Detecting and Diagnosing Grammatical Errors for Beginning Learners of German: 
From Learner Corpus Annotation to Constraint Satisfaction Problems

Dissertation

Presented in Partial Fulfillment of the Requirements for 
the Degree Doctor of Philosophy in the 
Graduate School of The Ohio State University

By

Adriane Boyd, M.A., M.S.

Linguistics Graduate Program
The Ohio State University
2012

Dissertation Committee:
Walt Detmar Meurers, Co-Advisor
Michael White, Co-Advisor
Kathryn Corl
William Schuler
Copyright by
Adriane Boyd
2012
Abstract

This thesis presents a corpus of beginning learner German with a reliable error annotation scheme and an approach for detecting and diagnosing grammatical errors in learner language. A constraint-based dependency parser provides the foundation for a flexible and modular analysis of German by representing parsing as a constraint satisfaction problem. The grammar checker, Fledgling, detects and diagnoses errors using constraint relaxation with a general-purpose conflict detection algorithm. Fledgling is developed and evaluated using authentic learner productions from the learner corpus. It judges grammaticality correctly for 80% of sentences and is 82–91% accurate in determining whether a sentence contains selection, agreement, or word order errors.
Dedication

to my parents
Acknowledgements

My advisor, Detmar Meurers, and I share an interest in reading acknowledgements, since they reveal the fascinating human side of scientific research. I am afraid I am not going to do my own acknowledgements justice, since it is impossible to thank everyone who enriched my years in graduate school.

First and foremost, I would like to thank Detmar for his support throughout my time in graduate school, in Columbus and more recently in Tübingen. He has provided a great deal of guidance and encouragement in all areas of academic life. I am also grateful to my committee, Mike White, Kathy Corl, and William Schuler, for their feedback and support, in particular Kathy for her assistance in collecting the German learner data that was essential for this thesis.

The Linguistics Department at Ohio State provided a wonderful environment for graduate students to become more skilled researchers and teachers. I am thankful for their continued support while I completed my master’s thesis in Computer Science and Engineering and while I worked on my dissertation in Tübingen. I am grateful for funding provided by the OSU Dean’s Distinguished University Fellowship, the OSU Linguistics Department, the National Science Foundation, and the Seminar für Sprachwissenschaft at the University of Tübingen.

It was a pleasure to go through graduate school with such a dedicated, enthusiastic cohort: David Durian, Anouschka Foltz, Kathleen Hall, Ilana Heintz, Julia Papke, Sharon
Ross, and Elizabeth Smith. DJ Hovermale deserves thanks for his assistance in Columbus as I was wrapping up my dissertation from Tübingen, and I would also like to thank the students at the University of Tübingen who contributed to the EAGLE corpus annotation: Cornelius Fath, Stefanie Wolf, and Marion Zepf.

Finally, I would like to thank my parents, Mark and Barbara Boyd, for their encouragement and unwavering support, and Arnd Bergmann, for everything.
Vita

2003 ................................................. B.A., Linguistics and German, University of North Carolina at Chapel Hill
2007 ................................................. M.A., Linguistics, The Ohio State University
2008 ................................................. M.S., Computer Science and Engineering, The Ohio State University

Publications


Fields of Study

Major Field: Linguistics
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Dedication</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>iv</td>
</tr>
<tr>
<td>Vita</td>
<td>vi</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>viii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xiv</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xvi</td>
</tr>
<tr>
<td>List of Abbreviations</td>
<td>xxi</td>
</tr>
<tr>
<td>Chapter 1: Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.1.1 Natural Language Processing in Computer-Aided Language Learning</td>
<td>4</td>
</tr>
<tr>
<td>1.1.2 Syntactic Analysis of Learner Language</td>
<td>6</td>
</tr>
<tr>
<td>1.2 Design of the Fledgling Grammar Checker</td>
<td>13</td>
</tr>
<tr>
<td>1.3 EAGLE: An Error-Annotated Corpus of Beginning Learner German</td>
<td>13</td>
</tr>
<tr>
<td>1.4 Outline</td>
<td>14</td>
</tr>
<tr>
<td>I Background</td>
<td></td>
</tr>
</tbody>
</table>

Chapter 2: Modeling German Grammar                                     17
### Chapter 2: A Brief Overview of German Syntax

- **2.1 A Brief Overview of German Syntax** ........................................... 18
  - **2.1.1 Declension and Agreement in Noun Phrases** .......................... 18
  - **2.1.2 Verb Conjugation and Subject-Verb Agreement** ....................... 23
  - **2.1.3 Relative Clauses** .................................................. 25
  - **2.1.4 Preposition Subcategorization** ....................................... 26
  - **2.1.5 Verb Subcategorization** ............................................ 28
  - **2.1.6 Word Order** ...................................................... 31
- **2.2 Types of Errors** ............................................................ 35
  - **2.2.1 Motivation for Error Typology** ....................................... 36
- **2.3 Summary** ................................................................. 36

### Chapter 3: Grammar Checking in ICALL

- **3.1 Approaches to Error Detection and Diagnosis in ICALL** ................. 37
  - **3.1.1 Demonstration Grammar** ........................................... 38
  - **3.1.2 Mal-Rules** .......................................................... 41
  - **3.1.3 Parser Modification** ............................................ 46
  - **3.1.4 Meta-Rules** ...................................................... 51
  - **3.1.5 Constraint Relaxation** ............................................ 53
- **3.2 Parser-Based ICALL for German** .......................................... 56
  - **3.2.1 Die Sprachmaschine: Harroff (1986)** ................................ 56
  - **3.2.2 Spion and Syncheck: Spion and Syncheck: Sanders and Sanders** (1987); Sanders (1991); Sanders and Sanders (1995) .......................... 57
  - **3.2.3 Herr Kommissar: DeSmedt (1995)** ................................ 58
  - **3.2.4 Schwend (1995)** ................................................. 59
  - **3.2.5 BRIDGE: Weinberg et al. (1995); Sams (1995)** .................... 61
  - **3.2.6 Menzel and Schröder (1998, 1999)** ................................ 62
  - **3.2.7 German Tutor/E-Tutor: Heift (1998)** ................................ 65
  - **3.2.8 Textana: Schulze (2001)** ......................................... 69
  - **3.2.9 DiBEx: Klenner (2003, 2004)** ................................... 70
  - **3.2.10 Fortmann and Forst (2004)** ..................................... 71
  - **3.2.11 PromisD: Reuer (2003, 2005)** .................................. 74
- **3.3 Summary** ................................................................. 77

### Chapter 4: Constraint Satisfaction Problems

- **4.1 Constraint Satisfaction Problems** ........................................ 80
- **4.2 Solving Constraint Satisfaction Problems** ................................ 82
  - **4.2.1 Search** ............................................................. 83
  - **4.2.2 Consistency Techniques** .......................................... 87
  - **4.2.3 Combining Consistency and Search Techniques** .................... 90
  - **4.2.4 Constraint Propagation and Distribution** ......................... 98
Chapter 8: Error Analysis for the EAGLE Corpus

8.1 Overview of Error Distribution
8.1.1 Comparison to Previous Analyses
8.2 Detailed Error Analysis
8.2.1 Selection Errors
8.2.2 Agreement Errors
8.2.3 Word Order Errors
8.2.4 Word Form Errors
8.2.5 Punctuation Errors
8.3 Relevant Phenomena for Grammar Development and Grammar Checking
8.3.1 Sentence-Level Analysis
8.3.2 Verb Subcategorization Frames
8.3.3 Noun Phrases
8.3.4 Subject-Verb Agreement
8.3.5 Determiner-Adjective-Noun Agreement
8.3.6 Relative Pronoun-Antecedent Agreement
8.3.7 Reflexive-Subject Agreement
8.3.8 Topological Field Order
8.3.9 Mittelfeld Order
8.4 Summary

III Error Detection and Diagnosis

Chapter 9: Constraint-Based Grammar for Beginning Learners of German

9.1 Grammar Development
9.1.1 Coverage of Existing XDG German Grammars
9.1.2 Grammar Extensions and Modifications
9.1.3 Evaluating XDG Principles
9.1.4 Customizing Constraints for Error Diagnosis
9.2 Extending the Lexicon
9.2.1 Lexical Coverage
9.2.2 Morphological Generation
9.2.3 Verb Subcategorization Frame Database
9.2.4 Additional Vocabulary
12.2.1 Grammar Development ........................................... 250
12.2.2 Error Detection and Diagnosis ................................. 251
12.3 Outlook for ICALL .................................................. 254

Bibliography ..................................................................... 255
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Declension of <em>Mann</em></td>
</tr>
<tr>
<td>2.2</td>
<td>Declension of the Definite Determiner</td>
</tr>
<tr>
<td>2.3</td>
<td>Declension of the Indefinite Determiner</td>
</tr>
<tr>
<td>2.4</td>
<td>Adjective Endings</td>
</tr>
<tr>
<td>2.5</td>
<td>Combined Determiner-Adjective-Noun Declension for <em>der alte Mann</em></td>
</tr>
<tr>
<td>2.6</td>
<td>Combined Determiner-Adjective-Noun Declension for <em>die alte Frau</em></td>
</tr>
<tr>
<td>2.7</td>
<td>Conjugation of <em>kaufen</em></td>
</tr>
<tr>
<td>2.8</td>
<td>Subject-Verb Agreement for Present Tense of <em>kaufen</em></td>
</tr>
<tr>
<td>3.1</td>
<td>Overview of Parser-Based ICALL Approaches for German</td>
</tr>
<tr>
<td>7.1</td>
<td>Classification of Non-Word Spelling Errors in EAGLE</td>
</tr>
<tr>
<td>7.2</td>
<td>Annotation Layers in EAGLE</td>
</tr>
<tr>
<td>7.3</td>
<td>Interannotator Agreement for Grammaticality</td>
</tr>
<tr>
<td>7.4</td>
<td>$\alpha$ for Error Description by Error Typology Depth</td>
</tr>
<tr>
<td>8.1</td>
<td>Workbook vs. Essay Errors by Top-Level Category in EAGLE</td>
</tr>
<tr>
<td>10.1</td>
<td>Mapping Between Principles and EAGLE Error Types</td>
</tr>
</tbody>
</table>
11.1 Grammaticality Scores .............................................. 240
11.2 Sources of Errors in Grammaticality Judgement for Test10 .............. 241
11.3 Evaluation by Error Type ............................................. 243
11.4 Test10 Evaluation after EAGLE Filtering .............................. 246
11.5 Intersection by Error Typology Depth ................................. 246
11.6 MASI Scores by Error Typology Depth ............................... 247
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Topological Fields for Example (19)</td>
<td>34</td>
</tr>
<tr>
<td>3.1</td>
<td>Demonstration Context-Free Grammar (CFG)</td>
<td>39</td>
</tr>
<tr>
<td>3.2</td>
<td>Grammatical Sentence (21a)</td>
<td>40</td>
</tr>
<tr>
<td>3.3</td>
<td>Grammatical Sentence (21d)</td>
<td>40</td>
</tr>
<tr>
<td>3.4</td>
<td>Sample Error Types with Associated Mal-Rules</td>
<td>42</td>
</tr>
<tr>
<td>3.5</td>
<td>(22b) with Mal-Rules</td>
<td>43</td>
</tr>
<tr>
<td>3.6</td>
<td>(22a) with Mal-Rules</td>
<td>43</td>
</tr>
<tr>
<td>3.7</td>
<td>Mal-Rules in the Lexicon</td>
<td>44</td>
</tr>
<tr>
<td>3.8</td>
<td>(22a) with Mal-Rules in the Lexicon</td>
<td>44</td>
</tr>
<tr>
<td>3.9</td>
<td>Parsing with Modified Feature Unification</td>
<td>47</td>
</tr>
<tr>
<td>3.10</td>
<td>Modified Chart Parsing for a Missing Word Error (22e)</td>
<td>51</td>
</tr>
<tr>
<td>3.11</td>
<td>Meta-Rules for a Missing Word Error (22e)</td>
<td>53</td>
</tr>
<tr>
<td>3.12</td>
<td>Demonstration Constraint-Based Grammar</td>
<td>54</td>
</tr>
<tr>
<td>3.13</td>
<td>Constraint Relaxation for Subject-Verb Agreement Error (22a)</td>
<td>55</td>
</tr>
<tr>
<td>3.14</td>
<td>Initial Dependency Representation for Eliminative Parsing</td>
<td>63</td>
</tr>
</tbody>
</table>
List of Abbreviations

BVL ............ Buchholz Verb Lexicon
CALL ............ Computer-Aided Language Learning
CDG ............ Constraint Dependency Grammar
CFG ............ Context-Free Grammar
CSP ............ Constraint Satisfaction Problem
EAGLE ........ Error Annotated German Learner Corpus
HPSG ............ Head-Driven Phrase Structure Grammar
ICALL ........ Intelligent Computer-Aided Language Learning
LFG ............ Lexical Functional Grammar
MASI ............ Measuring Agreement on Set-Valued Items
NLTK ............ Natural Language Toolkit
PCFG ............ Probabilistic Context-Free Grammar
POS ............ Part of Speech
WCDG ............ Weighted Constraint Dependency Grammar
XDG ............ Extensible Dependency Grammar
XDK ............ XDG Development Kit
Chapter 1: Introduction

Analyzing learner language in order to detect and diagnose learner errors requires knowledge about the ways in which learner language differs from native language. This thesis examines the types of grammatical errors made by beginning learners of German through the creation of an error-annotated corpus and explores the automatic detection and diagnosis of errors using parsing with constraint relaxation. Automatic analysis of learner language is particularly applicable in the area of intelligent computer-aided language learning (ICALL), where natural language processing techniques such as grammar checking allow an ICALL system to provide detailed feedback to learners.

