the dog'. Finally, feature 15 gives preference to analyses where the preposition *van* is a modifier of a noun.

Conjunctions Two of the twenty most discriminative features involve conjunctions. The first (6) is a dispreference for conjunctions where conjuncts have a varying depth. In conjunctions, the model prefers derivations where all

conjuncts in a conjunction have an equal depth. The other feature 17 gives a preferences to conjunctions with parallel conjuncts — conjunctions where every conjunct is constructed using the same grammar rule.

Punctuation The Alpino grammar is very generous in allowing optional punctuation. An empty punctuation sign is used to fill grammar rule slots when no punctuation is used or realized. Two features indicate preferences with respect to optional punctuation. The first (10) gives preference to filling the second daughter slot of the *vp_mod_v* with the empty punctuation sign. This implies that derivations are preferred where a modifier and a verb are not separated by punctuation. The second feature 14 indicates a general preference for the occurrence of empty optional punctuation in the derivation tree.

Subjects/objects In Dutch, subject fronting is preferred over object fronting. For instance, *Spanje won de wereldbeker* 'Spain won the World Cup' is preferred over *de wereldbeker won Spanje* 'the World Cup won spain'. Feature 8 will in many cases contribute to the preference of having topicalized noun phrase subjects. It disprefers having a noun phrase left of the verb. For example, *zij heeft met de hond gelopen* 'she has with the dog walked' is preferred over *met de hond heeft zij gelopen* 'with the dog she has walked'. Feature 11 encodes the preference for subject fronting, by penalizing derivations where the topic is a non-subject noun phrase.

Other syntactic preferences The remaining features are syntactic preferences that do not belong to any of the previous categories. Feature 4 indicates a dispreference for derivations where bare nouns occur. Feature 12 indicates a preference for derivations where a noun occurs along with an adjective. Finally, feature 13 gives preference to the prepositional complement (*pc*) relation if a preposition is a dependent of a verb and lexical analysis shows that the verb can combine with that prepositional complement.

We can conclude from this description of features that many of the features that are paramount to parse disambiguation and fluency ranking are task-independent, modeling phenomena such as subject/object fronting, modifier adjoining, parallelism and depth in conjunctions, and the use of punctuation.

7.4 Conclusion

In this chapter we have used feature selection techniques for maximum entropy modeling to analyze the hypothesis that the models in reversible stochastic

attribute-value grammars use task-independent features. We used grafting to select the most effective features for parse disambiguation, fluency ranking, and reversible models. In our quantitative analysis we have shown that the reversible model does not put more emphasis on task-specific features. In fact, the opposite is true: in the reversible model task-independent features become more defining than in the directional models.

We have also provided a qualitative analysis of the twenty most effective features, showing that many of these features are relevant to both parsing and generation. Effective task-independent features for Dutch model phenomena such as subject/object fronting, modifier adjoining, parallelism and depth in conjunctions, and the use of punctuation.

Chapter 8

Error mining

8.1 Introduction

The coverage (and accuracy) of a grammar are crucial to the success of parsers or generators written for that grammar. Improving coverage increases the number of sentences that a parser can analyze and the variety of sentences that can be generated. However, finding incomplete descriptions by hand can become a tedious task once a grammar has evolved.

In this chapter, we describe error mining, a technique that can be used to detect incomplete descriptions in a grammar or lexicon by parsing a large number of unannotated sentences. Error mining is closely related to classification — if we divide the sentences of a corpus in a bag of sentences for which the parser could find an analysis spanning the full sentence (henceforth called *parsable*) and a bag of *unparsable* sentences, we could construct a binary classifier that predicts the class of a sentence (parsable or unparsable). However, in this case we are not so much interested in classifying sentences, but in the features that indicate that a sentence is in the class 'unparsable', since these features could point to the source of a parsing error. An error miner performs two tasks:

- **Feature extraction** identifies superficial properties of sentences (such as sequences of words or part-of-speech tags), that are promising features.
- **Feature selection:** identifies the features that are the most discriminative for the 'unparsable' class.

Error mining can be used in addition to more traditional methods of grammar engineering, such as the manual construction of a set of example sentences

1.	@ , schreewt	*@ , screams*
2.	het libido	*the[neut] libido*
3.	de piste	*the circus-ring/pissed*
4.	goeden huize	*good[+infl] house[+infl]*
5.	tenzij .	*unless .*

Table 8.1: Example output of the error miner of van Noord (2004).

with the corresponding annotations to maintain or improve the coverage and accuracy of the parser [Nerbonne et al., 1993]. These methods are valuable too, but the error mining technique has the advantage that it can be applied to (large amounts of) new, previously unseen sentences. As a result, it can discover errors that need not have been anticipated, and which are not reflected in existing treebanks. For this reason, error mining is also particularly useful for adapting the parser to new domains and text genres.

As illustrated in van Noord [2004], an error miner is able to find quite distinct types of problems. Some typical examples reported in van Noord [2004] are repeated in Table 8.1, listing the features that indicated that problem. Here the character @ is used to indicate a sentence boundary. The examples illustrate the following types of problems:

1. Mistakes by the tokenizer and sentence splitter. In this example, a sentence boundary has been introduced wrongly after a direct quote, for instance in a sentence such as:

   (1) Je bent gek!, schreeuwt Franca.
   You are crazy!, yells Franca.
   (Franca yells: You are crazy!)

2. Mistakes in the lexicon (e.g., wrong attribute-value specification for a syntactic property such as gender or agreement). In this case, the noun 'libido' was specified as non-neuter in the lexicon, whereas it is typically used with the neuter determiner in the corpus.

3. Omissions in the lexicon. In this case, the lexicon did contain 'piste', the past tense form of the verb 'pissen', but it did not contain the more frequent noun 'piste' (circus ring).

4. Frozen expressions with idiosyncratic syntactic properties. The phrase 'van goeden huize' ('of good family') is a frozen expression with archaic inflection.

5. Incomplete grammatical descriptions. In the grammar, subordinate coordinators always need to combine with a complement. The word sequence 'tenzij .' is due to sentences in which a subordinate coordinator occurs without such a complement:

   (2) Gij zult niet doden, tenzij.
   Thou shallt not kill, unless.

In section 8.2 we first give a description of what is expected of feature extraction and feature selection components. We will then review the error miners of van Noord [2004] (Section 8.3.1) and Sagot and de la Clergerie [2006] (Section 8.3.2). Then, we discuss our new error miner in Section 8.4, which combines the strengths of the former proposals [de Kok et al., 2009]. Since error mining is most effective when huge corpora are parsed, some optimizations are required to keep error mining tractable. In Section 8.6 we discuss these optimizations. Previously, error miners have only been evaluated qualitatively. In Section 8.7 we discuss a simple evaluation method based on precision and recall. Finally, we compare the three error miners discussed in this chapter in Section 8.8.

8.2 Preliminaries

8.2.1 Feature selection

One of the tasks of an error miner is feature selection. It may seem attractive to apply one of the selection methods that was discussed in Chapter 6. The problem of applying correlation selection or one of the maximum entropy methods is, however, the size of the datasets and feature space used in error mining. For example, in the experiments that we discuss in this chapter, we use a corpus of 103 million sentences. The feature space in these experiments consisted of millions of features. In comparison, the training set in our fluency ranking experiments consisted of approximately 200,000 unique derivations after sampling, and nearly 4800 features before applying feature selection. The correlation and maximum entropy-based selection methods already required hours or days (depending on the method and granularity) to complete on a dataset and feature space that is orders of magnitude smaller than those in error mining.

Since it is infeasible to apply normal feature selection methods, error mining uses shallower methods for feature selection. What binds the methods discussed in this chapter is that they all use the notion of *suspicion*. The suspicion of

a feature is a number between 0 and 1 that provides an estimation of how effective that feature is in characterizing an unparsable sentence. If a feature has a suspicion of zero, it does not discriminate unparsable sentences at all. If a feature has a suspicion of one, it is believed to be a perfect discriminator for unparsable sentences. A feature with a high suspicion will typically point to a source of parsing errors.

8.2.2 Feature extraction

Given that the input to error mining consists of unannotated sentences, the error miner has to rely on surface properties of sentences. Consequently, the features in error mining are usually words and n-grams. Miners can also use other features that can be extracted with other natural language processing components that do not require parsing, such as part-of-speech tags.

In order to select features that are effective and not overlapping, feature extraction may already use a simple form of pre-selection to filter out features that do not look promising.

8.3 Previous work

In the literature, two promising error mining techniques have been been proposed. van Noord [2004] uses word n-grams of arbitrary length as its features and implements a simple, frequency-based feature selection method. Sagot and de la Clergerie [2006] proposes a more sophisticated iterative selection method. However, that method provides only rudimentary feature extraction. In this section, we will describe these proposals. However, in contrast to van Noord [2004] and Sagot and de la Clergerie [2006], we will formulate error mining explicitly in terms of feature selection and extraction.

8.3.1 Suspicion as a ratio

van Noord [2004] introduces the notion of parsability. The parsability of a feature, e.g., a word or a bigram, is the ratio of succesfully parsed sentences with regards to all sentences that contain this feature. Following Sagot and de la Clergerie [2006], we define the inverse of parsability, *suspicion*. In the introduction of this chapter, it was assumed that the corpus that is being mined can be divided into parsable and unparsable sentences. In the following description we use an error function $error(s)$, which is zero if the sentence s was parsable or one if it was not parsable. The use of an error function opens the

possibility of refining feature selection if more information about the severity of the parse error is available.

The suspicion of the feature f_i, $susp(f_i)$, then simply reduces to the mean error score of the sentences in which the feature f_i occurs. Here, S is the bag of sentences, and $f_i(s)$ is the number of occurrences of feature f_i in sentence s.

$$\text{susp}(f_i) = \frac{\sum_{s \in S} k(f_i(s)) \cdot error(s)}{\sum_{s \in S} k(f_i(s))} \tag{8.1}$$

where

$$k(f_i(s)) = \begin{cases} 1 & \text{if } f_i(s) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{8.2}$$

A variation of this method, which is actually used in the software provided for van Noord [2004], is to use the value of $f_i(s)$ directly:

$$susp(f_i) = \frac{\sum_{s \in S} f_i(s) \cdot error(s)}{\sum_{s \in S} f_i(s)} \tag{8.3}$$

We use the latter method when we evaluate the miner of van Noord [2004] in Section 8.7.

A feature can be a single word, but it can also be an n-gram. The inclusion of n-grams has been shown in van Noord [2004] to be very beneficial. In fact, all of the examples that were discussed in the introduction of this chapter (Table 8.1) are bigrams.

As another motivating example, consider the expression 'by and large' in English. Suppose that this expression is not treated in the dictionary, then presumably the parser will typically fail for sentences which include this trigram. As a result, the suspicion of this trigram will be very high. On the other hand, because each of the words 'by', 'and', and 'large' are very frequent in their normal usage, the suspicion of these individual words will be about as low as most other words. If no n-gram features are used, this systematic error will go unnoticed.

As an unwanted side-effect, note that all longer n-grams which contain the trigram 'by and large' will likewise get a very high suspicion. This is undesirable. Therefore, if n-grams are included as features in the error miner, a selection criterion is required. van Noord [2004] proposes to add a longer n-gram only if its suspicion is higher than all of the n-grams it contains:

$$\text{susp}(w_h ... w_i ... w_j ... w_k) > susp(w_i ... w_j) \tag{8.4}$$

As a result, there usually is only a small number of long n-grams which the error miner needs to take into account. This also implies that there is no need to set a value for n a priori - but rather the data determines which longer n-grams are required. As a further heuristic, if a longer n-gram satisfies the selection criterion, then the corresponding shorter n-grams are no longer considered as features for the error miner.

The resulting error mining technique appears to work well in practice. Yet, the method can be criticized because in the case that a sentence cannot be parsed correctly, all the words and n-grams which occur in that sentence are blamed for it. So it can happen that a feature obtains a high suspicion, simply because it co-occurs accidentally with real problematic features. In other words, the feature selection in this miner is limited, since it used a simple ratio.

8.3.2 Iterative error mining

The error mining method described by Sagot and de la Clergerie [2006] alleviates the problem of accidentally suspicious features. It does so by taking the following characteristics of suspicious features into account during feature selection:

1. If a feature occurs in parsable sentences, it becomes less likely that it is the cause of a parsing error.

2. The suspicion of a feature depends on the suspicions of other features in the sentences where it occurs.

3. A feature observed in a shorter sentence is initially more suspicious than a feature observed in a longer sentence.

These three characteristics make the suspicion of a feature dependent on its sentential context. To account for locality, this method introduces the notion of *observation suspicion*, $susp(f_i(s))$ which is the suspicion of feature f_i in sentence s. The (global) suspicion of a feature is defined as the average of all observation suspicions,

$$\text{susp}(f_i) = \frac{\sum_{s \in S} susp(f_i(s))}{\sum_{s \in S} f_i(s)} \tag{8.5}$$

The observation suspicions are dependent on the feature suspicions, making the method iterative. The observation suspicion is defined as the feature suspicion, normalized by suspicions of the other features that are active within the same sentence:

$$\text{susp}^{n+1}(f_i(s)) = \frac{error(s) \cdot susp^{n+1}(f_i)}{\sum_{f_j \in F(s)} susp^{n+1}(f_j)} \tag{8.6}$$

Here, $F(s)$ is the set of features that are active (have a non-zero frequency) in sentence s. Since the mining procedure is iterative, the suspicion of a feature is redefined to depend on the observation suspicions of the previous iteration:

$$\text{susp}^{n+1}(f_i) = \frac{\sum_{s \in S} \text{susp}^{n}(f_i(s))}{\sum_{s \in S} f_i(s)} \tag{8.7}$$

Given the recursive dependence between feature suspicions and observation suspicions, starting and stopping conditions are defined for iterative mining. The observation suspicions are initialized by uniformly distributing suspicion over the features that are observed in a sentence:

$$\text{susp}^{0}(f_i(s)) = \frac{error(s) \cdot f_i(s)}{\sum_{f_j \in F(s)} f_j(s)} \tag{8.8}$$

Mining is stopped when the process reaches a fixed point where the feature suspicions have stabilized.

This method addresses the main shortcoming of ratio-based error mining. However Sagot and de la Clergerie [2006] only use this method to mine unigrams and bigrams. They note that they have attempted to mine longer n-grams, but encountered data sparseness problems. Also, their work does not provide any criteria of when bigrams are to be preferred as features over unigrams.

8.4 N-gram expansion

8.4.1 Inclusion of n-grams

While the iterative miner described by Sagot and de la Clergerie [2006] only mines on features that are word unigrams and bigrams, our experience with the miner described by van Noord [2004] has shown that including n-grams that are longer than bigrams as features during mining can capture many additional phenomena. Reconsider the aforementioned trigram ‘by and large’. If this expression is the cause of a parsing error, then the words ‘by’, ‘and’ and ‘large’ have very low suspicions during error mining, while ‘by and large’ itself has a very high suspicion.

8.4.2 Suspicion sharing

It may seem attractive to use every possible n-gram in a sentence as a feature in iterative mining. However, this does not yield interesting results because suspicion will be shared between competing features. For instance, if use of the word 'velar' as an adjective in the trigram 'voiced velar stop' caused a parsing error, the trigram, and its bigrams 'voiced velar' and 'velar stop', will be suspicious as well. In such cases multiple features compete for suspicion, and the adjective 'velar' will have less opportunity to be detected as the cause of the errors.

Another concern is that the number of n-grams within a sentence grows at such a rate ($\sum_{i=0}^{n-1} n - i$) that the feature space becomes too large for feature selection to be tractable.

8.4.3 Expansion method

To avoid the sharing of suspicion between features, we propose a feature extraction method that adds and expands n-grams when it is deemed useful. This method iterates through a sentence unigram by unigram and expands unigrams to longer n-grams when there is sufficient evidence that the expansion will be useful. We then combine this feature extractor with the selection method of Sagot and de la Clergerie [2006]. Within this extractor, we use the definition of suspicion that was described in Section 8.3.1.

The method is based on the following observation: if we consider the word bigram w_1, w_2, either one of the unigrams or the bigram can be problematic. If one of the unigrams is problematic, the bigram will inherit suspicion of the problematic unigram. If the bigram is problematic, the bigram will have a higher suspicion than both of its unigrams. Consequently, we will want to expand the unigram w_1 to the bigram w_1, w_2 if the bigram is more suspicious than both of its unigrams. If the bigram is just as suspicious as one of its unigrams, such an expansion is not necessary, since we want to point to the cause of the parsing error as exactly as possible.

The same procedure is applied to longer n-grams. For instance, the expansion of the bigram w_1, w_2 to the trigram w_1, w_2, w_3 is only permitted if w_1, w_2, w_3 is more suspicious than its bigrams. Given that the suspicion of w_3 aggregates to w_2, w_3, we account for w_3 and w_2, w_3 simultaneously in this comparison.

The general criterion is that the expansion to an n-gram $i..j$ is permitted when susp($i..j$) > susp(i..j-1) and susp($i..j$) > susp(i+1..j). This gives us a bag of n-grams $w_1..w_x$, $w_2..w_y$, ... $w_{|s_i|}..w_{|s_i|}$ that represents sentence s_i optimally.

This method differs from that of van Noord [2004] in that the method of van Noord [2004] considers all n-grams in a sentence, while our method does not consider $w_i..w_j..w_k$ if the expansion to $w_i..w_j$ failed.

In Table 8.2, we illustrate our method to expand the n-gram feature 'voor' to 'voor uur van de' in the following sentence:

(3) De Disney-topman staat voor uur van de waarheid.
The Disney top exective stands before hour of the truth.
(Lit: It's the hour of truth for the Disney top executive.)

The counts in this example are based on real data.

Expansion	**susp($i..j$)**	**susp($i..j-1$)**	**susp($i+1..j$)**	**Expand**
voor → voor uur	$\frac{48}{50}$	$\frac{778949}{9590152}$	$\frac{116975}{1563498}$	yes
voor uur van	$\frac{40}{40}$	$\frac{48}{50}$	$\frac{856}{9779}$	yes
voor uur van de	$\frac{30}{30}$	$\frac{40}{40}$	$\frac{297}{3748}$	no

Table 8.2: Expansion of the feature 'voor' to 'voor uur van'.

Our expansion method also works with features that incorporate other kinds of surface information, such as part-of-speech tags. In such cases, we prefer to expand a feature using the most general type of information that is available. If w is a word and p a part-of-speech tag, and we are trying to expand the feature w_1, w_2 using either p_3 or w_3, the following procedure is followed. First we attempt to expand to w_1, w_2, p_3, since a part-of-speech tag is more general than a word. This expansion is allowed when $\text{susp}(w_1, w_2, p_3) > \text{susp}(w_1, w_2)$ and $\text{susp}(w_1, w_2, p_3) > \text{susp}(w_2, p_3)$. If the expansion is not allowed, then expansion to w_1, w_2, w_3 is attempted. As a result, mixed patterns emerge that are as general as possible.

8.4.4 Data sparseness

While this expansion method looked promising in our initial experiments, we found it to be too eager. This eagerness is caused by sparsity in the data. Since longer n-grams are less frequent, they also tend to be more suspicious if they occur in unparsable sentences. The expansion criterion does not take data sparseness into account.

We introduce an expansion factor to handle sparseness. This factor depends on the frequency of an n-gram in the set of unparsable sentences, and

asymptotically approaches one for higher frequencies. As a result the burden of proof is inflicted on the expansion: the longer n-gram needs to be relatively frequent or much more suspicious than its (n-1)-grams. The expansion criteria are changed to $\text{susp}(i..j) > \text{susp}(i..j\text{-}1) \cdot extFactor(i..j)$ and $\text{susp}(i..j) > \text{susp}(i\text{+}1..j) \cdot extFactor(i..j)$, where

$$extFactor(i..j) = 1 + exp(-\alpha \sum_{s \in S} error(s) \cdot f_{i..j}) \tag{8.9}$$

As we show in Section 8.8.2, $\alpha = 0.01$ proved to be a good setting.