The two main contributions of this thesis are an error-annotated learner corpus and a constraint-based grammar checking approach that models parsing as a constraint satisfaction problem. First, the grammatical errors produced by beginning learners of German are examined by creating a corpus of learner data from students in introductory university-level courses (EAGLE, Boyd, 2010a). A new error annotation scheme is developed for the corpus, which can reliably annotate the multiple, ambiguous, overlapping errors common in beginning learner productions (Boyd, 2010b). The interannotator agreement for the annotation scheme is evaluated in detail, which is a crucial aspect of the annotation process that has been insufficiently addressed in much of the previous work on error annotation.
Second, a grammar checker, *Fledgling*, is developed using a dependency parser (Debusmann, 2006) that views parsing as finding solutions to constraint satisfaction problems. Grammatical errors are detected and diagnosed by finding conflicting constraints in the constraint satisfaction problems. Authentic learner data from the EAGLE corpus is used to develop and evaluate the grammar checker.

1.1 Motivation

Natural language processing components such as grammar checkers can be integrated into computer-aided language learning (CALL) systems in order to extend the kinds of activities that can be evaluated automatically. CALL systems without natural language processing components cope with the wide range of possible learner input either by restricting the types of activities or by requiring the instructor to evaluate the learner’s response if it does not match any of the stored correct responses (cf. Amaral and Meurers, 2011). Restricting the types of activities can reduce the naturalness of the exercises for the learner and requiring instructor input for responses delays the feedback to the learner, reducing the CALL system to an online dropbox. CALL systems typically provide little informative feedback beyond *correct/incorrect* and do not differentiate between different types of learner errors. A misspelling or typo may be given the same score and feedback as a wildly incorrect answer because the system only has the capability to compare the learner’s response letter-by-letter or word-by-word against a stored key.

Augmenting a CALL system with the ability to generate a linguistic analysis of the learner’s input to the system allows the system to provide detailed, immediate feedback and to handle a wider range of activities. Nagata (1993) shows that feedback targeted to specific errors
improved learners’ performance and the learners preferred the more detailed error messages because it was otherwise difficult to pinpoint errors. A wider range of activities allows for more natural exercises for the learner. Since most learner errors at the beginning level are in form rather than in content (Juozulynas, 1994), incorporating a syntactic parser into a CALL system should allow for targeted feedback and less constrained activities for the majority of beginning learner errors.

Generating an appropriate linguistic analysis of learner language presents a significant challenge for a parser for several reasons. First, learner language and learner errors differ significantly from the language and errors produced by a native speaker. Parsers developed for native speakers often cannot cope with the productions of non-native learners. Second, the types of errors produced may also vary significantly from learner to learner depending on factors such as the learner’s native language or whether the learner is a beginner or advanced student of the second language. Finally, in a learning context, it is typically not sufficient for a parser to simply fail on ill-formed language because the failure does not provide any detailed feedback about the particular errors in the sentence. A robust parser would not fail on ill-formed language, however it would be unable to differentiate grammatical and ungrammatical input.

This thesis focuses on the final challenge: developing a parser-based approach that can detect and diagnose errors in learner language, in particular for beginning learners of German. Menzel and Schröder (1998) identifies the two opposing requirements for syntactic parsers in CALL: high precision is needed to distinguish grammatical from ungrammatical sentences while robustness is needed to provide a partial analysis of an ungrammatical sentence for feedback purposes. The grammar checker developed in this thesis, Fledgling, uses
a high-precision constraint-based parser with constraint relaxation to diagnose errors and provide partial analyses for ill-formed input. The grammar checker approach is evaluated using EAGLE, the corpus of error-annotated learner data.

1.1.1 Natural Language Processing in Computer-Aided Language Learning

A CALL system that intends to provide detailed feedback with no linguistic analysis component requires a great deal of effort on the part of the instructor to envisage possible errors and provide appropriate feedback for each possible answer to each exercise.

If the learner’s input to the system\(^1\) does not match one of the stored answers, then the system may not be able to evaluate the learner’s response or may only be able to provide vague feedback, such as the point in the string where the learner’s response diverges from a correct response. Even for a simple fill-in-the-blank exercise, it is difficult and tedious for the instructor to imagine most of the possible learner responses. For an exercise with freer input such as a short essay, it becomes impossible to envisage learner responses with any success.

(Nagata, 2009) shows the difficulties in envisaging responses for a Japanese language exercise where the learner is given a translation exercise that requires a one-sentence response: *Tell the teacher that you were made by your mother to wash many big dishes.* After taking into account all the possible lexical, orthographic, and word-order variation, there are a total of 6,048 correct responses. When all potential combinations of errors are envisaged for

\(^1\)Throughout this thesis, *input* will refer to the language produced by the learner that is provided as input to a CALL or ICALL system for analysis. In contrast to its use in literature on foreign language teaching and second language acquisition, the term *input* will not be used to refer to the foreign language input provided to the learner as part of a language course.
this one-sentence response, the total number of possible incorrect responses is 967,680. It is clearly not feasible for a language teacher to create manually create meaningful feedback for each of these possible responses.

The online workbook that accompanies the introductory German textbook Deutsch: Na klar! (Quia Books, 2007) provides an illustration of the types of activities available in a typical CALL system. The workbook offers multiple choice, multiple select, fill-in-the-blank, short answer, and essay questions. The question types in the Quia Workbook can be divided into three groups:

- **Computer graded:** The multiple choice, multiple select, and fill-in-the-blank exercises are graded automatically. If the student answers a question incorrectly, the possible correct answers are shown next to the student’s response with no further feedback specific to the student’s response.²

- **Computer graded with possible instructor review:** The system attempts to grade short answer questions by comparing the student's answer to stored keys. If the student’s answer does not match a key exactly, the question is flagged for instructor review. Until instructor review, the response is counted as incorrect, but a grade range is provided to show the student how much the overall grade could improve after instructor review. Again, the feedback consists only of showing the possible correct responses next to the student’s response.

²There is nothing preventing the Quia workbook or a workbook like it from providing more feedback in these cases, but these exercises in the Deutsch: Na klar! workbook do not provide any further feedback.
• **Instructor graded:** The essay questions are graded by instructor only. No feedback is given other than confirmation of submission and the student must wait for instructor review. Instructors are given a small text box to enter any feedback, which makes it difficult and tedious to specify corrections for a longer text, and the student only sees the response if he/she makes a special effort to go back to the exercise and check for feedback.

The goal of the Fledgling grammar checker to extend the kinds of questions for which such an online workbook can provide automatic grading and to improve the feedback provided. In order to achieve this, it is necessary to incorporate natural language processing tools such as a syntactic parser.

### 1.1.2 Syntactic Analysis of Learner Language

A syntactic parser allows a CALL system to provide a deeper analysis of the learner input. There are a variety of existing approaches used in parser-based CALL systems to handle syntactically ill-formed input: mal-rules, modified parsing algorithms, and constraint relaxation. Mal-rule approaches (e.g., Schwind, 1988; Schneider and McCoy, 1998) and the modified parsing algorithms that handle missing, extra, and incorrectly ordered words (e.g., Reuer, 2005) typically use parsers that rely on a phrase-structure grammar (Chomsky, 1957) or a phrase structure backbone such as in Lexical Functional Grammar (Kaplan and Bresnan, 1983), which cause a number of difficulties for error detection and diagnosis. In particular, phrase-structure approaches suffer from a lack of modularity and difficulty handling word order errors and errors outside the local domain. Before turning to these
drawbacks, the following section discusses the differences between phrase-structure and constraint-based grammars.

**Phrase-Structure vs. Constraint-Based Grammars**

The parser-based CALL systems rely on grammars with a phrase-structure backbone or grammars that consist of a set of constraints on the appearance of a well-formed sentence, which will be referred to as a *constraint-based grammar*.³ These two types of grammar differ fundamentally in how they determine whether an input sentence is well-formed and what the possible analyses of the sentence may be structured.

A phrase-structure grammar uses rules (i.e., axioms) to license specific analyses of input sentences and the “well-formedness of the utterance is shown by proving that it is a consequence of the grammar axioms” (Johnson, 1994, p. 233). If a phrase-structure grammar does not contain any rules, it does not license any sentences. Each time a new rule is added, the grammar licenses additional sentences and/or additional analyses.

In contrast, a constraint-based grammar only permits analyses that conform to the constraints in the grammar. Parsing is driven by the notion of *satisfiability*: a sentence that satisfies all the constraints in the grammar is a well-formed sentence. An empty constraint-based grammar would license all possible sequences of all lexical items. Each additional constraint reduces the number of sentences and/or analyses licensed by the grammar.

³As explained in Johnson (1994), both “validity-based” approaches, where the grammar consists of a set of axioms such as phrase-structure rules, and “satisfiability-based” approaches, where the grammar is a set of constraints, can be expressed using constraints and thus both are “constraint-based” approaches (p. 221–222). In this thesis, “constraint-based grammar” will refer to solely to a “satisfiability-based” approach where a grammar consists of a set of constraints.
Drawbacks to Phrase Structure Analysis

Modularity  The first issue for phrase-structure grammars is that phrase-structure rules combine different types of grammatical constraints into a single, indivisible unit. Even a relatively simple phrase-structure rule such as $NP \rightarrow \text{Det}[sg] \text{Noun}[sg]$ makes specifications about three different types of grammatical features: subcategorization, agreement, and word order. First, it specifies that the noun requires a determiner; second, it describes the agreement required between the determiner and the noun; finally, it specifies the word order within this phrase, i.e., that the determiner should come before the noun. Relying on a phrase-structure analysis makes it difficult to disentangle different types of errors in learner input. Error diagnosis components in an ICALL system benefit from a more modular analysis than is provided by phrase-structure grammar.

Local domain  Phrase structure rules also restrict the kinds of errors that can be easily handled to the local domain, which is particularly problematic for word order errors. A grammar with a rule such $S \rightarrow NP \ VP$ can be augmented with a mal-rule $S \rightarrow VP \ NP$ to detect an inversion of the subject and entire verb phrase, but cannot easily capture an error where the subject is reordered within the verb phrase. If the rule $S \rightarrow NP \ VP$ licenses the grammatical sentence in example (1a), the mal-rule $S^* \rightarrow VP \ NP$ licenses (1b) but there is no simple mal-rule that can license (1c). Without making major changes to the syntactic analysis (e.g., flattening the tree), mal-rules only license reordering of sisters in a local tree.

(1)  
   a. Mary [loves cats].
   c. * loves Mary cats.
As German word order is dependent on the clause type (main vs. subordinate), the context necessary to diagnose an error is an entire clause or sentence, which goes well beyond the local domain in a phrase-structure rule.

**Freer word order**  In addition to the general problems mentioned up to this point, German’s freer word order poses a particular problem for a phrase-structure analysis, as phrase-structure trees are not well-suited to describe the order of languages with free(r) word order. In his book *Linear Syntax*, Kathol (2000) observes:

“[T]he linear organization in a language such as German has properties which, albeit correlated with notions of hierarchically defined constituency, are ultimately not derivable from the latter.”

This observation is illustrated by the difficulties that existing German treebanks have in performing a phrase-structure based analysis of German (cf. Boyd, 2007) given the relatively high frequency of non-projective structures (Havelka, 2007). The NEGRA and TIGER Treebanks (Brants et al., 1997, 2002) allow crossing branches in their trees in order to handle discontinuous structures, which cannot be handled by traditional context-free parsing algorithms. The TüBa-D/Z Corpus (Telljohann et al., 2005) introduces intermediate topological field nodes to describe the word order and uses edge labels to relate discontinuous constituents.
Constraint Relaxation

Constraint relaxation is an alternative to approaches that rely on modifications to parsers with context-free backbones. Constraint relaxation techniques identify constraints that cannot be satisfied in an ungrammatical sentence and retract or relax them in order to provide a partial analysis. Constraint relaxation is used to detect agreement and case selection errors in many existing ICALL approaches as part of modified unification algorithms (e.g., Heift, 2003a; Reuer, 2003a, 2005) or as the core approach with a purely constraint-based grammar (e.g. Menzel and Schröder, 1998).

Constraint relaxation has the advantage that combinations of errors do not need to be envisaged as with mal-rules. Mal-rules need to elaborate each specific combination of errors, such as an agreement error and a word order error that appear in the same phrase. If mal-rules are created for a rule such as $NP \rightarrow Det[sg]\ N[sg]$, there will be three mal-rules: one for the agreement error ($NP \rightarrow Det[sg]\ N[pl]$), one for a word order error ($NP \rightarrow N[sg]\ Det[sg]$), and a third mal-rule for both errors occurring together ($NP \rightarrow N[pl]\ Det[sg]$). A constraint relaxation approach can specify that an agreement constraint may be relaxed and that a word order constraint can also be relaxed, but the modular nature of the constraints does not require the elaboration of each possible combination of error types. A purely constraint-based approach also has the advantage that all types of syntactic errors can be handled using the same constraint relaxation algorithm.
Statistical Dependency Parsing

Given the drawbacks to a phrase-structure analysis for German, a dependency analysis offers an attractive alternative and is one that has been in use for languages with freer word order such as Czech (cf. Prague Dependency Treebank, Böhmová et al., 2003). In the past decade there has also been a great deal of interest in statistical dependency parsing (cf. numerous CoNLL shared tasks on multilingual dependency parsing: Buchholz and Marsi, 2006; Nivre et al., 2007; Hajič et al., 2009), which has led to many improvements in dependency parsing algorithms.

The dependency representations used in statistical dependency parsing typically focus on syntactic or semantic relations with no formal model of word order aside from notions of whether a dependent is left or right of its head or whether a subtree is projective. As (Debusmann, 2006) observes, dependency grammars “have for a long time lacked a declarative and workable account of word order.” Some work has started to address this (Bröker, 1998; Gerdes and Kahane, 2001; Duchier and Debusmann, 2001a), but these approaches have not been integrated into mainstream statistical dependency parsing. Bröker (1998); Gerdes and Kahane (2001); Duchier and Debusmann (2001a) have in common that they treat the traditional syntactic/semantic dependency parse as unordered and derive possible linear realizations from the unordered tree.