8.5 Scoring functions

After error mining, we can extract a list of forms, ordered by suspicion. However, normally we are interested in forms that are both suspicious and frequent. Sagot and de la Clergerie [2006] proposed three scoring functions that can be used to rank forms:

1. Concentrating on suspicions:

$$score(f_i) = \text{susp}(f_i) \tag{8.10}$$

2. Concentrating on most frequent potential errors:

$$score(f_i) = \text{susp}(f_i) \ \cdot \ \sum_{s \in S} f_i(s) \tag{8.11}$$

3. Balancing between these possibilities:

$$score(f_i) = \text{susp}(f_i) \cdot ln(\sum_{S \in S} f_i(s)) \tag{8.12}$$

Since the frequency of a form in parsable sentences is not relevant to error mining, it is more interesting to focus on forms that are only frequent in unparsable sentences [de Kok et al., 2009]. In further experiments, we discovered that it is best to give a penalty for occurrences in parsable sentences as well. Some tokens, such as the start of sentence marker, have a low suspicion, but a very high frequency. Such features would still show up in the mining results when the number of occurrences in parsable sentences is not taken into account. We simply take such cases into account by subtracting the frequency of

a feature in parsable sentences from the frequency of that feature in unparsable sentences. Given the following definition of $delta(f)$,

$$delta(f_i) = \sum_{s \in S, error(s) > 0} f_i(s) \; - \sum_{s \in S, error = 0} f_i(s) \tag{8.13}$$

we revise the second and third scoring functions to:

$$score(f_i) = \; \text{susp}(f_i) \cdot delta(f_i) \tag{8.14}$$

$$score(f_i) = \begin{cases} 0 & \text{if } delta(f_i) \leq 0 \\ \text{susp}(f_i) \cdot (1 + ln(delta(f_i))) & \text{if } delta(f_i) > 0 \end{cases} \tag{8.15}$$

In the experiments later in this chapter, we use the scoring function that performed the best for a specific error miner. In the case of the iterative miner of Sagot and de la Clergerie [2006] and the miner proposed in this chapter, the scoring function in Equation 8.14 was the most effective. For the miner of van Noord [2006], the scoring function in Equation 8.15 performed the best.

8.6 Implementation

To be able to mine large corpora, we had to apply some interesting optimizations. We discuss some of these optimizations in this section.

8.6.1 Compact representation of data

During n-gram expansion (Section 8.4), the miner calculates ratio-based suspicions of n-grams using frequencies of n-grams in parsable and unparsable sentences. An n-gram can (potentially) be expanded to have the same length as the sentence. Consequently, it is not practical to store n-gram frequencies in a hash table, since we would have to store every possible n-gram within the corpus. Instead, we compute a suffix array [Manber and Myers, 1990] for the parsable and unparsable sentences.[1] A suffix array is an array that stores the indices of the tokens in a corpus. The array is sorted such that the indices point to the n-grams in the corpus in lexicographic order. The first column of Table 8.3 shows the suffix array for the corpus 'to be or not to be'. The corresponding lexicographically sorted n-grams are shown in the second column.

[1] We use the suffix sorting algorithm by Peter M. McIlroy and M. Douglas McIlroy.

Suffix array	Corresponding n-gram
5	be
1	be or not to be
3	not to be
2	or not to be
4	to be
0	to be or not to be

Table 8.3: Suffix array for the corpus *to be or not to be* with the n-gram that corresponds to each index.

We use suffix arrays differently than van Noord [2004], because our expansion algorithm requires the parsable and unparsable frequencies of the (n-1)-grams, and the second (n-1)-gram is not (necessarily) adjacent to the n-gram in the suffix array. As such, we require random access to frequencies of n-grams occurring in the corpus. We can compute the frequency of any n-gram by looking up its first and last occurrence in the suffix array.[2] For example, if we look up 'to be' in the suffix array in Table 8.3 we will find that the fifth element is the first occurrence of 'to be' and the sixth element the last. Consequently, we know that the frequency of 'to be' is two. As van Noord [2004] we use perfect hashing to represent tokens throughout the error miner. Since hash codes are generally shorter than the average token length, this saves memory. Moreover, it improves performance across the board, since integer comparisons are much faster than string comparisons.

8.6.2 Determining ratios for pattern expansion

While suffix arrays provide a compact and relatively fast data structure for looking up n-gram frequencies, they are not fit for determining the frequency of mixed patterns, such as n-grams that consist of a mix of words and part-of-speech tags. For instance, consider the expansion of *het Adj* in the sentence *het/Det grote/Adj huis/Noun is/Verb leeg/Adv* 'the big house is empty'. To look up the frequencies of the competing features *het Adj Noun* and *het Adj huis* we would need suffix arrays corresponding to two different corpora:

(4) a. ... het Adj Noun ...
b. ... het Adj huis ...

[2]Since the suffix array is sorted, finding the first and last occurrences is a binary search in $O(log(n))$ time.

Since we have this choice for every position of the corpus, we would need t^n different suffix arrays, where t is the number of types of information and n the length of the corpus.

Since suffix arrays are not viable for looking up frequencies of such mixed patterns, we use a different method for computing frequencies of mixed patterns. First, we build a hash table for each type of information that can be used in patterns. A hash table contains an instance of such information as a key (e.g. a specific token or part-of-speech tag) and a set of corpus indices with that instance as the value associated with that key. Then, we can look up the frequency of a sequence *i..j* by calculating the set intersection of the indices of *j* and the indices found for the sequence *i..j-1*, after incrementing the indices of *i..j-1* by one. Et cetera.

For example, if we want to obtain the frequency of *het Adj huis* using the hash table in Table 8.4, we proceed as follows:

1. Look up the indices of *het* $\rightarrow \{300, 766, 937, 1233\}$
2. Increment the indices by one $\rightarrow \{301, 767, 938, 1235\}$
3. Compute the intersection of these indices and the indices of *Adj* $\rightarrow$ $\{301, 767, 938, 1235\} \cap \{767, 938, 1319\} = \{767, 938\}$
4. Increment the indices by one $\rightarrow \{768, 939\}$
5. Compute the intersection these indices and the indices of *huis* $\rightarrow \{768, 939\} \cap$ $\{323, 768, 1844\} = \{768\}$
6. Obtain the frequency $\rightarrow |\{768\}| = 1$

Instance	**Indices**
het	$\{300, 766, 937, 1233\}$
Adj	$\{767, 938, 1319\}$
huis	$\{323, 768, 1844\}$
$\vdots$	$\vdots$

Table 8.4: Example of a table with instances and their indices.

The complexity of calculating frequencies following this method is linear, since the set of indices for a given instance can be retrieved with $O(1)$ time complexity, while both incrementing the set indices and set intersection can

be performed in $O(n)$ time. However, n can be very large — for instance, the start of sentence marker forms a substantial part of the corpus and is looked up once for every sentence. In our implementation we limit the time spent on such patterns by caching very frequent bigrams in a hash table.

8.6.3 Suspicion purification

Since only one feature within a sentence will normally be responsible for a parsing error, many features will have almost no suspicion at all. However, during the mining process, their suspicions will be recalculated during every cycle. Mining can be sped up considerably by removing features that have a negligible suspicion. As an added advantage, it allows the suspicion of forms that were near one to be exactly one, since their (negligible) competition is removed.

In an experiment where we mined the Dutch Wikipedia corpus using n-gram expansion and iterative mining, our error miner extracted 4.8 million features that occurred 13.4 million times in unparsable sentences. When we mined the same material and dropped features that converged to a suspicion lower than 0.001, error mining resulted in a set of 3.5 million features that occurred 4.0 million times in unparsable sentences. This shows that this *suspicion purification* reduces the feature space considerably, leading to more efficient error mining.

8.7 Evaluation

8.7.1 Methodology

In previous on error mining, error mining methods have been evaluated primarily manually. Both van Noord [2004] and Sagot and de la Clergerie [2006] conduct a qualitative analysis of highly suspicious features. But once one starts experimenting with various extensions, such as n-gram expansion and expansion factor functions, it is difficult to gauge the contribution of modifications through a small-scale qualitative analysis.

To be able to evaluate changes to the error miner, we have supplemented qualitative analysis with a automatic quantitative evaluation method. Such a method should judge an error miner in line with the interests of a grammar engineer:

- An error miner should return features that point to problems that occur in a large number of sentences.

- The features that are returned by the error miner should be as exact as possible in pointing to the problem.

The first requirement is relatively easy to test — the error miner should return features that occur in a relatively large number of unparsable sentences. It is less clear how the second requirement should be tested, since it requires that a human checks the features to be relevant. However, if we assume that the quality of features correlates strongly to their discriminative power, then we would expect a miner to return features that only occur in unparsable sentences. These characteristics can be measured using the recall and precision metrics from information retrieval:

$$R = \frac{|\{S_{unparsable}\} \cap \{S_{retrieved}\}|}{|\{S_{unparsable}\}|} \tag{8.16}$$

$$P = \frac{|\{S_{unparsable}\} \cap \{S_{retrieved}\}|}{|\{S_{retrieved}\}|} \tag{8.17}$$

Consequently, we can also calculate the f-score [van Rijsbergen, 1979]:

$$F - score = \frac{(1 + \beta^2) \cdot (P \cdot R)}{(\beta^2 \cdot P + R)} \tag{8.18}$$

The f-score is often used with $\beta = 1.0$ to give equal weight to precision and recall. In evaluating error mining, this can cater to cheating. For instance, consider an error miner that recalls the start of sentence marker as the first problematic feature. Such a strategy would instantly give a recall of 1.0, and if the coverage of a parser for a corpus is relatively low, a relatively good initial f-score will be obtained. For this reason, we give more bias to precision by using $\beta = 0.5$.

8.7.2 Material

In our experiments we use the Flemisch Mediargus newspaper corpus. This corpus consists of 103 million sentences (1.5 billion words). For 6.2% of the sentences no full analysis could be found. Flemish is a variation of Dutch written and spoken in Belgium, with a grammar and lexicon that deviates slightly from standard Dutch. Previously, the Alpino grammar and lexicon were never specifically modified for parsing Flemish.

8.8 Results

In this section, we discuss the results of the quantitative and qualitative evaluation. We will first compare the miners described in van Noord [2004] and Sagot and de la Clergerie [2006]. Then, we examine the performance of the expansion method that we proposed in this chapter and compare it to the competition. Finally, we will conclude this section with an qualitative evaluation of iterative error mining with our expansion method.