Aside from word order, the main drawback for statistical parsers, either dependency or PCFG-based, is that they are typically intended to be robust and have little ability to distinguish grammatical from ungrammatical sentences, which is a basic requirement for a parser used in a grammar checker for language learners.
Alternatives to Syntactic Analysis

There are several alternatives to syntactic analysis that typically focus on a narrow range of error types:

Finite state approaches have been used in many grammar checkers developed for native speakers such as FiniteCheck (Sofkova Hashemi, 2003), a grammar checker developed for Swedish primary school children. FiniteCheck uses a robust parser to provide an initial analysis and finite state transducers to distinguish grammatical and ungrammatical constructions. The open source grammar checker LanguageTool (Naber, 2003) uses a part-of-speech tagger along with regular expressions to detect grammatical and style errors. In general, finite state approaches require all errors to be envisaged in explicit rules, which can lead to the same combinatorial explosion of rules as for mal-rules when there are multiple, overlapping errors.

Machine learning approaches have been developed to detect and correct certain subsets of grammatical errors, typically part-of-speech-based errors such as article or preposition errors. There is a wide range of research in preposition and other function word prediction (Gamon et al., 2008; De Felice, 2008; Tetreault and Chodorow, 2008; Bergsma et al., 2009; Elghafari et al., 2010). Lee (2009) presents a range of machine learning approaches used to detect and correct errors in function words in English. Linguistic features from the context around a function word, including analysis by shallow or deep parser, are used to detect and correct errors. One drawback to machine learning approaches is that they are specific to particular phenomena in a particular language and generally only suited for a small range of the types of errors made by learners.
1.2 Design of the Fledgling Grammar Checker

A dependency analysis with non-projective structures is well-suited for the analysis of German and a purely constraint-based allows an error diagnosis algorithm to find a wide range of errors using the same approach, namely constraint relaxation. These desires are the main motivating factors in the choice of the constraint-based parsing approach underlying the Fledgling grammar checker.

The Extensible Dependency Grammar (XDG) formalism (Duchier, 1999; Debusmann et al., 2004; Debusmann, 2006) was chosen as the grammar formalism for Fledgling because it is a purely constraint-based grammar formalism that uses linked dependency graphs, called multigraphs, to represent different levels of linguistic analysis, which can range from syntactic dependencies to predicate-argument structure to intonation. Constraints in different levels of analysis work in parallel to constrain possible parses. The division of syntactic dependencies from linear order in XDG provided an especially attractive motivation for its use in error diagnosis, since it makes it possible to diagnose word order errors without an unwanted interaction with subcategorization or agreement constraints.

1.3 EAGLE: An Error-Annotated Corpus of Beginning Learner German

The second focus of this thesis is the creation of an error-annotated corpus of beginning learner German that can be used in the development and evaluation of the Fledgling grammar checker. A significant weakness in much previous research on parser-based ICALL for German and other languages is the lack of an evaluation using real or realistic learner
data. Corpora are available for advanced learner German (e.g., Siemen et al., 2006), but there were no publicly available corpora of beginning learner German.

The EAGLE Corpus (Boyd, 2010a, see chapter 7) was collected from German learners at The Ohio State University. The learner data, approximately 60,000 tokens from exam essays and online workbook activities, was annotated using a new syntactic error annotation scheme that relies on well-defined linguistic units to structure the error typology and distinguish ambiguous error types. A multi-layer, standoff format (Lüdeling et al., 2005) is used to handle multiple, overlapping, and ambiguous errors. The EAGLE error annotation scheme includes clear guidelines for ambiguous and otherwise difficult cases in order to support consistent annotation, which can be challenging for error-annotated learner corpora (Lüdeling, 2008; Meurers, 2009).

1.4 Outline

The thesis consists of three parts: background, the error-annotated learner corpus, and the grammar checker. Following this introduction, the first part of the thesis begins with an introduction to the relevant background in the areas of German grammar, intelligent computer-aided language learning, and constraint-based parsing with constraint satisfaction problems (CSPs). Chapter 2 provides a short introduction to German grammar and chapter 3 describes approaches to syntactic error detection and diagnosis in Intelligent Computer-Aided Language Learning (ICALL) along with an overview of parser-based ICALL systems developed for German. Chapter 4 gives an introduction to constraint satisfaction problems and chapter 5 describes the formulation of syntactic parsing as a CSP.
with a focus on Extensible Dependency Grammar. Approaches for detecting conflicts in overconstrained CSPs are presented in chapter 6.

Part II presents the EAGLE Corpus, a corpus of written responses from beginning learners of German that was collected and annotated for this thesis. Chapter 7 describes the data and the annotation scheme, which is a linguistically-grounded annotation scheme with a focus on syntactic errors. Chapter 8 provides a detailed analysis of the distribution of errors in EAGLE.

The Fledgling grammar checker is presented in Part III. The grammar developed for the grammar checker is described in chapter 9 and the conflict detection and diagnosis approach that forms the core of the grammar checker is presented in chapter 10. Chapter 11 evaluates the grammar checker using data from the EAGLE Corpus and the thesis is concluded in chapter 12.
Part I

Background
Chapter 2: Modeling German Grammar

This chapter provides an overview of German syntax with an emphasis on the material learned in a typical university-level first-year German course. Both the readers of this dissertation and the learners who contributed to the corpus discussed in chapter 7 are assumed to be fluent speakers of English, so the following sections focus on syntactic phenomena where German and English differ. The first section covers the basics of German syntax and the second section gives a short introduction to the types of errors made by beginning learners of German.

The works consulted for this chapter include: Deutsche Grammatik (Helbig and Buscha, 1991), a grammar reference for teachers of German as a foreign language; Duden: Grammatik der Gegenwartssprache (Duden, 1998), a standard grammar reference for native speakers; and the two-volume Grundriss der deutschen Grammatik (Eisenberg, 1998, 1999), an introduction to German phonology, morphology, and syntax for linguists. In English, one overview of German grammar aimed at intermediate learners is A Practice Grammar of German (Dreyer and Schmitt, 1999).
2.1 A Brief Overview of German Syntax

German is a fusional language characterized by a large degree of syncretism and so-called ‘freer’ word order (Höhle, 1986b; Uszkoreit, 1987). Grammatical features are indicated by suffixes and/or stem internal vowel changes (Umlaut and Ablaut). The following sections address declension and agreement within noun phrases, verb conjugation and subject-verb agreement, agreement in relative clauses, preposition subcategorization, verb subcategorization, and word order.

2.1.1 Declension and Agreement in Noun Phrases

German nouns belong to one of three grammatical genders: masculine, feminine, or neuter. In many cases, the grammatical gender of a noun can be predicted from its meaning or final suffix, but there are numerous exceptions. Learners are typically recommended to memorize gender for each noun by memorizing the nominative singular form of the definite article, which is declined for gender, with each noun. The beginning German textbook used at The Ohio State University, Deutsch: Na klar!, provides the following explanation: “The grammatical gender of a noun that refers to a human being generally matches biological gender; that is, most words for males are masculine, and words for females are feminine. Aside from this though, the grammatical gender of German nouns is largely unpredictable. . . . Since the gender of nouns is generally unpredictable, you should make it a habit to learn the definite article with each noun” (Donato et al., 2008, p. 32–33).

Nouns are declined for case and number. There are four cases (nominative, accusative, dative, genitive) and two numbers (singular, plural). Noun forms display a large degree of
syncretism. Number is clearly marked by suffixes and/or umlaut for most nouns, but only a subset of grammatical cases are indicated unambiguously by endings on the noun itself. There are numerous ways to build the plural in German.

Similar to the situation with noun gender, it is sometimes possible to predict the plural form of a noun based on its gender, meaning, suffix, or etymology, but there are many exceptions. Helbig and Buscha (1991) describe eight common plural patterns while *Deutsch: Na klar!* lists nine patterns, including a null plural, in its introduction to plural forms. It is also frequently recommended that learners memorize plural formation with each noun (cf. Donato et al., 2008, p. 63).

In the singular, most masculine and neuter noun forms are identical with the exception of the genitive, which is marked with an -(e)s ending. A few masculine nouns belong to an alternate declension, where the non-nominative forms are marked with -(e)n. Feminine singular forms are identical for all cases. In the plural, nouns whose plural forms do not end in -n or -s are marked in the dative with the ending -(e)n. To illustrate, Table 2.1 shows the declension of the masculine noun *Mann* ‘man’, which has four distinct forms, the feminine noun *Frau* ‘woman’, which has two distinct forms, and the neuter noun *Messer* ‘knife’, which has a null plural and three distinct forms.

<table>
<thead>
<tr>
<th></th>
<th>Sg</th>
<th>Pl</th>
<th>Sg</th>
<th>Pl</th>
<th>Sg</th>
<th>Pl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nom</td>
<td>Mann</td>
<td>Männer</td>
<td>Frau</td>
<td>Frauen</td>
<td>Messer</td>
<td>Messer</td>
</tr>
<tr>
<td>Acc</td>
<td>Mann</td>
<td>Männer</td>
<td>Frau</td>
<td>Frauen</td>
<td>Messer</td>
<td>Messer</td>
</tr>
<tr>
<td>Daft</td>
<td>Mann</td>
<td>Männern</td>
<td>Frau</td>
<td>Frauen</td>
<td>Messer</td>
<td>Messern</td>
</tr>
<tr>
<td>Gen</td>
<td>Mannes</td>
<td>Männer</td>
<td>Frau</td>
<td>Frauen</td>
<td>Messers</td>
<td>Messer</td>
</tr>
</tbody>
</table>

Table 2.1: Declension of *Mann*
Table 2.1 demonstrates the large degree of syncretism in noun declension. For the eight possible case and number feature combinations for each noun, there are usually only two or three distinct forms. Less syncretism is shown by deadjectival nouns and a small number of nouns referring to people, descriptions, and singular abstract concepts (e.g., *der Neue* ‘the new (one)’, *der Beamte* ‘the civil servant’, *das Interessante* ‘the interesting (thing)’) as they are inflected like adjectives (see the discussion of determiner-adjective-noun inflection at the end of this section). These nouns have up to four distinct forms for the 24 possible determiner-case-number combinations. Additionally, nominative and accusative forms are much more frequent than dative or genitive forms,\(^4\) thus in the majority of instances, the form of a noun on its own indicates number but does not indicate case.

Within a noun phrase that includes a determiner or adjective, the declension of the determiner and adjective often indicate the case information that is absent from the noun declension. Determiners are inflected for gender, case, and number, and they agree with their head nouns for these three features. Adjectives also agree with their head nouns in gender, case, and number, and they additionally agree with the determiner in terms of definiteness. Plural forms of determiners and adjectives are identical for all genders.

Table 2.2 shows the declension of the definite determiner *der/die/das*.

\(^4\)In the Negra German newspaper corpus, there are twice as many nominative subjects as accusative objects and twelve times as many subjects as dative objects (Boyd, 2007).
Table 2.2: Declension of the Definite Determiner

<table>
<thead>
<tr>
<th></th>
<th>Masc Sg</th>
<th>Fem Sg</th>
<th>Neut Sg</th>
<th>Pl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nom</td>
<td>der</td>
<td>die</td>
<td>das</td>
<td>die</td>
</tr>
<tr>
<td>Acc</td>
<td>den</td>
<td>die</td>
<td>das</td>
<td>die</td>
</tr>
<tr>
<td>Dat</td>
<td>dem</td>
<td>der</td>
<td>dem</td>
<td>den</td>
</tr>
<tr>
<td>Gen</td>
<td>des</td>
<td>der</td>
<td>des</td>
<td>der</td>
</tr>
</tbody>
</table>

For the sixteen possible combinations of gender-case-number, there are only six distinct forms of the definite determiner. However if the gender and number of a noun are known, the definite article narrows the case down to one or two possibilities. There are also six distinct forms of the indefinite article ein/eine/ein, as shown in Table 2.3.

Table 2.3: Declension of the Indefinite Determiner

<table>
<thead>
<tr>
<th></th>
<th>Masc Sg</th>
<th>Fem Sg</th>
<th>Neut Sg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nom</td>
<td>ein</td>
<td>eine</td>
<td>ein</td>
</tr>
<tr>
<td>Acc</td>
<td>einen</td>
<td>eine</td>
<td>ein</td>
</tr>
<tr>
<td>Dat</td>
<td>einem</td>
<td>einer</td>
<td>einem</td>
</tr>
<tr>
<td>Gen</td>
<td>eines</td>
<td>einer</td>
<td>eines</td>
</tr>
</tbody>
</table>

As mentioned above, adjectives are declined for gender, case, number, and definiteness. Definiteness is determined by the type of determiner or absence of a determiner. The strong adjective declension is used where there is no determiner, the weak adjective declension is used after a definite determiner (e.g., der ‘the’, dieser ‘this’), and the mixed adjective declension is used after indefinite and possessive determiners (ein ‘a’, sein ‘his’, kein ‘no’). Table 2.4 shows the adjective endings for each declension.

5Given the difficulties in predicting the gender or plural form of a particular noun, it can rarely be assumed that beginning learners of German are certain about the gender or plural form of a noun.
In noun phrases containing a determiner-adjective-noun sequence, the combination of noun gender, noun case ending, determiner, and adjective ending frequently fully specifies the case and number of the noun phrase. There are seven distinct sequences for the eight case-number combinations for *der alte Mann* ‘the old man’, shown in Table 2.5.

<table>
<thead>
<tr>
<th></th>
<th>Masc Sg</th>
<th>Fem Sg</th>
<th>Neut Sg</th>
<th>Pl</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strong</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nom</td>
<td>-e</td>
<td>-e</td>
<td>-e</td>
<td>-en</td>
</tr>
<tr>
<td>Acc</td>
<td>-en</td>
<td>-e</td>
<td>-e</td>
<td>-en</td>
</tr>
<tr>
<td>Dat</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
</tr>
<tr>
<td>Gen</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
</tr>
<tr>
<td><strong>Weak</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nom</td>
<td>-er</td>
<td>-e</td>
<td>-es</td>
<td>-e</td>
</tr>
<tr>
<td>Acc</td>
<td>-en</td>
<td>-e</td>
<td>-es</td>
<td>-e</td>
</tr>
<tr>
<td>Dat</td>
<td>-em</td>
<td>-er</td>
<td>-em</td>
<td>-en</td>
</tr>
<tr>
<td>Gen</td>
<td>-en</td>
<td>-er</td>
<td>-en</td>
<td>-er</td>
</tr>
<tr>
<td><strong>Mixed</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nom</td>
<td>-er</td>
<td>-e</td>
<td>-es</td>
<td>-en</td>
</tr>
<tr>
<td>Acc</td>
<td>-en</td>
<td>-e</td>
<td>-es</td>
<td>-en</td>
</tr>
<tr>
<td>Dat</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
</tr>
<tr>
<td>Gen</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
</tr>
</tbody>
</table>

Table 2.4: Adjective Endings

Table 2.5: Combined Determiner-Adjective-Noun Declension for *der alte Mann*
In contrast, feminine nouns show a greater degree of syncretism with five distinct sequences for eight slots, as shown in Table 2.6.

<table>
<thead>
<tr>
<th></th>
<th>Sg</th>
<th>Pl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nom</td>
<td>die alte Frau</td>
<td>die alten Frauen</td>
</tr>
<tr>
<td>Acc</td>
<td>die alte Frau</td>
<td>die alten Frauen</td>
</tr>
<tr>
<td>Dat</td>
<td>der alten Frau</td>
<td>der alten Frauen</td>
</tr>
<tr>
<td>Gen</td>
<td>der alten Frau</td>
<td>den alten Frauen</td>
</tr>
</tbody>
</table>

Table 2.6: Combined Determiner-Adjective-Noun Declension for *die alte Frau*

### 2.1.2 Verb Conjugation and Subject-Verb Agreement

German verbs are inflected for person, number, tense, voice, and mood. Like English, there are simple present and simple past forms along with complex forms such as present perfect and passive using auxiliary verbs. In addition to 1st/2nd/3rd person forms just like English, German has a 2nd person formal form, which aside from capitalization is identical to the 3rd person plural form and therefore omitted from the tables in this section. Table 2.7 shows the present, past, and present perfect forms for *kaufen* ‘buy’.

<table>
<thead>
<tr>
<th></th>
<th>Present</th>
<th>Past</th>
<th>Present Perfect</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sg</td>
<td>Pl</td>
<td>Sg</td>
</tr>
<tr>
<td>1st</td>
<td>kaufe</td>
<td>kauften</td>
<td>habe gekauft</td>
</tr>
<tr>
<td>2nd</td>
<td>kaufst</td>
<td>kauft</td>
<td>hast gekauft</td>
</tr>
<tr>
<td>3rd</td>
<td>kauft</td>
<td>kaufen</td>
<td>hat gekauft</td>
</tr>
</tbody>
</table>

Table 2.7: Conjugation of *kaufen*
Most verbs form the present perfect and past perfect with the auxiliary verb haben ‘have’ (86% of verbs by type in Buchholz 1996), but some verbs, mainly ones that indicate movement or a change of condition, use the auxiliary sein ‘to be’ instead.\(^6\)

Verb conjugations display less syncretism than noun declensions, but there are still many identical forms. The 1st and 3rd plural forms are identical for all tenses, e.g. kaufen. In the present tense, the 3rd singular and 2nd plural are identical, e.g. kauft. In the past tense for full verbs and in all tenses for modals, the 1st and 3rd singular are also identical. Subjects and verbs agree in person and number as shown in Table 2.8. When verbs appear in context with their subjects, there is no remaining ambiguity about the person or number of the verb.

<table>
<thead>
<tr>
<th></th>
<th>Singular</th>
<th>Plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>ich kaufe</td>
<td>wir kaufen</td>
</tr>
<tr>
<td>2nd</td>
<td>du kaufst</td>
<td>ihr kauft</td>
</tr>
<tr>
<td>3rd</td>
<td>er kauft</td>
<td>sie kaufen</td>
</tr>
</tbody>
</table>

Table 2.8: Subject-Verb Agreement for Present Tense of kaufen

Some verbs derived through prefixation have so-called “separable” prefixes that sometimes appear as a separate unit within the clause. Depending on the type of clause (see section 2.1.6 for more information), the prefix may appear at the end of the clause separate from the main verb stem. For example, the verb ankommen ‘to arrive’ is derived from the verb kommen ‘to come’ and the prefix an- ‘in/at’. In a verb-first or verb-second clause, the

\(^6\)There are traces of a similar distinction remaining in English until quite recently in sentences such as “He is come.” In the Corpus of Historical American English (Davies, to appear), the frequency has been falling since the beginning of the 19th century, but there are instances of “is come” in news articles as late as the 1950s.
prefix appears at the end of the clause as in (2). In a verb-final clause as in (3), the inflected verb appears as a single unit with the prefix

The prefixes are also separated from the verb stem when the past participle is formed *angekommen* ‘arrived’ and in *to*-infinitives *anzukommen* ‘to arrive’.

(2) a. Kommt sie morgen an?
   ‘Will she arrive tomorrow?’
   b. Sie kommt morgen an.
   ‘She will arrive tomorrow.

(3) Er wollte wissen, ob sie morgen ankommt.
   ‘He wants to know whether she will arrive tomorrow.’

2.1.3 Relative Clauses

Relative pronouns agree with their antecedents in gender and number, but not in case, since the case of the relative pronoun reflects its grammatical role in the relative clause. The following examples in (20a) contain an accusative object *den Mann* modified by various relative clauses. In each relative clause, the relative pronoun agrees in gender and number (masc,sg) with the antecedent, but is in a different case in each context as selected by the verb in the relative clause.

(4) a. Ich kenne den Mann, der alt ist.
   ‘I know the man who is old.’
   b. Ich kenne den Mann, den du gestern kennengelernt hast.
   ‘I know the man you met yesterday.’
   c. Ich kenne den Mann, dem du helfen willst.
   ‘I know the man you want to help’
I know the man who you want to help.

d. * Ich kenne den Mann, der du helfen willst.
I know the man who you want to help

2.1.4 Preposition Subcategorization

There are two main types of prepositions: prepositions that select a single case in all contexts and prepositions known as two-way prepositions that select either the dative or accusative depending on context.

Prepositions that select a single case in all contexts can select either the genitive, dative, or accusative, as shown in (5).

(5) a. während des Jahres
during the year
b. bei dem Wetter
in the weather
c. gegen die Wand
against the wall

Two-way prepositions can take either the dative or accusative depending on the context and intended meaning. In general, the dative is used when describing a location and the accusative is used with motion or direction toward the object of the preposition.

(6) a. Sie geht in die Stadt.
she walks in the city
‘She is walking into the city.’
b. Sie geht in der Stadt.
she walks in the city
‘She is walking (around) in the city.’
This choice is often dependent on whether the main verb in the clause expresses location or direction. The examples in (7) show the dative with verbs specifying location (wait, shop) and the accusative with verbs expressing direction (travel, go).

(7) Dative (location):
   a. Der Mann wartet vor dem Bahnhof.  
      the man waits in front of the train station
      ‘The man is waiting in front of the train station.’
   b. Ich kaufe im Supermarkt ein.  
      I shop in the grocery store
      ‘I am shopping in the grocery store.’

(8) Accusative (direction):
   a. Die Frau fährt in ein armes Land.  
      the woman travels in a poor country
      ‘The woman is traveling to a poor country.’
   b. Sie gehen in die Apotheke.  
      they go in the pharmacy
      ‘They are going into the pharmacy.’

There are number of verb pairs similar to the English pair lie/lay, where two similar verbs express either location or direction and correspondingly require dative or accusative with two-way prepositions.

(9) liegen/legen ‘lie/lay’
   a. Die Decke liegt auf dem Bett.  
      the blanket lies on the bed
      ‘The blanket is lying on the bed.’
   b. Sie legt die Decke auf das Bett.  
      she lay the blanket on the bed
      ‘She lays the blanket on the bed.’

(10) stehen/stellen ‘stand/place’
   a. Die Geburts tagskarte steht auf dem Tisch.  
      the birthday card stands on the table
      ‘The birthday card is standing on the table.’
b. Sie stellt die Geburtstagskarte auf den Tisch.
   she stands the birthday card onto the table_{masc,acc,sg}
   ‘She is placing (=is standing up vertically) the birthday card on the table.’

(11) sitzen/setzen ‘sit/set’

a. Sie sitzt auf dem Stuhl.
   she sets on the chair_{masc,dat,sg}
   ‘She is sitting onto the chair.’

b. Sie setzt sich auf den Stuhl.
   she sits herself on the chair_{masc,acc,sg}
   ‘She sits down on the chair.’

The single-case prepositions require no context beyond the preposition itself to determine the correct case for the object whereas the two-way prepositions require knowledge about the main verb and whether the prepositional phrase is an adverbial argument (see section 2.1.5) or an adjunct.

2.1.5 Verb Subcategorization

This section discusses the range of verb subcategorization frames in German. Information about the distribution of verb subcategorization frames included in the sections below comes from Buchholz (1996), a German verb lexicon with detailed subcategorization information for 12,023 verbs with a total of 30,042 subcategorization frames. It will be referred to as BVL for the remainder of this chapter. Subcategorization frame frequencies refer to the percentage of verbs in BVL that can potentially select the argument under consideration.
Subject Selection

The vast majority of verbs select nominative NP subjects. 98% of verbs in BVL select nominative NP subjects. 109 verbs have subcategorization frames that do not include a nominative NP subject and 50 verbs have at least one subcategorization frame that includes an expletive subject instead of a nominative NP subject. Example (12) shows the similarity between expletive subjects in English and in German.

(12) a. Es regnet.
   it rains
   ‘It is raining.’
   b. * Die Wolke regnet.
      the cloud rains

Argument Selection

Verbs can select noun phrase, reflexive pronoun, prepositional phrase, adverbial, and clausal arguments. For NP arguments, verbs can select the case of the NP (accusative, dative, genitive). Verbs whose subcategorization frames include a genitive NP are extremely rare (0.4%). Dative and accusative are both frequent, respectively 12% and 67% of verbs in BVL have subcategorization frames that include such arguments. Examples of each are given in (13).

(13) a. Er kaufte ein Buch.
   he bought a book\textit{neut,acc,sg}
   ‘He bought a book.’
   b. Er hat dem Mann geholfen.
   he has the man\textit{masc,dat,sg} helped.
   ‘He helped the man.’
   c. Wir gedenken der Opfer.
   we commemorate the victims\textit{neut,gen,pl}
‘We commemorate the victims.’

For PP arguments, verbs select the preposition and in the case of two-way prepositions, they also select the case of the preposition’s argument. In BVL, 25% of verbs have at least one subcategorization frame with a PP argument; there are 21 different prepositions used in PP arguments in the subcategorization frames in BVL. The examples in (14) show the same two-way preposition auf with accusative and dative arguments depending on the main verb.

(14) a. Sie wartet auf das Buch.
   she waits on the book\textit{neut,acc,sg}
   ‘She is waiting for the book.’

b. Sie leidet an der Grippe.
   she suffers on the flu\textit{fem,dat,sg}
   ‘She is suffering from the flu.’

34% of verbs have at least one subcategorization frame including a reflexive pronoun. Reflexives can be either accusative or dative as shown in (15). The difference between accusative and dative reflexive pronouns is visible only in the 1st and 2nd person singular.

(15) a. Ich habe mich bei ihm bedankt.
   I have myself\textit{acc} by him thanked
   ‘I thanked him.’

b. Ich habe mir die Telefonnummer gemerkt.
   I have myself\textit{dat} the telephone number noted
   ‘I memorized the telephone number.’

Approximately a quarter of the verbs in BVL include adverbial or clausal arguments. Adverbial arguments can be of several types (manner, duration, place, origin, direction, amount, time) and clausal arguments can specify the type of clause and subordinating conjunction (bare infinitive, to-infinitive, that-clause, if-clause, wh-clause, etc.). Examples
(16) and (17) from Maienborn (1991) show the selection of adverbial and clausal arguments.

(16) a. *Ich wohne.
   I live
b. Ich wohne in Berlin.
   I live in Berlin
   ‘I live in Berlin.’

(17) a. *Ich nehme an.
   I assume
b. *Ich nehme an, ob sie glücklich ist.
   I assume whether she happy is
c. Ich nehme an, dass sie glücklich ist.
   I assume that she happy is
   ‘I assume that she is happy.’

Approximately 30% of the verbs in BVL have a subcategorization frame with two or more arguments. In the most complicated subcategorization frames, which are not in frequent usage or typically learned in introductory courses, a verb may require a reflexive and specify two PP arguments. The most frequent frames in BVL with multiple arguments are accusative NP with clausal argument, accusative NP with dative NP, accusative NP with adverbial, and reflexive with PP.

2.1.6 Word Order

German is a so-called ‘freer’ word order language, with a few strict positional requirements, mainly for verbs, but otherwise fairly flexible phrase order. The strict positional requirements for verbs will be described first followed by the position of arguments and adjuncts. The topological field model, which consisely characterizes the constraints on verb and argument positions, is then introduced along with a short look at discontinuous
constituents in German. The topological field model and discontinuous constituents are not
typically covered in introductory German language courses, but these features of German
have implications for the design of the grammar formalism presented in chapter 5 and the
grammar development in chapter 9.

**Position of Finite and Non-Finite Verbs**

The non-finite verb(s) appear at the end of a clause while the finite verb position varies
based on the clause type. As in English, the finite verb appears clause-initially in a yes/no
question, as seen in (18a). Arguments and modifiers can appear between the finite verb and
the non-finite verb cluster. In unembedded declarative clauses, the finite verb appears in
the second position of the sentence with one constituent before it, as in (18b). In dependent
clauses introduced by a subordinating conjunction, the finite verb generally appears at the
end of the sentence, immediately following any non-finite verbs (18c).