8.8.1 Iterative error mining

Our first interest was if, and how much iterative error mining outperforms error mining with suspicion as a ratio. To test this, we compared the method described by van Noord [2004] and the iterative error miner of Sagot and de la Clergerie [2006]. For the iterative error miner we included all bigrams and unigrams without further selection. Figure 8.1 shows the F-scores for these miners after N retrieved features.

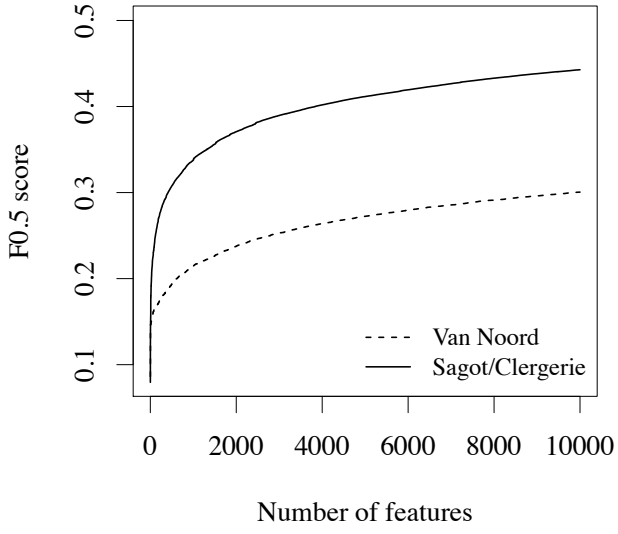


Figure 8.1: F-scores after retrieving N features for ratio-based mining, iterative mining on unigrams and iterative mining on uni- and bigrams.

The iterative miner of Sagot and de la Clergerie [2006] clearly outperforms the miner of van Noord [2004], despite the fact that the latter has a more

sophisticated feature extraction method. That this happens is understandable — suppose that 60% of the occurrences of a frequent feature is in unparsable sentences. In such a case, the ratio-based miner would assign a suspicion of 0.6. But, since the feature is relatively frequent, it would still be ranked very high, even though there is plenty of evidence that it is not responsible for parsing errors. This also manifests itself in the performance of scoring functions — the ratio-based miner was the only miner to perform best with the scoring function in Equation 8.15. This indicates that relying too much on frequencies is dangerous in ratio-based mining. However, relying purely on suspicion would return many forms with a low frequency.

Another interesting characteristic of these results are that the performance of the error miners seems to fit a logarithmic function. This is not surprising, since it shows that there are some very frequent errors and a long tail of less frequent errors. The fact that there is a long tail of infrequent parsing errors does not make the task of the grammar engineer hopeless. Our preliminary experiments with pattern mining that we discuss later in this section, shows that quite often infrequent word n-gram patterns can be combined into more frequent mixed patterns.

8.8.2 N-gram expansion

In our second experiment, we compare the performance of iterative mining on uni- and bigrams with an iterative miner that uses the n-gram expansion algorithm that was described in section 8.4. However, before carrying out this experiment, we need a proper value of α for the expansion factor. Figure 8.2 shows the performance of the iterative miners with n-gram expansion with a varying value of α. We also show the performance of the n-gram expansion method without the use of an expansion factor.

As we can see, the use of an expansion factor is clearly useful. The iterative miner that applies n-gram expansion without an expansion factor, chooses n-grams that are too specific and thus less frequent. Its performance is worse than that of the miners that use an expansion factor for the first 10,000 forms. The choice of an α value also influences the performance of the miner quite a bit. For the present experiment, we settled on $\alpha = 0.01$, which provides good performance across the board.

In Figure 8.3, we compare our miner ($\alpha = 0.01$) that uses word n-gram expansion with the miner of Sagot and de la Clergerie [2006]. Here, we see clearly that the inclusion of longer n-grams is beneficial to error mining, as long as we are careful to expand n-grams only when it is proved to be useful.

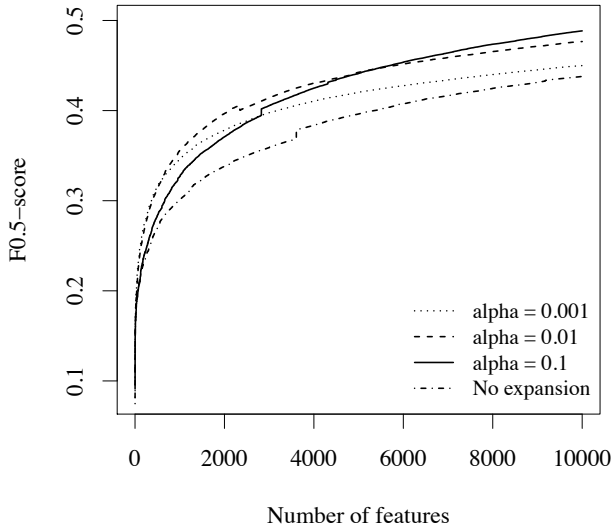


Figure 8.2: $F_{0.5}$-scores after retrieving N features in iterative mining using word n-gram expansion, with different expansion factors.

8.8.3 Manual analysis

We found many interesting longer n-grams in the results of the miner proposed in this chapter that could not have been captured by the miner of Sagot and de la Clergerie [2006]. Many of these examples are idiomatic expressions in Flemish that were not described in the Alpino lexicon. For example:

- had er (AMOUNT) voor veil [had (AMOUNT) for sale]
- (om de muren) van op te lopen [to get terribly annoyed by]
- Ik durf zeggen dat [I dare to say that]
- op punt stellen [to fix/correct something]
- de daver op het lijf [shocked]
- (op) de tippen (van zijn tenen) [being very careful]
- ben fier dat [am proud of]
- Nog voor halfweg [still before halfway]

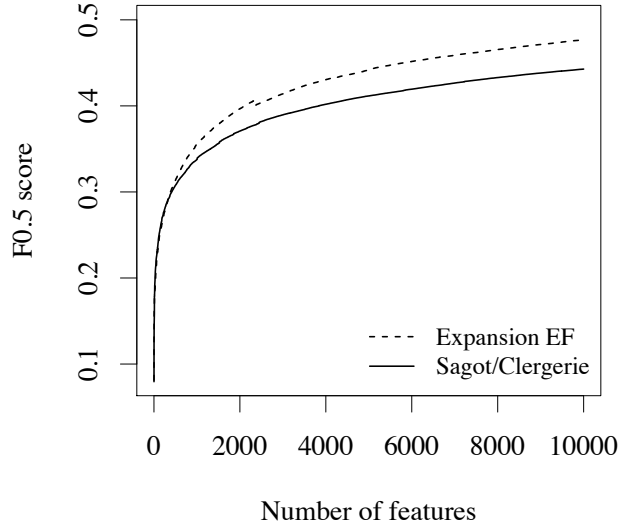


Figure 8.3: F-scores after retrieving N features for the miner of Sagot and de la Clergerie [2006] and the miner proposed in this chapter (with and without the use of an expansion factor).

- (om duimen en vingers) van af te likken [delicious]

We also found longer n-grams describing valid Dutch grammatical constructions that were not described by the Alpino grammar. For example:

- Het stond in de sterren geschreven dat [It was written in the stars that]
- zowat de helft van de [about half of the]
- er zo goed als zeker van dat [almost sure of]
- laat ons hopen dat het/dit lukt [let us hope that it/this works]

8.8.4 Pattern expansion

We have done some preliminary experiments with pattern expansion, allowing for patterns consisting of words and part-of-speech tags. For this experiment we trained a Hidden Markov Model part-of-speech tagger on the Dutch Eindhoven corpus using a small tag set. We then extracted 50000 unparsable and about

495000 parsable sentences from the Flemish Mediargus corpus. We then used iterative mining with a variant of our feature extractor that supports patterns.

We will give two short examples of patterns that were extracted to give an impression how patterns can be useful. A frequent pattern was *doorheen Noun* ('through' followed by a (proper) noun). In Flemish a sentence such as *We reden met de auto doorheen Frankrijk* 'We drove with the car through France' is allowed, while in standard Dutch the particle *heen* is separated from the preposition *door*. Consequently, the same sentence in standard Dutch is *We reden met de auto door Frankrijk heen.* Mining purely on word n-grams provided hints for this difference in Flemish through features such as *doorheen Krottegem*, *doorheen Engeland*, *doorheen Hawai*, and *doorheen Middelkerke*, but the pattern provides a more general description with a higher frequency.

Another pattern that was found is *wegens Prep Adj* ('because of' followed by a preposition and an adjective).[3] This pattern captures prepositional modifiers where *wegens* is the head and the following words within the constituent form an argument, such as in the sentence:

(5) Dat idee werd snel opgeborgen wegens te duur
That idea became soon archived because of too expensive
That idea was quickly rejected, because it was too expensive

This pattern provided a more general description of features such as *wegens te breed* 'because it is too wide', *wegens te deprimerend* 'because it is too depressing', and *wegens te ondraaglijk* 'because it is too unbearable'. In fact, there were 120 such word n-gram features that are condensed into the pattern *wegens Prep Adj*.

Including patterns as feature is attractive because: it more accurately points to the error in the grammar; it can expose itself better, since it has a higher frequency; and it is less work for the grammar engineer, because he has to inspect only one feature rather than 120 different features.

Since we are still optimizing the pattern expansion preprocessor to scale to large corpora, we have not performed an automatic evaluation using larger fragments of the Mediargus corpus yet.

8.9 Conclusions

In this chapter, we have combined iterative error mining with a new feature extractor that includes n-grams of an arbitrary length and that are long enough

[3]The preposition can be considered an incorrect tagging for adverbial, as we will see in the following examples.

to capture interesting phenomena, but not longer. We dealt with the problem of data sparseness by introducing an expansion factor that softens when the expanded feature is very frequent.

In addition to the generalization of iterative error mining, we have introduced a method for automatic evaluation that is based on the precision and recall scores commonly used in information retrieval. This allows us to test modifications to error minings quickly without going through the tedious task of ranking and judging the results manually.

Using this automatic evaluation method, we have shown that iterative error mining improves well upon ratio-based error mining. We have also shown that the use of a smart feature extraction method improves error miners substantially. The inclusion of longer n-grams captures many interesting problems that could not be found if a miner restricted itself to words and word bigrams.