(18) a. **Hat** Anna die Tür wieder zugeschlagen?
    has Anna the door again slammed-shut
    ‘Did Anna slam the door shut again?’

b. Anna **hat** die Tür wieder zugeschlagen.
    Anna has the door again slammed-shut
    ‘Anna slammed the door shut again.’

c. . . dass Anna die Tür wieder zugeschlagen **hat**.
    . . that Anna the door again slammed-shut has
    ‘. . that Anna slammed the door shut again.’

**Position of Arguments and Adjuncts**

While the positions of the verbs in a clause are determined by the clause type, the positions
of arguments and adjuncts are fairly free as long as the verb positions above are respected.
Extending example (18b) in (19), the subject Anna, accusative object die Tür, and adverb modifier wieder can appear in any order as long as one appears before the finite verb and the other two appear between the finite and non-finite verbs. In principle, all six permutations are permitted; syntactic, semantic, and in particular pragmatic discourse factors influence the order.

(19) a. Anna **hat** die Tür **wieder** zugeschlagen.
    Anna has the door again slammed-shut
    ‘Anna slammed the door shut again.’

    b. Die Tür **hat** Anna **wieder** zugeschlagen.
    the door has Anna again slammed-shut
    ‘Anna slammed the door shut again.’

    c. Wieder **hat** Anna die Tür **zugeschlagen**.
    again has Anna the door slammed-shut
    ‘Anna slammed the door shut again.’

Topological Fields

One conceptualization of German sentence structure posits topological fields (see Reis, 1980; Höhle, 1986a) that divide the sentence based on the positions of the verbs and subordinating conjunctions. The fields consist of a left bracket (linke Satzklammer), a right bracket (rechte Satzklammer), and fields before the left bracket (Vorfeld), in between the brackets (Mittelfeld), and after the right bracket (Nachfeld). Figure 2.1 shows the three sentences from (19) divided into topological fields. The final line in Figure 2.1 introduces a new example with an extraposed relative clause in the Nachfeld: *Hat Anna die Tür wieder zugeschlagen, die quietscht?* ‘Did Anna slam the door that squeaks shut again?’
In a question or unembedded declarative clause, the finite verb is the left bracket while in a subordinate clause, the subordinating conjunction is the left bracket and the finite verb appears in the right bracket. In all clauses, the non-finite verb cluster forms the right bracket. Arguments and modifiers can appear in the *Mittelfeld* in all clause types, and certain kinds of clauses and constituents can appear in the *Nachfeld*. In an unembedded declarative clause, one constituent (or partial constituent, as seen below in (20)) can appear right before the finite verb in the *Vorfeld*. By specifying constraints on the elements that can occur in the left/right sentence brackets and the three surrounding fields, the word order in any type of German clause can be concisely characterized using the topological field model.

### Discontinuous Constituents

German also has a number of constructions that contain discontinuous constituents, where a string with one syntactic function is not realized as a continuous unit in the sentence. A few examples of these include extraposed clauses (20a), placeholder elements (20b), and fronted partial constituents (20c).

(20) a. Anna hat **die Tür** zugeschlagen, **die ich neulich grün gestrichen habe.**

Anna has the door slammed-shut that I recently green painted

‘Anna slammed shut the door that I recently painted green.’
b. Das hat mich daran erinnert, daß wir morgen Farbe kaufen sollten.
that has me of it reminded that we tomorrow paint to buy should
‘That reminded me that we should buy paint tomorrow.’

c. Von diesen Türen habe ich nur zwei gestrichen.
of these doors have I only two painted
‘I have only painted two of these doors.’

In (20a), a relative or other clause that modifies an element before the non-finite verb cluster is extraposed after the non-finite verb cluster, appearing in the Nachfeld. In (20b), a pronominal adverbial appears in the position of a required PP argument of the verb and refers to an extraposed clause. In (20c), a PP modifying the accusative object appears in the position right before the finite verb (Vorfeld) while the head of the accusative object appears between the finite and non-finite verbs (Mittelfeld).

Clauses are extraposed much more frequently in German than in English. Gamon et al. (2002) found that up to 35% of relative clauses are extraposed in German in technical manuals and encyclopedia texts whereas less than 1% of similar English clauses are extraposed in the same genres.

2.2 Types of Errors

The types of grammatical errors made by the beginning learners of German in the EAGLE corpus, which will be discussed in detail in chapters 7–8, lie not unexpectedly in the areas where German differs from English. Some of the most frequent grammatical errors are: incorrect case for the NP complement of a verb, determiner-adjective-noun agreement, subject-verb agreement, incorrect case for the NP complement of a preposition, and incorrect word order for the finite verb in a main clause. A detailed error analysis will be
presented in chapter 8. In contrast, word order within noun phrases and within prepositional phrases is very similar to English so there are comparatively few errors in this area.

2.2.1 Motivation for Error Typology

The error typology developed for beginning learners of German, which is presented in chapter 7, is motivated by the fundamentals of German grammar presented in this chapter. The top-level categories in the error typology link to the central types of constraints found in the grammar: agreement, selection, and word order. The individual error types will correspond to many of the syntactic phenomena presented in section 2.1, which has focused on the facets of German syntax which are very different from English or particularly challenging for learners coming from English-speaking backgrounds.

2.3 Summary

Some of the syntactic phenomena in German that differ quite noticeably from English include a much greater degree of inflectional morphology (grammatical gender, adjective inflection, etc.), which leads to more complicated agreement and selection along with numerous differences in word order at the clause level. The most frequent types of errors made by English-speaking learners of German, especially beginning learners, fall into these categories.
To provide detailed feedback to learners, an intelligent computer-aided language learning (ICALL) system needs to be able to provide analyses of both grammatical and ungrammatical sentences. This chapter reviews the existing approaches to grammatical error detection and diagnosis in ICALL and provides a historical overview of the research on parser-based ICALL for German.

3.1 Approaches to Error Detection and Diagnosis in ICALL

The development of error detection and diagnosis components for use in ICALL is driven by two conflicting needs (Menzel and Schröder, 1998): a high-precision parser that distinguishes grammatical from ungrammatical sentences and a robust parser that allows for a (partial) analysis of an ungrammatical sentence in order to diagnose the location and type of any errors. Existing off-the-shelf parsers typically fulfill only one of these two requirements, so it is necessary to modify the grammar, the parsing algorithm, or both in order to develop a parser that is adequate for use in a language learning context.

Using a demonstration grammar introduced in section 3.1.1, this section will discuss the existing approaches to error diagnosis: mal-rules (section 3.1.2), parser algorithm modification (section 3.1.3), meta-rules (section 3.1.4) and constraint relaxation (section 3.1.5).
The mal-rule and parser modification approaches typically take as the starting point a grammar with a context-free backbone (CFG). All sentences generated by the CFG are grammatical sentences and any it cannot generate are deemed ungrammatical. A parser for a CFG provides one or more parses for all grammatical sentences and fails to parse any other input. The fact that the parser fails can be used to identify ungrammatical sentences and thereby detect an error, but the parser does not provide any analysis of ungrammatical sentences, so it is not possible to derive detailed information about the location or type of error for the learner.

To allow the parser to also provide an analysis of ungrammatical sentences and facilitate error diagnosis, the grammar or the parser must be modified so that parsing continues despite the fact that the original CFG does not generate an ungrammatical sentence. The mal-rule approach, described in section 3.1.2, modifies the grammar whereas the approaches in section 3.1.3 modify the parsing algorithm to handle particular types of errors. The meta-rule approach described in section 3.1.4 augments the parser with external routines that handle specific types of errors.

### 3.1.1 Demonstration Grammar

A very simple context-free grammar, shown in Figure 3.1, will be used to demonstrate several approaches in the following sections. Feature unification and percolation are spelled out explicitly so a simple context-free parser can be used. The following sentences (among others) are generated by this grammar. Parses of examples (21a) and (21d) are shown in Figures 3.2 and 3.3 with features omitted for brevity.

(21) a. der Mann sieht das Buch (‘the man sees the book’)
**Rules**

S → NP[1,nom,sg] VP[1,sg]  
S → NP[3,nom,sg] VP[3,sg]  

**Lexicon**

NP[1,nom,sg] → ich  
Det[3,nom,masc,sg] → der  
Det[3,nom,masc,sg] → den  
Det[3,nom,neut,sg] → das  
Det[3,nom,neut,sg] → das  
Noun[3,nom,masc,sg] → Mann  
Noun[3,acc,masc,sg] → Mann  
Noun[3,nom,neut,sg] → Buch  
Noun[3,acc,neut,sg] → Buch  
Verb[1,sg] → sehe  
Verb[3,sg] → sieht

Figure 3.1: Demonstration Context-Free Grammar (CFG)
Some sentences that cannot be generated by this grammar are shown below. These examples are crafted to show specific types of errors in sentences similar to the grammatical sentences. In addition to these ungrammatical sentences, there are also several grammatical sentences that cannot be generated by this grammar such as das Buch sieht der Mann, the object-verb-subject ordering of ‘the man sees the book’. To parse these sentences, additional rules would need to be added to the grammar.

(22) a. * ich sieht den Mann (*subject-verb agreement error*)
    b. * der Mann das Buch sieht (*word order error*)
    c. * der Mann sieht (*missing NP object*)
d. * ich sehe der Mann (*incorrect case in object*)
e. * ich sehe Mann (*missing determiner*)

An equivalent constraint grammar will be presented in section 3.1.5.

3.1.2 Mal-Rules

Given a grammar of well-formed rules as in Figure 3.1, which can be used to generate
the sentences in (21), one approach to analyzing ill-formed sentences as in (22) is to add
specially marked rules called *mal-rules* to the grammar that describe expected errors. The
parser has access to both the well-formed rules and mal-rules when parsing a sentence.
In order to prevent spurious analyses, some approaches such as the meta-rule approach
(Weischedel and Sondheimer, 1983) and the ICICLE system (Schneider and McCoy, 1998;
Michaud and McCoy, 2001) will first attempt to parse the sentence using only well-formed
rules and will only include mal-rules in the grammar if the initial parse fails. Others use
mal-rules in tandem with well-formed rules throughout.

If the sentence is parsed successfully using only the well-formed rules, the sentence is
correct. If the sentence is parsed using a combination of the well-formed rules and mal-
rules, the sentence contains an error. The node in the tree where the mal-rule was applied
is used to identify the location of the error. If the mal-rule addresses a specific type of
local error, the mal-rule itself can be used to classify the error. If more than one mal-rule
is involved in a parse containing a non-local error, it is not as straight-forward to associate
the presence of a mal-rule in the parse with a specific error type.
Mal-rules that could be added to the demonstration CFG are shown in Figure 3.4. A sample mal-rule is provided for each error type in the left-hand column. The first rule allows a 1st-person singular subject to combine with a 3rd-person singular verb. The second rule allows the object to precede the verb. In this example, the mal-rules are marked with * so that the location of the error can be identified in the tree. Parses for the ill-formed sentences (22a) and (22b) are shown in Figures 3.5 and 3.6.

Mal-rules can be used to handle a wide variety of errors including agreement, selection, word order, missing word, and extra word errors. Example mal-rules for these types of errors are shown in Figure 3.4. The feedback to the learner can include information about the location of the error using the location of the marked (*) node in the tree and the particular mal-rule gives information about the type of error. For (22a), the feedback would be that there is a subject-verb agreement error in *ich sieht* and for (22b), a word order error in *das Buch sieht*.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Sample Mal-Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Agreement</td>
<td>S* → NP[1,sg] VP[3,sg]</td>
</tr>
<tr>
<td>2. Word order</td>
<td>S* → NP Verb</td>
</tr>
<tr>
<td>3. Selection</td>
<td>S* → Verb NP[nom]</td>
</tr>
<tr>
<td>4. Missing word</td>
<td>NP* → Noun</td>
</tr>
<tr>
<td>5. Extra word</td>
<td>VP* → Verb Verb NP</td>
</tr>
</tbody>
</table>

Figure 3.4: Sample Error Types with Associated Mal-Rules
Mal-rules may be added to the lexicon as well as to the grammar. The German Tutor system described in Heift (1998) uses mal-rules in the lexicon to handle agreement errors. A subset of the lexicon from Figure 3.1 is shown with additional mal-rules that handle subject verb agreement in Figure 3.7. A parse for *ich sieht den Mann (22a) is shown Figure 3.8. It is nearly identical to Figure 3.6 except that the marked node is now on the NP subject ich instead of the S root node. The location of the mark indicates that the error was a number agreement error in the subject NP. By augmenting the lexicon with mal-rules for all possible incorrect feature values, the German Tutor system can handle agreement errors without modifying the parsing algorithm.

The ICICLE system (Schneider and McCoy, 1998; Michaud and McCoy, 2001) is a system for native American Sign Language (ASL) speakers writing English. ASL is structurally

---

7Heift (1998) stores lexical entries as feature structures so the appearance of her lexicon is quite different from the simplified example presented here, however the approach is the same.
very different from English, so ASL speakers make the same kinds of errors in written English as other non-native speakers. For example, ASL native speakers often omit the copula *be* producing sentences such as *The boy happy* (examples from Schneider and McCoy, 1998). A mal-rule such as VP* → AdjP handles this kind of error. Another common error is deleting topic NPs as in *Is happy*. A second mal-rule (S* → VP) is introduced to handle this case.

Multiple mal-rules, such as the two introduced for the ICICLE system in the previous paragraph, can interact and lead to many spurious parses, especially in the case of mal-rules handling omissions. For example, the adjective *Happy* could be parsed as a complete sentence using both mal-rules above, so the parser considers this possibility for any adjective even if it is part of a well-formed sentence. When many different types of omission are possible, the number of spurious parses increases dramatically, so the system needs to limit the application of mal-rules for the system to be efficient. An exception to this is mal-rules
in the lexicon as mentioned above: because mal-rules in the lexicon are all unary rules with terminal symbols on the right-hand side, they do not have the potential to interact like mal-rules in the grammar.