We have also shown preliminary work on a feature extractor that does not just expand to word n-grams, but allows for more general patterns that can incorporate additional information, such as part-of-speech tags and lemmas. Our initial experiments with this feature extractor showed promising results. However the performance of this extractor needs to be improved in order to be able to use it on large corpora, such as the Mediargus corpus.

The error mining methods described in this chapter are generic, and can be used for any grammar or parser, as long as the sentences within the corpus can be divided into a list of parsable and unparsable sentences.

Chapter 9

Conclusion

The overarching objective of this thesis is to demonstrate that it is possible to make a system where a parser and generator share not only one grammar and lexicon, but also one statistical model for parse disambiguation and fluency ranking. To ensure that the development of such a formalism is not merely a theoretical exercise, we stated that any proposal for a reversible statistic model should fulfill two requirements: (1) the model should perform as well as carefully developed parse disambiguation and fluency ranking models; and (2) there should be a certain amount of integration between the tasks within that model.

In this thesis, we proposed the Reversible Stochastic Attribute-Value Grammar formalism (Chapter 5), that combines attribute-value grammar with one model that can be used in parse disambiguation and fluency ranking. We also demonstrated that Reversible Stochastic Attribute-Value Grammar satisfies the aforementioned requirements. In the experiments of Chapter 5, we showed that the reversible component does not perform significantly differently from components that were specifically developed and trained for parse disambiguation and fluency ranking. In Chapter 7 we isolated the most discriminative features of reversible models and showed that features that are used in both directions are used in reversible models. In fact, even more so than in directional models, showing that there is indeed integration between both tasks.

To come to this conclusion, however, we needed a system that provides an attribute-value grammar, a parser, a generator, and methods to find discriminative features. The Alpino system [van Noord, 2006] provides an attribute-value grammar (Chapter 2) and parser for Dutch.

After providing an overview of attribute-value grammar in Alpino in Chap-

ter 2, we introduced our generator for the Alpino grammar in Chapter 3. One of the difficult challenges of generation is to reduce the search space enough to make generation tractable. In this chapter, we introduced a new method for top-down guidance in generation, *semantic restriction*. This method unifies lexical items with the relevant portions of the input in order to guide generation. As a result, the construction of spurious items is nipped in the bud at the earliest possible moment, namely during unification.

We introduced our fluency ranking model in Chapter 4. This model uses maximum entropy modeling to exploit features that describe a derivation, such as word and tag trigram models, rule enumerations, and deep syntactic features. We evaluated the ranker and conducted an error analysis that shows that most of the remaining errors were caused by limitations of the input representation, rather than the ranker. We also provided a survey of methods to estimate the probability of sentences, abstract representations and derivations in the training data. We showed that the method that uses a uniform distribution for sentences and abstract representations and binary quality indicators for events performed significantly better than the other methods. As an additional benefit, this method does not skew the training data in favor of parse disambiguation or fluency ranking in reversible models.

To verify which features are effective in parse disambiguation, fluency ranking, and reversible models, we applied feature selection. In Chapter 6, we compared our own *correlation selection* method and four other selection methods. The grafting method [Perkins et al., 2003] was shown to be the most effective feature selection method in both parse disambiguation and fluency ranking. We then used this method to provide an extensive analysis of the most discriminative features in fluency ranking and showed that the fluency of a sentence can be described by a small number of features that model word and part-of-speech trigram distributions, topicalization, modifier adjoining, and ordering in the middle field.

Finally, in Chapter 8 we proposed a generalized error miner, that can detect shortcomings in a grammar or lexicon by using a large number of sentences that were not manually annotated. Our proposal differs from previous work in that it extracts mixed patterns of an arbitrary length and is well-equipped to deal with data sparsity. We compare this error miner with other error miners using a new quantitative evaluation method that takes the concerns of grammar engineers into account. Using this evaluation method, we show that our error miner consistently outperforms the competition.

Bibliography

S. Abney. Stochastic Attribute-Value Grammars. *Computational Linguistics*, 23(4):597–618, 1997.

H. Alshawi. Resolving quasi logical forms. *Comput. Linguist.*, 16:133–144, September 1990. ISSN 0891-2017.

G. Andrew and J. Gao. Scalable training of l 1-regularized log-linear models. In *Proceedings of the 24th international conference on Machine learning*, pages 33–40. Association for Computing Machinery, 2007.

W.H. Atteveldt. *Semantic network analysis: Techniques for extracting, representing, and querying media content*. PhD thesis, Vrije Universiteit Amsterdam, 2008.

S.J. Benson and J. Moré. A limited-memory variable-metric algorithm for bound-constrained minimization. Technical report, Technical Report ANL/MCS-P909-0901, Mathematics and Computer Science Division, Argonne National Laboratory, 2001.

A.L. Berger, V.J.D. Pietra, and S.A.D. Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):71, 1996.

J. Bos. Wide-coverage semantic analysis with Boxer. In Johan Bos and Rodolfo Delmonte, editors, *Semantics in Text Processing. STEP 2008 Conference Proceedings*, Research in Computational Semantics, pages 277–286. College Publications, 2008.

G. Bouma, J. Mur, G. van Noord, L. Van Der Plas, and J. Tiedemann. Question answering for Dutch using dependency relations. *Accessing Multilingual Information Repositories*, pages 370–379, 2006.

T. Brants. TnT – a statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing (ANLP-2000)*, Seattle, WA, USA, 2000.

S. Busemann. Best-first surface realization. In Donia Scott, editor, *Proceedings of the Eighth International Natural Language Generation Workshop (INLG '96)*, pages 101–110, 1996.

A. Cahill. Correlating human and automatic evaluation of a German surface realiser. In *Proceedings of the ACL-IJCNLP 2009 Conference - Short Papers*, pages 97–100, 2009.

A. Cahill, M. Forst, and C. Rohrer. Stochastic realisation ranking for a free word order language. In *ENLG '07: Proceedings of the Eleventh European Workshop on Natural Language Generation*, pages 17–24, 2007.

A. Cahill, M. Burke, R. O'Donovan, S. Riezler, J. van Genabith, and A. Way. Wide-coverage deep statistical parsing using automatic dependency structure annotation. *Comput. Linguist.*, 34(1):81–124, March 2008. ISSN 0891-2017. doi: 10.1162/coli.2008.34.1.81.

B. Carpenter. *The logic of typed feature structures: with applications to unification grammars, logic programs and constraint resolution.* Cambridge University Press, 1992.

J. Carroll and S. Oepen. High efficiency realization for a wide-coverage unification grammar. *Natural Language Processing–IJCNLP 2005*, pages 165–176, 2005.

A. Cauchy. Méthode générale pour la résolution des systemes déquations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

S.F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.

S. Clark and J. R. Curran. Log-linear models for wide-coverage ccg parsing. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, EMNLP '03, pages 97–104, 2003.

S. Clark and J. R. Curran. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting of the ACL*, pages 103–110, Barcelona, Spain, 2004.

P.R. Cohen. *Empirical methods for artificial intelligence*, volume 55. MIT press Cambridge, Massachusetts, 1995.

A. Copestake. Appendix: Definitions of typed feature structures. *Natural Language Engineering*, 6(1):109–112, 2000.

A. Copestake, D. Flickinger, C. Pollard, and I.A. Sag. Minimal Recursion Semantics: An introduction. *Research on Language & Computation*, 3(4): 281–332, 2005.

J. Daciuk. Finite state tools for natural language processing. In *Proceedings of the COLING 2000 Workshop Using Toolsets and Architectures to Build NLP Systems*, pages 34–37, 2000.

J. Daciuk and G. van Noord. Finite automata for compact representation of tuple dictionaries. *Theoretical Computer Science*, 313:45–56, February 2004. ISSN 0304-3975.

J.N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43(5):1470–1480, 1972.

J. De Belder, D. de Kok, G. van Noord, F. Nauze, L. van der Beek, and M. Moens. Question answering of informative web pages: How summarisation technology helps. In Peter Spyns and Jan Odijk, editors, *Essential Speech and Language Technology for Dutch*. Springer, 2012.

D. de Kok. Feature selection for fluency ranking. In *Proceedings of the 6th International Natural Language Generation Conference (INLG)*, pages 155–163, Trim, Ireland, 2010.

D. de Kok. Discriminative features in Reversible Stochastic Attribute-Value Grammars. In *Proceedings of the UCNLG+Eval: Language Generation and Evaluation Workshop*, pages 54–63, Edinburgh, United Kingdom, 2011. Association for Computational Linguistics.

D. de Kok and G. van Noord. A sentence generator for Dutch. In *Proceedings of the 20th Computational Linguistics in the Netherlands conference (CLIN)*, 2010.

D. de Kok, J. Ma, and G. van Noord. A generalized method for iterative error mining in parsing results. In *Proceedings of the 2009 Workshop on Grammar Engineering Across Frameworks (GEAF 2009)*, pages 71–79, Suntec, Singapore, August 2009. Association for Computational Linguistics.

D. de Kok, B. Plank, and G. van Noord. Reversible Stochastic Attribute-Value Grammars. In *Proceedings of the ACL HLT 2011 Conference-Short Papers*, pages 194–199, Portland, OR, USA, 2011. Association for Computational Linguistics.

S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(4):380–393, 1997.

R. M. W. Dixon. Where have all the adjectives gone? *Studies in language*, (1):19–80, 1977.

O. J. Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64, 1961.

R. Fletcher and C.M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154, 1964.

M. Forst. Filling statistics with linguistics: property design for the disambiguation of German LFG parses. In *DeepLP '07: Proceedings of the Workshop on Deep Linguistic Processing*, pages 17–24, Prague, Czech Republic, 2007.

S. Geman and M. Johnson. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 279–286. Association for Computational Linguistics, 2002.

I.J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264, 1953.

J.T. Goodman. A bit of progress in language modeling: Extended version. *Computer Speech & Language*, 15(4):403–434, 2001.

E. T. Jaynes. Information theory and statistical mechanics. *The Physical Review*, 106:620–630, May 1957a.

E. T. Jaynes. Information theory and statistical mechanics. II. *The Physical Review*, 108:171–190, Oct 1957b.

F. Jelinek. Interpolated estimation of markov source parameters from sparse data. *Pattern recognition in practice*, pages 381–397, 1980.

M. Johnson. Parsing with discontinuous constituents. In *Proceedings of the 23rd annual meeting on Association for Computational Linguistics*, pages 127–132. Association for Computational Linguistics, 1985.

M. Johnson and S. Riezler. Exploiting auxiliary distributions in stochastic unification-based grammars. In *Proceedings of the 1st NAACL conference*, pages 154–161, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

M. Johnson, S. Geman, S. Canon, Z. Chi, and S. Riezler. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the ACL*, pages 535–541. Association for Computational Linguistics, 1999.

H. Kamp and U. Reyle. *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*, volume 42. Kluwer Academic Dordrecht, The Netherlands, 1993.

R. M. Kaplan and J. Bresnan. Lexical-Functional Grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, MA, 1982.

M. Kay. Syntactic processing and functional sentence perspective. In *TINLAP '75: Proceedings of the 1975 workshop on Theoretical issues in natural language processing*, pages 12–15, Cambridge, MA, USA, 1975.

M. Kay. Functional Unification Grammar: a formalism for machine translation. In *Proceedings of the 10th international conference on Computational linguistics*, COLING '84, pages 75–78, Stanford, CA, USA, 1984. Association for Computational Linguistics.

M. Kay. Chart generation. In *Proceedings of the 34th annual meeting on ACL*, pages 200–204. Association for Computational Linguistics, 1996.

K. Knight and V. Hatzivassiloglou. Two-level, many-paths generation. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 252–260. Association for Computational Linguistics, 1995.

J. Koster. Dutch as an sov language. *Linguistic analysis*, 1(2):111–136, 1975.

I. Langkilde. Forest-based statistical sentence generation. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 170–177. Morgan Kaufmann Publishers Inc., 2000.

I. Langkilde and K. Knight. The practical value of N-grams in generation. In *Proc. of the Ninth International Workshop on Natural Language Generation*, pages 248–255, 1998.

I. Langkilde-Geary. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of the 12th International Natural Language Generation Workshop*, pages 17–24, 2002.

G.J. Lidstone. Note on the general case of the bayes-laplace formula for inductive or a posteriori probabilities. *Transactions of the Faculty of Actuaries*, 8 (182-192):13, 1920.

D.C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

J. Liu and A. Haghighi. Ordering prenominal modifiers with a reranking approach. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1109–1116. Association for Computational Linguistics, 2011.

R. Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the sixth conference on natural language learning (CoNLL-2002)*, pages 49–55, 2002.

U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on discrete algorithms*, pages 319–327. Society for Industrial and Applied Mathematics, 1990. ISBN 0898712513.

A.A. Markov. The theory of algorithms. *Trudy Matematicheskogo Instituta im. VA Steklova*, 42:3–375, 1954.

E. Marsi and E. Krahmer. Explorations in sentence fusion. In *Proceedings of the European Workshop on Natural Language Generation*, pages 109–117, 2005.

Y. Matsumoto, H. Tanaka, H. Hirakawa, H. Miyoshi, and H. Yasukawa. BUP: a bottom-up parser embedded in Prolog. *New Generation Computing*, 1(2): 145–158, 1983.

I. D. Melamed, R. Green, and J. Turian. Precision and recall of machine translation. In *HLT-NAACL*, pages 61–63, 2003.

I. Mel'čuk. Dependency syntax. *State University of New York Press, Albany, NY*, 1988.

C.S. Mellish. Implementing systemic classification by unification. *Computational Linguistics*, 14(1):40–51, 1988.

M. Mitchell. Class-based ordering of prenominal modifiers. In *Proceedings of the 12th European Workshop on Natural Language Generation*, pages 50–57. Association for Computational Linguistics, 2009.

Y. Miyao and J. Tsujii. Maximum entropy estimation for feature forests. In *Proc. of HLT*, volume 2, 2002.

Y. Miyao and J. Tsujii. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd Annual Meeting of the ACL*, pages 83–90, Ann Arbor, Michigan, 2005.

J. Nerbonne, K. Netter, A.K. Diagne, J. Klein, and L. Dickmann. A diagnostic tool for German syntax. *Machine Translation*, 8(1):85–107, 1993.

J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

E.W. Noreen. *Computer-intensive methods for testing hypotheses: an introduction*. A Wiley Interscience publication. Wiley, 1989. ISBN 9780471611363.

M. Osborne. Estimation of stochastic attribute-value grammars using an informative sample. In *Proceedings of the 18th conference on Computational linguistics-Volume 1*, pages 586–592. Association for Computational Linguistics, 2000.

F.C.N. Pereira and D.H.D. Warren. Parsing as deduction. In *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, pages 137–144. Association for Computational Linguistics, 1983.

S. Perkins, K. Lacker, and J. Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. *The Journal of Machine Learning Research*, 3:1333–1356, 2003. ISSN 1532-4435.

J.D. Phillips. Generation of text from logical formulae. *Machine Translation*, 8(4):209–235, 1993.

B. Plank. *Domain Adaptation for Parsing*. PhD thesis, University of Groningen, 2011.

E. Polak and G. Ribiere. Note sur la convergence de méthodes de directions conjuguées. *Revue Fr Inf Rech Oper*, 3(1):35–43, 1969.

F. Popowich. A chart generator for shake and bake machine translation. *Advances in Artifical Intelligence*, pages 97–108, 1996.

R. Quirk and S. Greenbaum. A university grammar of English-workbook. 1974.

E. Reiter and R. Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87, 1997.

S. Riezler and J.T. Maxwell. On some pitfalls in automatic evaluation and significance testing for MT. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 57–64, 2005.

S. Riezler and A. Vasserman. Incremental feature selection and l1 regularization for relaxed maximum-entropy modeling. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP04), Barcelona, Spain*, 2004.

S. Riezler, T. H. King, R. M. Kaplan, R. Crouch, J. T. Maxwell III, and M. Johnson. Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the ACL*, pages 271–278, Philadelphia, PA, USA, 2002.

G. Russell, S. Warwick, and J. Carroll. Asymmetry in parsing and generating with unification grammars: case studies from elu. In *Proceedings of the 28th annual meeting on Association for Computational Linguistics*, ACL '90, pages 205–211, Pittsburgh, PA, USA, 1990. Association for Computational Linguistics.

B. Sagot and É. de la Clergerie. Error mining in parsing results. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 329–336, Sydney, Australia, 2006. Association for Computational Linguistics.

J. Shaw and V. Hatzivassiloglou. Ordering among premodifiers. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, ACL '99, pages 135–143, College Park, MD, USA, 1999. Association for Computational Linguistics. ISBN 1-55860-609-3.

S. M. Shieber. *An Introduction to Unification-Based Approaches to Grammar*, volume 4 of *CSLI Lecture Notes Series*. Center for the Study of Language and Information, Stanford, CA, 1986.

S. M. Shieber. A uniform architecture for parsing and generation. In *Proceedings of the 12th COLING conference*, Budapest, 1988.

S.M. Shieber, G. van Noord, F.C.N. Pereira, and R.C. Moore. Semantic-head-driven generation. *Computational Linguistics*, 16(1):30–42, 1990.

S.M. Shieber, Y. Schabes, and F.C.N. Pereira. Principles and implementation of deductive parsing. *The Journal of logic programming*, 24(1-2):3–36, 1995.

R. Sproat and C. Shih. The cross-linguistic distribution of adjective ordering restrictions. *Interdisciplinary approaches to language*, pages 565–593, 1991.

L. Tesnière. *Eléments de syntaxe structurale*. Libraire C. Klincksieck, Paris, 1959.

R. Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996. ISSN 0035-9246.

K. Toutanova, C. D. Manning, S. M. Shieber, D. Flickinger, and S. Oepen. Parse disambiguation for a rich HPSG grammar. In *First Workshop on Treebanks and Linguistic Theories (TLT)*, pages 253–263, Sozopol, 2002.

I.A. Trujillo. *Lexicalist machine translation of spatial prepositions*. PhD thesis, University of Cambridge, 1995.

G. van Noord. Error mining for wide-coverage grammar engineering. In *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 446, Barcelona, Spain, 2004. Association for Computational Linguistics.

G. van Noord. **A**t **L**ast **P**arsing **I**s **N**ow **O**perational. In *TALN 2006 Verbum Ex Machina, Actes De La 13e Conference sur Le Traitement Automatique des Langues naturelles*, pages 20–42, Leuven, 2006.

G. van Noord. Using self-trained bilexical preferences to improve disambiguation accuracy. In *Proceedings of the International Workshop on Parsing Technology (IWPT)*, ACL 2007 Workshop, pages 1–10, Prague, 2007. Association for Computational Linguistics, ACL.

G. van Noord and G. Bouma. Adjuncts and the processing of lexical rules. In *Proceedings of the 15th conference on Computational linguistics-Volume 1*, pages 250–256. Association for Computational Linguistics, 1994.

G. van Noord and R. Malouf. Wide coverage parsing with Stochastic Attribute Value Grammars. Draft available from the authors. A preliminary version of this paper was published in the Proceedings of the IJCNLP workshop Beyond Shallow Analyses, Hainan China, 2004., 2005.

G. van Noord, I. Schuurman, and G. Bouma. Lassy syntactische annotatie, revision 19455, 2011.

G. van Noord, F. Van Eynde, D. de Kok, J. van der Linde, I. Schuurman, E. Tjong Kim Sang, and V. Vandeghinste. Large scale syntactic annotation of written Dutch: Lassy. In Peter Spyns and Jan Odijk, editors, *Essential Speech and Language Technology for Dutch*. Springer, 2012.

C. J. van Rijsbergen. *Information retrieval*. Butterworths, London, 2 edition, 1979.

V. Vandeghinste. Tree-based target language modeling. In *13th Annual conference of the European Association for machine translation*, pages 152–159, 2009.

E. Velldal. *Empirical Realization Ranking*. PhD thesis, University of Oslo, Department of Informatics, 2008.

E. Velldal and S. Oepen. Maximum entropy models for realization ranking. In *Proceedings of MT-Summit X*, Phuket, Thailand, 2005.

E. Velldal and S. Oepen. Statistical ranking in tactical generation. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 517–525, Sydney, Australia, July 2006. Association for Computational Linguistics.