McCoy et al. (1996) developed a learner model named SLALOM (Steps of Language Acquisition in a Layered Organization Model) that includes a model of language transfer errors that are likely based on their learners’ ASL background and a model of second language acquisition. SLALOM uses information about errors that are likely at particular stages of acquisition to determine activate certain mal-rules are present in the grammar. Restricting the presence of mal-rules to errors that the learner is likely to make at a particular stage increases the probability that the system provides the appropriate analysis of the learner’s utterance and improves the efficiency of the parser.

Schwind (1988) presents an ICALL system for learners of German that uses mal-rules to handle word order errors of native French speakers, including placing the adjective after the noun instead of before it (NP* → Det Noun Adj) and for placing the past participle next to the finite verb instead of at the end of the sentence (VP* → Verb VerbPP NP). She notes that these kinds of syntactic errors will need to be anticipated, so the treatment is not very general. Bender et al. (2004) argues that despite the fact that a mal-rules approach will never cover all cases of ill-formed input, they can provide a precise diagnosis for common errors. Even though the system may not be able to diagnose certain kinds of errors, providing a diagnosis and feedback for common types of errors is still beneficial to learners.

For the ICICLE system, Schneider and McCoy (1998) use knowledge about the target users to constrain the system. They consider limiting the system to using only one mal-rule in
each parse, but this prevents them from handling sentences with multiple errors, which are common for their users. Instead, they notice that learners rarely omit more than one item (except for determiners) in a sentence, so a special feature is percolated when an omission mal-rule is applied that prevents other omission mal-rules from appearing in the same tree.

### 3.1.3 Parser Modification

Instead of modifying the grammar, an alternative method is to modify the parsing algorithm. There are two aspects of common parsing algorithms that have been modified for use in parser-based CALL: the feature unification algorithm for parsers using unification (Schieber, 1986; Sag et al., 1986) and the chart parsing algorithm for chart parsers (Earley, 1970; Cocke and Schwartz, 1970; Kasami, 1965; Younger, 1967). An introduction to chart parsing algorithms is provided in the textbook *Speech and Language Processing* (Jurafsky and Martin, 2008).

#### Unification Modification

In the previous section on mal-rules, the features present on the phrasal categories in the demonstration CFG did not need to be analyzed as units separate from the phrasal categories because feature unification was handled by the rules directly (e.g., $S \rightarrow NP[1,\text{nom,sg}]$ $VP[1,\text{sg}]$; $S \rightarrow NP[3,\text{nom,sg}]$ $VP[3,\text{sg}]$). The demonstration grammar is simple enough that directly specifying the feature unification by hand is possible, however for a large number of features and lexical categories, it quickly becomes unwieldy. The grammar is more compact and better captures the generalizations of the language if it is possible to specify a single rule for $S \rightarrow NP$ $VP$ and require that the person and number features of the NP and
the VP agree: $S \rightarrow \text{NP[PER,\text{nom},\text{NUM}]} \text{ VP[PER,\text{NUM}]}$. The parser checks that both PER and NUM agree before applying the $S \rightarrow \text{NP VP}$ rule.

In the case of well-formed input, the features agree as required and the parser can provide a parse. In the case of input containing errors, an unmodified parser will be unable to apply the $S \rightarrow \text{NP VP}$ for a sentence such as *ich sieht den Mann* (22a) because the subject and verb do not agree in person. To handle these cases, the parser is modified so that it can continue despite clashing feature values. This modification is a limited type of constraint relaxation, which will be addressed more generally in section 3.1.5.

An example of modified feature unification is shown in Figure 3.9. The parser makes a note at the point in the tree where it proceeded despite clashing features. Here, an agreement error is recorded between the subject and the verb.
In many cases with agreement errors, the nature of the error is ambiguous. When two words do not agree, it is often difficult to determine what the learner intended to write.\(^8\) For example, in (22a), the learner could have chosen the incorrect subject pronoun (\textit{ich} ‘I’ instead of \textit{er} ‘he’) or the learner could have conjugated the verb incorrectly (\textit{sieht} ‘sees’ instead of \textit{sehe} ‘see’). Because the phrase with the agreement error may be embedded in a larger clause, it may be necessary to consider both possibilities in order to find the best partial analysis of the ungrammatical sentence. The issue of ambiguous errors will be discussed further in section 7.3.3 in the development of the EAGLE error annotation scheme.

To handle ambiguous cases resulting from agreement errors, Reuer (2003a, 2005) describes a general modified feature unification algorithm that preserves non-clashing features and considers all possibilities for clashing features while recording the alternate values as errors for use in diagnosis:

\[
\left[ F_1 : a \right] \cup \left[ F_1 : b, F_2 : c \right] = \left\{ \left[ F_1 : a, F_2 : c \right], \left[ F_1 : b, F_2 : c, \text{err} : [F_1 : a] \right] \right\}
\]

When the values for feature F1 clash (\textit{a} in the left AVM and \textit{b} in the right), the parser creates two alternatives, one where \textit{a} is the value chosen for feature F1 and the value \textit{b} is recorded as an error in a special error attribute (\textit{err} : [\textit{F1} : \textit{b}]) and one where \textit{b} is the value chosen and \textit{a} is recorded as the error.

The modified feature unification approach can be used for agreement errors as in the example above and also for a subset of selection errors, for example when the case of a verb...
argument is incorrect in a rule such as $S \rightarrow V \text{ NP}[\text{ACC}]$. If a verb selects an accusative object and the only available argument is dative, then the modified algorithm could allow the dative argument to be parsed as the argument of the verb even though there is a clash in the case feature. Selection errors that involve more than a case mismatch, such as an NP argument in the place of a PP argument, could not be handled by a modified feature unification algorithm.

**Parse Algorithm Modification**

Mellish (1989), Kato (1994), and Reuer (2005) have proposed modified chart parsing algorithms for handling syntactically ill-formed input. Mellish (1989) and Kato (1994) focus on unknown/misspelled words, missing words, and extra words. They initially run a bottom-up chart parser on the input. If no parse is found, a generalized top-down parser uses the partial analyses in the chart to predict how the input could be modified to generate a complete parse. The top-down parser has access to global information about the possible structure of the sentence and can recognize substitutions, deletions, and insertions by examining the edges in the chart.

Reuer (2005) extends the Earley chart parsing algorithm (Earley, 1970) to allow for deletions, insertions, and word order errors. Given the state of the chart, the current word, the next word, and the context-free grammar, the parsing algorithm modifies the chart so that the parser can proceed. No additional top-down parser is needed to handle errors. The parser proceeds as normal unless it reaches a state where there is no possible way to connect the next word to any hypothesis in the chart. It then introduces an edge into the chart that allows the parser to continue.
In the case of a missing word, a 0-length token is inserted. The category of the inserted
token depends on the grammar. Only tokens that could possibly appear between the current
word and the next word are considered, which limits the parser’s search space. In the case
of an extra word, the current word’s edge is extended over the extra word and the extra
word is ignored.

An example with a missing word (22e) is shown in Figure 3.10. When the parser finds no
way to incorporate Mann into the current chart, it considers the possible tokens that could
appear between sehe and Mann given the rules in the grammar. Using the rule NP → Det
Noun, it can predict that one possibility is a missing determiner and add the Det edge to
the chart. The location of the error is given by the state of the parser when it failed, and the
type of error is given by the modifications the parser had to make so that it could continue.

Word order errors pose more difficulties because an entire phrase consisting of multiple
words may need to be moved. The parser needs some mechanism to identify how many
words to consider moving. In Reuer (2003a), the modified algorithm finds the longest
possible phrase at the position where an error was detected and continues parsing to see if
the phrase can be inserted later in the sentence.

The chart parser algorithm modification approach has the advantage over the mal-rule ap-
proach that there is no need to specify mal-rules for each kind of insertion, deletion, or
permutation and that the algorithm is independent from the particular context-free gram-
mar used. However, the parser may propose many insertions, deletions, and permutations
that the learners are unlikely to make, potentially generating many spurious analyses. In
order to restrict the parser’s search space, information from a learner model about the types
of errors a learner is likely to make is needed.
3.1.4 Meta-Rules

Meta-rules were introduced in Weischedel and Sondheimer (1983), which describes an implementation for an augmented transition network parser (Woods, 1970). The parser initially tries to parse the sentence using well-formed rules only. If no parse is found, the parser then searches for a meta-rule that is applicable to the current state, applies the meta-rule to relax the constraints that are blocking further parsing, and continues with the well-formed rules from that point.

As defined in Weischedel and Sondheimer (1983), meta-rules have a very general form:

\[ C_1 C_2 \ldots C_n \rightarrow A_1 A_2 \ldots A_m \]
The constraints $C_1 \ldots C_n$ on the left-hand side are constraints on the current state of the parser that determine whether the meta-rule can apply. The actions $A_1 \ldots A_m$ make changes to the input and the current state so that the parser can continue if possible. Alternatively, meta-rules can call a halt to further processing if there is no way to recover from the error. The parse algorithm modifications in section 3.1.3 could be viewed as a kind of meta-rule that modifies the chart so that the chart parsing algorithm can then continue.

Considering the same missing word error (22e), a meta-rule can be constructed that inserts a determiner so that the parser can continue. The ungrammatical example (22e) can be parsed by applying a meta-rule at the point where the parser fails using the original CFG from Figure 3.1.

**Meta-rule 1**: If the current node is an NP and the next word is a noun, insert a null determiner and proceed.

Meta-rules are stated very generally, so they are not specific to context-free grammar or syntactic errors and they are very powerful because they can make any sort of changes to the current state of the parser. The meta-rules described in Weischedel and Sondheimer (1983) handle spelling errors, commonly confused words (e.g., *good/well*), and semantic errors such as semantic class restrictions (e.g., relaxing the animate requirement for the subject of *drink*). They propose treatments for many types of errors that can be identified at a single stage of parsing, but they acknowledge that they do not have a general way of dealing with word order errors. Meta-rules are also dependent on the grammar because the kinds of actions that need to be performed can depend on the rules present in the grammar.
3.1.5 Constraint Relaxation

Approaches that use constraint-based grammars offer an alternative to the previous approaches that rely on specialized modifications to context-free grammars and parsers. Constraint satisfaction problems are used for many kinds of tasks: scheduling, resource allocation, etc. Constraint satisfaction problems and constraint solving algorithms will be introduced in detail in chapter 4.

A constraint-based grammar consists of a set of lexical items and a set of constraints that grammatical sentences need to satisfy. A sentence is considered grammatical if there exists an analysis that satisfies all constraints. If there is no way to satisfy all constraints, the sentence is ungrammatical. An example grammar for demonstration purposes is shown in Figure 3.12, which is weakly equivalent to the context-free grammar from Figure 3.1.
**Constraints**

Selection Constraints:
1) A word must satisfy its selection requirements.

Agreement Constraints:
2) A Verb and its subject (Noun[nom]) agree in number and person.
3) A Noun and its determiner (Det) agree in gender, number, and case.

Sentence Form Constraints:
4) A sentence contains exactly one Verb.
5) Except for the Verb, all words in the sentence must be selected.

Ordering Constraints:
6) Noun[nom] ≺ Verb
7) Verb ≺ Noun[acc]
8) Det[nom] ≺ Noun[nom]
9) Det[acc] ≺ Noun[acc]

**Lexicon**

<table>
<thead>
<tr>
<th>Word</th>
<th>Category</th>
<th>Features</th>
<th>Selects</th>
</tr>
</thead>
<tbody>
<tr>
<td>ich</td>
<td>Noun</td>
<td>(1,nom,masc,sg)</td>
<td>-</td>
</tr>
<tr>
<td>der</td>
<td>Det</td>
<td>(3,nom,masc,sg)</td>
<td>-</td>
</tr>
<tr>
<td>den</td>
<td>Det</td>
<td>(3,acc,masc,sg)</td>
<td>-</td>
</tr>
<tr>
<td>das</td>
<td>Det</td>
<td>(3,nom,neut,sg)</td>
<td>-</td>
</tr>
<tr>
<td>das</td>
<td>Det</td>
<td>(3,acc,neut,sg)</td>
<td>-</td>
</tr>
<tr>
<td>Mann</td>
<td>Noun</td>
<td>(3,nom,masc,sg)</td>
<td>Det</td>
</tr>
<tr>
<td>Mann</td>
<td>Noun</td>
<td>(3,acc,masc,sg)</td>
<td>Det</td>
</tr>
<tr>
<td>Buch</td>
<td>Noun</td>
<td>(3,nom,neut,sg)</td>
<td>Det</td>
</tr>
<tr>
<td>Buch</td>
<td>Noun</td>
<td>(3,acc,masc,sg)</td>
<td>Det</td>
</tr>
<tr>
<td>sehe</td>
<td>Verb</td>
<td>(1,sg)</td>
<td>Noun[nom], Noun[acc]</td>
</tr>
<tr>
<td>sehe</td>
<td>Verb</td>
<td>(3,sg)</td>
<td>Noun[nom], Noun[acc]</td>
</tr>
</tbody>
</table>

Figure 3.12: Demonstration Constraint-Based Grammar
In the case of an ungrammatical sentence such as *ich sieht den Mann (22a), no analysis can be found that satisfies all constraints, specifically the agreement constraint. In order to provide an analysis, it is necessary to relax one or more constraints. In this case, an analysis, shown in Figure 3.13, can be found if the subject-verb agreement constraint is relaxed. The type of error is indicated by the type of constraint that was relaxed, and the location is indicated by where the constraint failed to hold.

Menzel and Schröder (1998) treat parsing as a constraint satisfaction problem in the framework of Constraint Dependency Grammar (Maruyama, 1990a). Enforcing all constraints would lead to a parse failure in the case of ill-formed input, so Menzel and Schröder give each constraint a weight between 0 and 1 that shows how much the violation of that constraint would affect the acceptability of the sentence. The best parse is the parse that violates the fewest and weakest constraints.
### Table 3.1: Overview of Parser-Based ICALL Approaches for German

<table>
<thead>
<tr>
<th>System</th>
<th>Mal-Rules</th>
<th>Parser Modification</th>
<th>Constraint Relaxation</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Die Sprachmaschine</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Spion/Syncheck</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Herr Kommissar</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Schwind (1995)</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRIDGE</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Menzel and Schröder (1998)</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>German/E-Tutor</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Textana</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>DiBEx</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Fortmann and Forst (2004)</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PromisD</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

This section provides an overview of previous research on parser-based ICALL specifically for learners of German. The development of the earliest tutoring systems for German began in the 1980s and has continued since. Table 3.1 lists the systems discussed in the section in roughly chronological order and summarizes the types of approaches used. For German and other languages, Heift and Schulze (2007) provide an excellent general survey of research in natural language processing in CALL.

### 3.2 Parser-Based ICALL for German

This section provides an overview of previous research on parser-based ICALL specifically for learners of German. The development of the earliest tutoring systems for German began in the 1980s and has continued since. Table 3.1 lists the systems discussed in the section in roughly chronological order and summarizes the types of approaches used. For German and other languages, Heift and Schulze (2007) provide an excellent general survey of research in natural language processing in CALL.

#### 3.2.1 Die Sprachmaschine: Harroff (1986)

The ICALL system *Die Sprachmaschine* (Harroff, 1986) (‘The Language Machine’) uses simple sentence patterns to restrict learner input. The learners build sentences by selecting
base forms from the lexicon. The system adds determiners and conjugates verbs as necessary to form a grammatical sentence. It can handle sentences in the following formats:

- subject + verb + object (if needed) + time word (opt.) + prepositional object
- time word + verb + subject + object (if needed) + prepositional object

The system provides feedback to the students if the word order is incorrect or if the sentence does not follow lexical semantical constraints. The lexicon includes 65 nouns, 18 verbs, 9 prepositions, and the adverbs heute ‘today’ and jetzt ‘now’. Learners can also ask the system direct questions about German grammar such as: ‘What is the accusative of noun?’

### 3.2.2 Spion and Syncheck: Sanders and Sanders (1987); Sanders (1991); Sanders and Sanders (1995)

*Spion* (‘Spy’), whose development began in 1981, is an adventure game aimed at college-level German learners. The plot involves spies in East and West Berlin, who the German learners need to interrogate, move around to different locations within the city, and instruct to perform particular actions. The system uses a semantic network and a simple syntactic parser with a phrase structure-based grammar limited to interrogative and imperative sentences.

Research on the parsing component of *Spion* led to the development of a German parser called *Syncheck* for intermediate learners of German at Miami University (Sanders and Sanders, 1987; Sanders, 1991). *Syncheck* uses a Definite Clause Grammar written in Prolog with a lexicon size of approximately 9,000. *Syncheck*, which was intended to be used as
a grammar checker, ran into the exact problem mentioned at the beginning of this chapter for high-precision parsers: the parser can distinguish grammatical from ungrammatical sentences only by checking whether it is possible to parse the input using the grammar. When the sentence is ungrammatical, the parser fails, which means that the learners can be given useful feedback about the grammaticality of input sentences. Because the parser is not able to provide a partial syntactic analysis of ungrammatical sentences, more detailed feedback cannot be provided.

3.2.3 Herr Kommissar: DeSmedt (1995)

Herr Kommissar (‘Mr. Inspector’) is a “role-playing detective game” where the German learners play a police inspector and need to interrogate suspects in order to solve a murder. Herr Kommissar can detect spelling errors and uses a parser to find many types of grammatical errors: noun declension, adjective-noun agreement, pronominalization of noun phrases, preposition case selection including two-way prepositions, verb complement case selection, subject-verb agreement, verb conjugation including the future and perfect tenses and passive voice, verb constructions using modal auxiliaries, and coordinate, subordinate, and relative clauses.

The Herr Kommissar system uses a parsing technique DeSmedt calls “prediction-driven parsing” to analyze the learner productions. This method first identifies the main verb and then tries to sort the remainder of the sentence into the elements in the verb’s subcategorization frame. If there’s no element that fulfills the requirements for a particular complement, then the parser searches for an as-of-yet unattached phrase that comes closest to fulfilling the complement’s requirements. “[L]east-distance heuristics are the key to Herr
Kommissar’s ability to cope successfully with” the errors in surface forms such as articles and prepositions that often cause difficulties for parsers using context-free grammar rules. DeSmedt (1995) reports that the parser detects and corrects 92.8% of all errors with a false positive rate of 1.1% in beta tests with students at Carnegie Mellon University and at the University of Maryland’s Language House.

3.2.4 Schwind (1995)

Schwind (1995) extends a unification algorithm in order to record information about the features that cannot be unified in order to provide an error analysis. Using this approach, her system can identify agreement errors, syntactic selection errors including missing or extra arguments, word order errors, and lexical selection errors involving constraints on verbs and their complements. Agreement and lexical selection errors do not need to be anticipated, because they are handled by the modified unification algorithm. The syntactic selection errors are anticipated using mal-rules that handle each specific error such as a missing determiner in a noun phrase or reordered constituents. The exercises in Schwind’s system that allow freer word choice are restricted to specific simple sentence forms such as subject+verb+complements, where the complements may include subordinate clauses such as that-clauses. Her system is implemented in PROLOG and has a vocabulary of about 600 verbs, 500 nouns, and 60 adjectives.

When there are multiple potential error diagnoses due to the ambiguity inherent in agreement errors (see section 3.1.3), Schwind (1995) uses information about expected case from the context to decide which error to report. For example, the phrase *der Kind ‘the child’
is not a possible noun phrase given the possible gender, case, and number features of the two words:

- *der*: masc, nom, sg; fem, dat, sg; fem, gen, sg; gen, pl

- *Kind*: neut, nom, sg; neut, acc, sg; neut, dat, sg

The following three examples show instances where the expected case is determined by the context. The complement *Kind* is expected to be nominative (23a), dative (23b), or accusative (23c). In the first example, since both *der* and *Kind* can be nominative, the error reported by the system is that the determiner does not agree in gender with *Kind*. The second example is similar: both the determiner and the noun can be dative, so the error is that the determiner does not agree in gender. In the third example, *der* cannot possibly be accusative, so the error is a case error related to *der* and the fact that the verb expects an accusative complement.

(23)  

a. Der Kind spielt.  
   the child plays  
   ‘The child is playing.’

b. Er gibt der Kind Milch.  
   he gives the child milk  
   ‘He gives the child milk.’

c. Sie kennt der Kind.  
   she knows the child  
   ‘She knows the child.’

As Heift and Schulze (2007) note, this approach assumes that the students are more likely to know the correct grammatical case for a particular complement of a verb rather than the grammatical gender of a noun. Depending on the student’s background and strengths, this may not be the case.
3.2.5 BRIDGE: Weinberg et al. (1995); Sams (1995)

Weinberg et al. (1995) and Sams (1995) describe the tutoring system BRIDGE that was developed by the United States Army Research Institute for intermediate-level learners of several languages including German. Their parser, which uses the government and binding framework (Chomsky, 1982), is written in PROLOG and records errors when features disagree. The German lexicon includes approximately 5,000 words.

BRIDGE can detect errors for German in subject-verb agreement, subject case selection, determiner-noun agreement, preposition case selection, verb complement selection including noun phrases and prepositional phrases, word order errors, and auxiliary selection in perfect tenses (haben vs. sein). Selection and agreement errors are detected by checking for feature mismatches while parsing. There is a separate mechanism that checks for two specific word order errors frequently produced by English speakers learning German: whether the finite verb appears at the end of a subordinate clause and whether the finite verb is placed in the second position following a sentence-initial adverbial.

The parser also uses flags that determine whether certain error types should be recorded while parsing, so a teacher can configure an activity to focus on particular error types rather than reporting all errors. If a flag for subject-verb agreement is not set, for example, the parser will not record any subject-verb agreement errors in the set of errors which it sends to the feedback module.

The BRIDGE parser was evaluated using approximately 200 correct and incorrect German sentences constructed by the authors to demonstrate likely syntactic constructions and errors produced by early intermediate learners (Weinberg et al., 1995). The parser provided
the correct parse for 135 out of 141 (96.4%) correct sentences and provided the appropriate error message for 61 out of 64 (95.3%) incorrect sentences.

3.2.6 Menzel and Schröder (1998, 1999)

Menzel and Schröder (1998, 1999) propose a constraint-based approach that uses weighted constraints and multiple levels of linguistic knowledge for error diagnosis. Their approach is based on *Constraint Dependency Grammar* (Maruyama, 1990b), the first formulation of dependency parsing as a constraint satisfaction problem. A constraint dependency grammar (CDG) consists of a finite set of words, a finite set of roles (such as *governor*), a set of labels (such as *det* or *subj*), and a set of constraints that any parse should satisfy. The parser creates an initial dependency representation (Maruyama: *constraint network*) where every word stands in every possible relation to every other word in the sentence. For example, the initial representation for the sentence *Ich sehe den Mann* ‘I see the man’ given a grammar with one role/label combination *head* would look like the diagram in Figure 3.14. Each additional label in the grammar would introduce a pair of arcs between each pair of words in the sentence.

The parser then takes the provided constraints into account and uses constraint propagation (see section 4.2.4) to eliminate arcs until it finds one or more parses that satisfy all constraints. When a parse is found for Figure 3.14, it could look like the parse in Figure 3.15. CDG is discussed in further detail in chapter 5.

CDG treats parsing as a constraint satisfaction problem (CSP), i.e., given the initial dependency representation and a set of constraints that eliminate relations from the initial representation, is it possible to find a parse? When a CSP has no solution (i.e., the input
Figure 3.14: Initial Dependency Representation for Eliminative Parsing

Figure 3.15: One Solution for Figure 3.14
is ungrammatical and there is no parse that satisfies all constraints), constraint propagation will prove that there is no solution, but it is unable to provide a partial analysis or an explanation for the reason for the parse failure, which an ICALL system needs in order to provide detailed feedback.

In order to make it possible to find partial analyses and perform error diagnosis, Menzel and Schröder (1998, 1999) reformulate the original CDG CSP as a partial constraint satisfaction problem with weighted constraints. Unlike a CSP, a partial constraint satisfaction problem (PCSP) allows solutions where some constraints are violated. In order to rank solutions to a PCSP, Menzel and Schröder assign weights to each constraint that indicate the seriousness of the violation of that constraint. They assign weights from 0.0–1.0, where a weight of 0.0 is given to a hard constraint that cannot be violated and a weight close to 1.0 is given to constraints that indicate preferences rather than errors. They define the best solution to the PCSP as the one where the product of the weights of the violated constraints is the closest to 1.0.

Menzel and Schröder (1998) describe a prototype that was used to evaluate this approach. Their prototype included the “active (future, present, perfect, past, and past perfect) and passive (present and past) voice of the verb, verbal and nominal genitive attributes, nominal groups including articles, adjectives, and nouns (declination classes, definiteness, and adverbial adjective modifiers), prepositional phrases, and simple subordinated clauses” and did not include modal verbs, negation, relative clauses, or coordination. In order to evaluate their system, they create 72 artificial variations of the sentence *Der Mann besichtigt den Marktplatz.* ‘The man visits the marketplace.’ that encompass the following kinds of errors: case selection for the subject and object, number agreement between the subject
and verb, and word order. They find that their system is able to find the intended analysis if
the sentence contains only one or two errors; for sentences with three or more errors, their
system finds the intended analysis less than half of time and it is unable to find any correct
analysis for sentences with five or more errors.

When constraints are allowed to be violated, the space of possible solutions can increase
dramatically in size, which leads to many spurious analyses and longer parsing times. In
order to curb this explosion, Menzel and Schröder (1998, 1999) introduce constraints from
additional linguistic levels of analysis. In particular, they add semantic restrictions on verb
arguments and domain knowledge to their prototype. In a language learning exercise where
the context is specified, domain knowledge can be used to decide whether a sentence is true
in this context. Partial analyses that are true within the domain can be preferred to those that
are false. Menzel and Schröder (1999) extend the 72 artificial variations from Menzel and
Schröder (1998) to include semantic restrictions on verb arguments and domain knowledge,
resulting in a test set of 648 sentences. With semantic restrictions and domain knowledge
that support the intended analysis, they are able to find the correct analyses for all sentences
with up to four errors and for over 50% of sentences with five to seven errors.

3.2.7 German Tutor/E-Tutor: Heift (1998)

German Tutor (Heift, 1998) (now expanded and renamed E-Tutor (Heift, 2010)) uses a
Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994) parser with a
context-free backbone. It uses lexical mal-rules to identify agreement and selection errors
and a separate technique to identify limited types of word order errors.
HPSG is highly lexicalized and represents lexical entries as feature structures, which provide at a minimum information about the phonological form of the word (PHON) and syntactic and semantic information (SYNSEM). A partial feature structure for the pronoun *er* ‘he’ is shown in Figure 3.16. The case, number, and person features for *er* can be found in the LOCAL feature structure. While parsing a sentence like *er lacht* ‘he laughs’, number and person information from the feature structures for *er* and *lacht* are required to unify. If the features do not agree, the sentence cannot be parsed.

As discussed in section 3.1.2, German Tutor (Heift, 1998) uses mal-rules in the lexicon to handle selection and agreement errors. Mal-rules in the lexicon are unlike mal-rules in the grammar because the scope of a mal-rule in the lexicon is one feature of a single word rather than a larger phrase or clause where the possibility of multiple errors in a single phrase leads to an explosion in the number of mal-rules. The number of mal-rules in the lexicon is limited by the lexical features associated with each entry. Section 3.1.2 gave a simplified example with context-free rules to illustrate mal-rules in the lexicon. In her system, Heift actually modifies the HPSG feature structures so that the parser will record
whether features like number and person agree instead of requiring that they agree. She accomplishes this by extending each feature with all possible values. The correct value has the feature `correct` and the incorrect values `error`. The feature structure for `er` would be expanded as shown in Figure 3.17. The entry for a verb like `lacht` would be modified so that it inherits the value of the 3rd feature from its subject rather than requiring a 3rd person subject. As a result, if `lacht` has a 2nd person subject, the value `error` will be recorded in its person feature; if there’s a 3rd person subject, the value `correct` will be recorded.