E. Velldal, S. Oepen, and D. Flickinger. Paraphrasing treebanks for stochastic realization ranking. In *Proceedings of the 3rd Workshop on Treebanks and Linguistic Theories (TLT)*, pages 149–160, 2004.

Z. Šidák. Rectangular confidence regions for the means of multivariate normal distributions. *Journal of the American Statistical Association*, pages 626–633, 1967.

M. White. Reining in CCG chart realization. In Anja Belz, Roger Evans, and Paul Piwek, editors, *Natural Language Generation, Third International Conference, INLG 2004, Brockenhurst, United Kingdom, July 14-16, 2004, Proceedings*, volume 3123 of *Lecture Notes in Computer Science*, pages 182–191. Springer, 2004. ISBN 3-540-22340-1.

M. White, R. Rajkumar, and S. Martin. Towards broad coverage surface realization with CCG. In *Proceedings of the Workshop on Using Corpora for NLG: Language Generation and Machine Translation (UCNLG+MT)*, 2007.

Y. Zhou, L. Wu, F. Weng, and H. Schmidt. A fast algorithm for feature selection in conditional maximum entropy modeling. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, EMNLP '03, pages 153–159, Sapporo, Japan, 2003.

J. Zhu, N. Lao, and E. P. Xing. Grafting-light: fast, incremental feature selection and structure learning of Markov random fields. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 303–312. Association for Computing Machinery, 2010.

Publications

- D. de Kok, J. Ma, and G. van Noord. A generalized method for iterative error mining in parsing results. In *Proceedings of the 2009 Workshop on Grammar Engineering Across Frameworks (GEAF 2009)*, pages 71–79, Suntec, Singapore, August 2009. Association for Computational Linguistics

- D. de Kok and G. van Noord. A sentence generator for Dutch. In *Proceedings of the 20th Computational Linguistics in the Netherlands conference (CLIN)*, 2010

- D. de Kok. Feature selection for fluency ranking. In *Proceedings of the 6th International Natural Language Generation Conference (INLG)*, pages 155–163, Trim, Ireland, 2010

- D. de Kok, B. Plank, and G. van Noord. Reversible Stochastic Attribute-Value Grammars. In *Proceedings of the ACL HLT 2011 Conference-Short Papers*, pages 194–199, Portland, OR, USA, 2011. Association for Computational Linguistics

- D. de Kok. Discriminative features in Reversible Stochastic Attribute-Value Grammars. In *Proceedings of the UCNLG+Eval: Language Generation and Evaluation Workshop*, pages 54–63, Edinburgh, United Kingdom, 2011. Association for Computational Linguistics

- G. van Noord, F. Van Eynde, D. de Kok, J. van der Linde, I. Schuurman, E. Tjong Kim Sang, and V. Vandeghinste. Large scale syntactic annotation of written Dutch: Lassy. In Peter Spyns and Jan Odijk, editors, *Essential Speech and Language Technology for Dutch*. Springer, 2012

- J. De Belder, D. de Kok, G. van Noord, F. Nauze, L. van der Beek, and M. Moens. Question answering of informative web pages: How summari-

sation technology helps. In Peter Spyns and Jan Odijk, editors, *Essential Speech and Language Technology for Dutch*. Springer, 2012

Samenvatting in het Nederlands

Het overkoepelende doel van dit proefschrift is aan te tonen dat het mogelijk is een systeem te maken waarin een zinsontleder en zinsgenerator niet alleen een grammatica en lexicon delen, maar ook één statistisch model voor disambiguatie van ontleedde zinnen (parse disambiguatie) en het naar vloeiendheid ordenen van gegenereerde zinnen (fluency ranking). Om te voorkomen dat de ontwikkeling van een dergelijk formalisme slechts een theoretische exercitie is, verwachten we dat zo'n reversibel statistisch model voldoet aan twee vereisten: (1) het model moet even goed presteren als modellen die specifiek voor parse disambiguatie en fluency ranking ontwikkeld zijn; en (2) er moet enige mate van integratie zijn tussen deze taken in een reversibel model.

In dit proefschrift introduceren we het Reversible Stochastic Attribute-Value Grammar formalisme (Hoofdstuk 5). Dit model combineert attibuut-waarde grammatica's met één model voor parse disambiguatie en fluency ranking. We tonen aan dat Reversible Stochastic Attribute-Value Grammar aan de bovengenoemde voorwaarden voldoet. In de experimenten in Hoofdstuk 5 laten we zien dat het omkeerbare component niet significant anders presteert dan componenten die specifiek voor parse disambiguatie en fluency ranking ontwikkeld zijn. In Hoofdstuk 7 isoleren we de meest onderscheidende eigenschappen van omkeerbare modellen. Aan de hand hiervan tonen we aan dat eigenschappen die zowel in parse disambiguatie als in fluency ranking actief zijn, daadwerkelijk in het omkeerbare model gebruikt worden.

Om tot deze conclusie te kunnen komen hadden we echter een systeem nodig dat een attribuut-waarde grammatica, een parser en een generator biedt, alsmede methodes om onderscheidende eigenschappen in modellen te vinden. Het Alpino systeem [van Noord, 2006] biedt een attribuut-waarde grammatica (Hoofdstuk 2) en ontleder voor het Nederlands.

Nadat we in Hoofdstuk 2 het attribuut-waarde grammatica formalisme in Alpino hebben beschreven, introduceren we in Hoofdstuk 3 onze generator voor de Alpino grammatica. Eén van de uitdagingen van generatie is de zoekruimte zodanig te beperken dat generatie uitvoerbaar is. In dit hoofdstuk introduceren we een nieuwe methode voor het begeleiden van generatie, *semantische restrictie*. Deze methode unificeert lexicale items met de relevante delen van de invoer om generatie te sturen. Het resultaat is dat de constructie van incorrecte items zo vroeg mogelijk afgebroken wordt, namelijke gedurende unificatie.

In Hoofdstuk 4 introduceren we ons model voor fluency ranking. Dit model gebruikt zogenaamde *maximum entropy* modellen om features te kunnen benutten die een derivatie beschrijven, zoals: woord en tag trigram modellen, opsommingen van grammaticaregels en diepe syntactische eigenschappen. We hebben de rangschikker geëvalueerd en hebben een foutanalyse uitgevoerd, die aantoont dat de meeste van de overblijvende fouten veroorzaakt zijn door beperkingen van de invoerrepresentatie. We hebben ook verschillende methodes vergeleken voor het schatten van waarschijnlijkheden van zinnen, abstracte representaties, en derivaties in de trainingsdata. We hebben aangetoond dat de methode die een uniforme distributie voor zinnen en abstracte representaties gebruikt significant beter werkt dan de andere methods. Een bijkomstig voordeel is dat deze methode de trainingsdata niet vervormt ten gunste van parse disambiguatie of fluency ranking in omkeerbare modellen.

Om te controleren welke features effectief zijn in parse disambiguatie, fluency ranking en omkeerbare modellen, hebben we feature selectie toegepast. In Hoofdstuk 6 hebben we onze correlatie-selectie methode en vier andere methodes vergeleken. We hebben aangetoond dat de *grafting* methode [Perkins et al., 2003] de meest effectieve methode is voor zowel parse disambiguatie als fluency ranking. We hebben deze methode vervolgens gebruikt om een uitgebreide analyse te maken van de meest onderscheidende eigenschappen in fluency ranking. Deze analyse toont aan dat de vloeiendheid van een zin geschat kan worden met een klein aantal eigenschappen, zoals eigenschappen die de distributie van woord en part-of-speech trigrammen, topicalisatie, de ordening van adjuncten en de ordening in het middenveld beschrijven.

Tenslotte hebben we in Hoofstuk 8 een *error miner* voorgesteld die tekortkomingen en fouten in een grammatica of een lexicon kan detecteren met behulp van een groot aantal ongeannoteerde zinnen. Ons voorstel verschilt van voorgaand werk, omdat het gemengde patronen van een arbitraire lengte kan vinden en goed om kan gaan met schaarste in de data. We vergelijken onze en andere error miners met behulp van een nieuwe kwantitatieve evaluatiemethode, die rekening houdt met de behoeften van grammatica-ontwikkelaars. Vervolgens tonen we met behulp van deze evaluatiemethode aan dat onze error

miner consistent beter presteert dan de competitie.

Groningen Dissertations in Linguistics (Grodil)

1. Henriëtte de Swart (1991). *Adverbs of Quantification: A Generalized Quantifier Approach.*
2. Eric Hoekstra (1991). *Licensing Conditions on Phrase Structure.*
3. Dicky Gilbers (1992). *Phonological Networks. A Theory of Segment Representation.*
4. Helen de Hoop (1992). *Case Configuration and Noun Phrase Interpretation.*
5. Gosse Bouma (1993). *Nonmonotonicity and Categorial Unification Grammar.*
6. Peter Blok (1993). *The Interpretation of Focus: an epistemic approach to pragmatics.*
7. Roelien Bastiaanse (1993). *Studies in Aphasia.*
8. Bert Bos (1993). *Rapid User Interface Development with the Script Language Gist.*
9. Wim Kosmeijer (1993). *Barriers and Licensing.*
10. Jan-Wouter Zwart (1993). *Dutch Syntax: A Minimalist Approach.*
11. Mark Kas (1993). *Essays on Boolean Functions and Negative Polarity.*
12. Ton van der Wouden (1994). *Negative Contexts.*
13. Joop Houtman (1994). *Coordination and Constituency: A Study in Categorial Grammar.*
14. Petra Hendriks (1995). *Comparatives and Categorial Grammar.*
15. Maarten de Wind (1995). *Inversion in French.*
16. Jelly Julia de Jong (1996). *The Case of Bound Pronouns in Peripheral Romance.*
17. Sjoukje van der Wal (1996). *Negative Polarity Items and Negation: Tandem Acquisition.*
18. Anastasia Giannakidou (1997). *The Landscape of Polarity Items.*
19. Karen Lattewitz (1997). *Adjacency in Dutch and German.*
20. Edith Kaan (1997). *Processing Subject-Object Ambiguities in Dutch.*
21. Henny Klein (1997). *Adverbs of Degree in Dutch.*
22. Leonie Bosveld-de Smet (1998). *On Mass and Plural Quantification: The Case of French 'des'/'du'-NPs.*
23. Rita Landeweerd (1998). *Discourse Semantics of Perspective and Temporal Structure.*

24. Mettina Veenstra (1998). *Formalizing the Minimalist Program.*
25. Roel Jonkers (1998). *Comprehension and Production of Verbs in Aphasic Speakers.*
26. Erik F. Tjong Kim Sang (1998). *Machine Learning of Phonotactics.*
27. Paulien Rijkhoek (1998). *On Degree Phrases and Result Clauses.*
28. Jan de Jong (1999). *Specific Language Impairment in Dutch: Inflectional Morphology and Argument Structure.*
29. Hae-Kyung Wee (1999). *Definite Focus.*
30. Eun-Hee Lee (2000). *Dynamic and Stative Information in Temporal Reasoning: Korean Tense and Aspect in Discourse.*
31. Ivilin Stoianov (2001). *Connectionist Lexical Processing.*
32. Klarien van der Linde (2001). *Sonority Substitutions.*
33. Monique Lamers (2001). *Sentence Processing: Using Syntactic, Semantic, and Thematic Information.*
34. Shalom Zuckerman (2001). *The Acquisition of "Optional" Movement.*
35. Rob Koeling (2001). *Dialogue-Based Disambiguation: Using Dialogue Status to Improve Speech Understanding.*
36. Esther Ruigendijk (2002). *Case Assignment in Agrammatism: a Cross-linguistic Study.*
37. Tony Mullen (2002). *An Investigation into Compositional Features and Feature Merging for Maximum Entropy-Based Parse Selection.*
38. Nanette Bienfait (2002). *Grammatica-onderwijs aan allochtone jongeren.*
39. Dirk-Bart den Ouden (2002). *Phonology in Aphasia: Syllables and Segments in Level-specific Deficits.*
40. Rienk Withaar (2002). *The Role of the Phonological Loop in Sentence Comprehension.*
41. Kim Sauter (2002). *Transfer and Access to Universal Grammar in Adult Second Language Acquisition.*
42. Laura Sabourin (2003). *Grammatical Gender and Second Language Processing: An ERP Study.*
43. Hein van Schie (2003). *Visual Semantics.*
44. Lilia Schürcks-Grozeva (2003). *Binding and Bulgarian.*
45. Stasinos Konstantopoulos (2003). *Using ILP to Learn Local Linguistic Structures.*
46. Wilbert Heeringa (2004). *Measuring Dialect Pronunciation Differences using Levenshtein Distance.*
47. Wouter Jansen (2004). *Laryngeal Contrast and Phonetic Voicing: A Laboratory Phonology Approach to English, Hungarian and Dutch.*
48. Judith Rispens (2004). *Syntactic and Phonological Processing in Developmental Dyslexia.*
49. Danielle Bougaïré (2004). *L'approche communicative des campagnes de sensibilisation en santé publique au Burkina Faso: les cas de la planification familiale, du sida et de l'excision.*
50. Tanja Gaustad (2004). *Linguistic Knowledge and Word Sense Disambiguation.*
51. Susanne Schoof (2004). *An HPSG Account of Nonfinite Verbal Complements in Latin.*

52. M. Begoña Villada Moirón (2005). *Data-driven identification of fixed expressions and their modifiability.*
53. Robbert Prins (2005). *Finite-State Pre-Processing for Natural Language Analysis.*
54. Leonoor van der Beek (2005). *Topics in Corpus-Based Dutch Syntax.*
55. Keiko Yoshioka (2005). *Linguistic and gestural introduction and tracking of referents in L1 and L2 discourse.*
56. Sible Andringa (2005). *Form-focused instruction and the development of second language proficiency.*
57. Joanneke Prenger (2005). *Taal telt! Een onderzoek naar de rol van taalvaardigheid en tekstbegrip in het realistisch wiskundeonderwijs.*
58. Neslihan Kansu-Yetkiner (2006). *Blood, Shame and Fear: Self-Presentation Strategies of Turkish Women's Talk about their Health and Sexuality.*
59. Mónika Z. Zempléni (2006). *Functional imaging of the hemispheric contribution to language processing.*
60. Maartje Schreuder (2006). *Prosodic Processes in Language and Music.*
61. Hidetoshi Shiraishi (2006). *Topics in Nivkh Phonology.*
62. Tamás Biró (2006). *Finding the Right Words: Implementing Optimality Theory with Simulated Annealing.*
63. Dieuwke de Goede (2006). *Verbs in Spoken Sentence Processing: Unraveling the Activation Pattern of the Matrix Verb.*
64. Eleonora Rossi (2007). *Clitic production in Italian agrammatism.*
65. Holger Hopp (2007). *Ultimate Attainment at the Interfaces in Second Language Acquisition: Grammar and Processing.*
66. Gerlof Bouma (2008). *Starting a Sentence in Dutch: A corpus study of subject- and object-fronting.*
67. Julia Klitsch (2008). *Open your eyes and listen carefully. Auditory and audiovisual speech perception and the McGurk effect in Dutch speakers with and without aphasia.*
68. Janneke ter Beek (2008). *Restructuring and Infinitival Complements in Dutch.*
69. Jori Mur (2008). *Off-line Answer Extraction for Question Answering.*
70. Lonneke van der Plas (2008). *Automatic Lexico-Semantic Acquisition for Question Answering.*
71. Arjen Versloot (2008). *Mechanisms of Language Change: Vowel reduction in 15th century West Frisian.*
72. Ismail Fahmi (2009). *Automatic term and Relation Extraction for Medical Question Answering System.*
73. Tuba Yarbay Duman (2009). *Turkish Agrammatic Aphasia: Word Order, Time Reference and Case.*
74. Maria Trofimova (2009). *Case Assignment by Prepositions in Russian Aphasia.*
75. Rasmus Steinkrauss (2009). *Frequency and Function in WH Question Acquisition. A Usage-Based Case Study of German L1 Acquisition.*
76. Marjolein Deunk (2009). *Discourse Practices in Preschool. Young Children's Participation in Everyday Classroom Activities.*
77. Sake Jager (2009). *Towards ICT-Integrated Language Learning: Developing an Implementation Framework in terms of Pedagogy, Technology and Environment.*

78. Francisco Dellatorre Borges (2010). *Parse Selection with Support Vector Machines.*
79. Geoffrey Andogah (2010). *Geographically Constrained Information Retrieval.*
80. Jacqueline van Kruiningen (2010). *Onderwijsontwerp als conversatie. Probleemoplossing in interprofessioneel overleg.*
81. Robert G. Shackleton (2010). *Quantitative Assessment of English-American Speech Relationships.*
82. Tim Van de Cruys (2010). *Mining for Meaning: The Extraction of Lexico-semantic Knowledge from Text.*
83. Therese Leinonen (2010). *An Acoustic Analysis of Vowel Pronunciation in Swedish Dialects.*
84. Erik-Jan Smits (2010). *Acquiring Quantification. How Children Use Semantics and Pragmatics to Constrain Meaning.*
85. Tal Caspi (2010). *A Dynamic Perspective on Second Language Development.*
86. Teodora Mehotcheva (2010). *After the fiesta is over. Foreign language attrition of Spanish in Dutch and German Erasmus Students.*
87. Xiaoyan Xu (2010). *English language attrition and retention in Chinese and Dutch university students.*
88. Jelena Prokić (2010). *Families and Resemblances.*
89. Radek Šimík (2011). *Modal existential wh-constructions.*
90. Katrien Colman (2011). *Behavioral and neuroimaging studies on language processing in Dutch speakers with Parkinson's disease.*
91. Siti Mina Tamah (2011). *A Study on Student Interaction in the Implementation of the Jigsaw Technique in Language Teaching.*
92. Aletta Kwant (2011). *Geraakt door prentenboeken. Effecten van het gebruik van prentenboeken op de sociaal-emotionele ontwikkeling van kleuters.*
93. Marlies Kluck (2011). *Sentence amalgamation.*
94. Anja Schüppert (2011). *Origin of asymmetry: Mutual intelligibility of spoken Danish and Swedish.*
95. Peter Nabende (2011). *Applying Dynamic Bayesian Networks in Transliteration Detection and Generation.*
96. Barbara Plank (2011). *Domain Adaptation for Parsing.*
97. Çağrı Çöltekin (2011). *Catching Words in a Stream of Speech: Computational simulations of segmenting transcribed child-directed speech.*
98. Dörte Hessler (2011). *Audiovisual Processing in Aphasic and Non-Brain-Damaged Listeners: The Whole is More than the Sum of its Parts.* Herman Heringa (2012). *Appositional constructions.*
99. Herman Heringa (2012). *Appositional constructions.*
100. Diana Dimitrova (2012). *Neural Correlates of Prosody and Information Structure.*
101. Harwintha Anjarningsih (2012). *Time Reference in Standard Indonesian Agrammatic Aphasia.*
102. Myrte Gosen (2012). *Tracing learning in interaction. An analysis of shared reading of picture books at kindergarten.*
103. Martijn Wieling (2012). *A Quantitative Approach to Social and Geographical Dialect Variation.*

104. Gisi Cannizzaro (2012). *Early word order and animacy.*
105. Kostadin Cholakov (2012). *Lexical Acquisition for Computational Grammars. A Unified Model.*
106. Karin Beijering (2012). *Expressions of epistemic modality in Mainland Scandinavian. A study into the lexicalization-grammaticalization-pragmaticalization interface.*
107. Veerle Baaijen (2012). *The development of understanding through writing.*
108. Jacolien van Rij (2012). *Pronoun processing: Computational, behavioral, and psychophysiological studies in children and adults.*
109. Ankelien Schippers (2012). *Variation and change in Germanic long-distance dependencies.*
110. Hanneke Loerts (2012). *Uncommon gender: Eyes and brains, native and second language learners, & grammatical gender.*
111. Marjoleine Sloos (2013). *Frequency and phonological grammar: An integrated approach. Evidence from German, Indonesian, and Japanese.*
112. Aysa Arylova (2013). *Possession in the Russian clause. Towards dynamicity in syntax.*
113. Daniël de Kok (2013). *Reversible Stochastic Attribute-Value Grammars.*

Grodil
Secretary of the Department of General Linguistics
Postbus 716
9700 AS Groningen
The Netherlands


富嶽三十六景
神奈川沖
浪裏