Errors can be identified by searching for `error` values that have been recorded in the final parse. The location of the `error` value identifies the location and type of error in the sentence. With these modifications to the lexicon, the HPSG grammar and parser do not need to be modified at all to analyze sentences with selection and agreement errors.
A separate technique is used to check for errors in verb position, which are the only word order errors that German Tutor detects. Verb positions are identified by examining particular locations in the context-free parse tree to determine whether the verbs are in the correct position. For example, a verb-second sentence is expected to have the finite verb in the position correct, shown in Figure 3.18. This position is determined with respect to the root S. The correct non-finite verb position is shown in the second tree in Figure 3.18. The non-finite verb is expected to be the rightmost daughter in the lowest level of the tree. In a verb-final sentence, both the non-finite and finite verbs are checked in the lowest level of the tree.

In effect, certain branches in the tree are checked in the final parse to determine whether the finite and non-finite verbs are in the error or correct positions. The main drawback of this method is that it cannot be extended to handle other types of ordering errors, such as Mittelfeld ordering errors, since it is not possible to associate them with particular locations in the tree.
German Tutor and E-Tutor have been in use at Simon Frazier University since 1999 (Heift, 2010). German Tutor/E-Tutor exercises are restricted to exercises with provided vocabulary such as building a sentence from a sequence of lemmas or translating a sentence. Restricted exercise types allow the system to detect spelling errors, missing and extra word errors, and word order errors in preprocessing steps rather than requiring the parser to detect such errors. In addition, the preprocessors require a stored list of correct answers for each exercise. For approximately 4,400 sentences produced by 33 learners over the course of a semester, Heift and Nicholson (2001) report that the system was able to correctly differentiate correct and incorrect responses in all cases. Note that a response may be grammatical, but incorrect for a particular exercise, so correct responses must be grammatical and include the target vocabulary. The system identified that 2,614 of the submitted sentences were incorrect (in content and/or grammaticality) and performed an error analysis, but the feedback provided was not always as detailed as desired by the authors.

3.2.8 Textana: Schulze (2001)

Textana, a grammar checker for English-speaking learners of German that focuses on the issue of feedback, uses an HPSG parser with constraint relaxation. If a constraint is relaxed, the type of error is recorded and the parser continues. Using the recorded errors, the parser can find the parse that has the fewest errors in order to identify the best parse. Schulze (2001) extends HPSG signs with a feature called REMARKS that stores information regarding which constraints have been violated. The parser can be configured to allow a particular subset of the grammatical constraints to be relaxed. Sample feedback from Textana is shown in Figure 3.19. The system explains the analysis it has found in simple
grammatical terms and identifies the words involved in the error. Textana was developed as a prototype and was never used by students.

**Input:** *Ich bauen den Tisch.* (I build the table)

**Feedback:** The Textana system analyzed “Ich” as the subject of the finite verb “bauen”. If you think the system’s analysis is correct, “bauen” should agree with “Ich” in person and number.

Figure 3.19: Textana: Sample Feedback

3.2.9 **DiBEx: Klenner and Visser (2003); Klenner (2004)**

Klenner and Visser (2003) and Klenner (2004) focus on the goal of creating explanatory dialogues when a sentence contains an error. They developed an error diagnosis component that their system DiBEx (Dialogue-Based Explanation) can use to create an appropriate dialogue. A sample dialogue from Klenner and Visser (2003) is shown in Figure 3.20. In this example, the learner has entered *Die Mond ist aufgegangen* ‘The moon has risen’, which contains an agreement error between die *fem* and Mond *masc*.

```plaintext
system: the article is wrong
user: why?
system: ‘die’ is feminine
user: so what?
system: the article must agree with the noun
user: what’s the problem?
system: German ‘Mond’ is masculine
user: i see
```

Figure 3.20: Sample Dialogue from Klenner and Visser (2003)
Their approach relies on a constraint-based representation of linguistic knowledge based on the work in Menzel (1992). They decided against using a parser augmented with mal-rules for reasons mentioned earlier this chapter in section 3.1.2, namely that mal-rules lead to a proliferation of rules, that it is difficult to pinpoint the type of error based on the form of a failed rule, and that the grammatical phenomena are complicated enough that it is not possible to anticipate all potential errors. Their system can handle errors in agreement, selection, and word order.

The problem of error diagnosis for DiBEx is simplified because the authors have a stored correct answer for each exercise. Instead of trying to parse the errorful answer, they use information from a range of sources to determine an error diagnosis: lexical features of the words in the provided answer, lexical features of the stored solution, and a syntactic analysis of the stored solution that has been generated using an LFG parser. They calculate the edit distance between the lexical features in the errorful answer and in the solution to determine which constituents to examine further. Based on predefined notions of agreement for the type of phrase, e.g., gender-case-number agreement in noun phrases, they can analyze the edits between the two phrases to determine the potential error types. The steps in this analysis are used to structure the explanation-based dialogues for the learners. According to their 2003 paper, their system is not yet able to handle multiple errors.

3.2.10 Fortmann and Forst (2004)

Fortmann and Forst (2004) propose error detection and diagnosis methods for an ICALL system that modifies an existing broad-coverage LFG parser (Butt et al., 1999; Dipper, 2003). Their general proposals are directed at well-known types of errors made by L2
learners of German and the work reported in their 2004 paper does not include an extended range of error types or an evaluation on learner data.

They propose mal-rules within the LFG framework that can handle ungrammatical word order, marked word order in the *Mittelfeld*, and agreement errors. The LFG functional-structure (f-structure), which represents grammatical functions, is augmented with a set feature `DAF-UNGRAM` that collects error types during parsing.

For ungrammatical word order and marked word order, mal-rules are added to the context-free rules in the the LFG constituent-structure (c-structure), which encodes syntactic constituents and word order. For more than one constituent in the *Vorfeld* (see section 2.1.6), they propose the following phrase-structure rule, which has been simplified from Fortmann and Forst (2004):

\[
CP \rightarrow XP \; XP^*[\text{add VORFELD to DAF-UNGRAM}] \; C'
\]

This rule allows any number of constituents (`XP^*`) to appear between `XP` (the expected initial constituent) and `C'` (the finite verb and any constituents after it). If an extra constituent appears in the `XP^*` position, the rule above records the error type `VORFELD` in the `DAF-UNGRAM` feature. In more detail: in the typical generative account of German word order, the constituent that appears in the *Vorfeld* has moved to the specifier of CP as in the following example *Er hat heute gelacht* ‘He laughed today’:

\footnote{DaF is a German abbreviation for *Deutsch als Fremdsprache* ‘German as a foreign language’.

72}
If any phrases appear between the initial constituent and C’ (i.e., XP* in the proposed rule), the sentence is ungrammatical, e.g.:

Because heute appears between the initial constituent and C’, the rule will record the error type VORFELD in the DAF-UNGRAM feature in the f-structure. A simplified representation of the f-structure for the example *Er heute hat gelacht is shown in Figure 3.23. The only difference between the f-structures for Er hat heute gelacht and *Er heute hat gelacht is the DAF-UNGRAM feature.
Fortmann and Forst’s approach for agreement errors is similar, however agreement errors have no effect on the c-structure, so the mal-rule that adds an error type to the DAF-UNGRAM feature is a so-called sublexical rule. For instance, if the subject and verb person features do not agree, the error type SVPERSAGR is added to DAF-UNGRAM. The DAF-UNGRAM feature is set-valued, so it can include multiple errors. For their example *Heute Otto siehst Anna ‘Today Otto see Anna’, which includes a word order error related to the Vorfeld and a subject-verb agreement error, the value of DAF-UNGRAM would be \{SVPERSAGR, VORFELD\}. As they acknowledge, using a set makes for simple bookkeeping, but it would cause problems in longer sentences where an error type such SVPERSAGR would not specify which subject/verb pair contained the error in a longer sentence.

3.2.11 PromisD: Reuer (2003a, 2005)

Reuer (2003a, 2005) introduces a mostly anticipation-free error recognition module that modifies an LFG parser with the modified feature unification and chart parser algorithms introduced in section 3.1.3. Reuer’s choice of LFG is motivated by several factors: efficient parsers already exist for large-coverage grammars (cf. XLE, Crouch et al., 2006),
the LFG concepts of f-structure and c-structure (cf. section 5.2.1) allow for grammatical configurations for a wide variety of languages, and many of the LFG primitives such as the notions of “subject” or “direct object” are expressed directly in the lexical entries and resulting parses, which makes it easier to generate feedback. Because the errors are annotated within the LFG feature structures, feedback can be generated from the annotations on the final parse. At each stage of parsing, partial parses have an error value related to the type of error (e.g., an agreement error has a lower value than a missing-word error). Items with a lower error value are preferred and items that exceed an upper limit are discarded.

Because the error detection approach relies solely on modifications to the parsing algorithm, Reuer’s parser uses a standard grammar and lexicon. In the f-structure, modified feature unification allows two clashing features to unify by keeping one of the possible values and storing the alternate value as an additional error feature. In the c-structure, four types of errors need to be addressed: insertions, deletions, substitutions, and word order errors. Reuer (2003a, 2005) focuses on two of these error types—deletions and word order errors—and proposes sketches of approaches for the insertions and substitutions.

Reuer’s modified feature unification algorithm adds a special feature err (mentioned previously in section 3.1.3) that keeps track of which feature could not be unified and what the alternate value of this feature could be. The err feature percolates up the tree and its value is specific to the current local context. An example from Reuer (2003a) shows how the err feature points to a person-feature mismatch within the subject NP but to a more general error in the subject once this NP has been parsed as the subject of the predicate:
This context-specific err feature percolation makes it unnecessary to use a mechanism such as the case filtering proposed by Schwind (1995) to disambiguate agreement errors (Reuer, 2003a). In order to find the specific error type, it is necessary to traverse the path in the parse to the original source of the error. In effect, the err feature keeps track of only local information about the error.

Missing and extra complements are detected using LFG’s completeness and coherence checks, which respectively check that all complements are present and that there are no extra complements. The modified chart parsing algorithm used to find deletions and word order errors is described in section 3.1.3. Deletions are limited to functional elements: “the chart-parsing component uses a part-of-speech (POS) list consisting mainly of ‘functional’ categories that restricts possible insertions of apparently omitted items in the sentence” (Reuer, 2003b, p. 500).

Reuer (2005) tested his system on 75 phrases collected from intermediate German learners at Humboldt University in Berlin and 75 sentences with exactly one morphosyntactic error per sentence from a corpus that demonstrates frequent types of errors made by learners (Heringer, 1995). The total test of 150 phrases/sentences contain a total of 169 errors. For 15% of the errors, the parser was unable to find a parse with a single root node given a limit
on the maximum number of edges (20,000). An incorrect error diagnosis was provided for 17% of the errors and 68% were diagnosed correctly with a single correct error diagnosis or with the preferred error type as one of multiple equally-ranked diagnoses. In some of the cases with an incorrect diagnosis, the preferred error was indeed found within the parse, but another error was ranked higher.

3.3 Summary

To augment a parser with the ability to provide analyses of ill-formed input, the grammar or the parsing algorithm needs to be adapted. The mal-rule approach modifies the grammar while the meta-rule approach and the modified unification and chart parsing algorithms adapt the parsing algorithm. Constraint-based approaches use constraint relaxation to detect errors, which may involve modifying the grammar (e.g., by adding weights to each constraint) and modifying the constraint solving algorithm so that it is possible to treat the problem as a partial constraint satisfaction problem.

Many ICALL systems developed for learners of German use a combination of these methods to address a variety of error types as shown in Table 3.1 at the beginning of section 3.2. Modified feature unification is often used to handle agreement errors (Schwind, 1988; Reuer, 2005). Heift (1998) uses mal-rules in the lexicon to handle selection and agreement errors while Schulze (2001) and Menzel and Schröder (1998) use constraint relaxation. Word order and missing/extra word errors are handled through mal-rules (Schwind, 1988, 1995), modified chart parsing algorithms (Reuer, 2005), or constraint relaxation (Menzel and Schröder, 1998).
A weakness in much of the recent research on parser-based ICALL for German is the lack of evaluation using real or realistic learner input. Many approaches have been evaluated using only a handful of sentences or with artificially generated test data. German Tutor/E-Tutor is the main exception to this (Heift, 1998, 2010), having been in use with German learners at Simon Fraser University for over a decade.
Chapter 4: Constraint Satisfaction Problems

Many everyday problems involve the intuitive notion of constraints. We have limited resources such as limited time or funds and we need to accomplish tasks such as scheduling meetings, seating guests at a dinner party, or determining which new camera to buy. These kinds of problems can be modeled as constraint satisfaction problems (CSPs), which define a set of variables and a set of constraints over these variables. CSPs extend from simple puzzles such as Sudoku to complicated resource allocation problems such as scheduling trains and associated employees across several countries. Beginning with work in the areas of computer vision and artificial intelligence in the 1960s and 1970s (Sutherland, 2003; Waltz, 1972), constraint programming has been an active area of research.

Constraint-based parsing approaches such as Constraint Dependency Grammar (Maruyama, 1990a) and Extensible Dependency Grammar (Duchier, 1999; Debusmann, 2006) treat parsing as the task of finding solutions (e.g., parsing or generation—connecting strings to categories) to CSPs that express constraints from the grammar and lexicon. This chapter will introduce the formulation of constraint satisfaction problems in section 4.1 and approaches for finding solutions to these problems in section 4.2. The material covered in this chapter is drawn from multiple sources including Kumar (1992); Tsang (1993); Barták (1999); Dechter (2003); Apt (2003); and Tack (2009).
4.1 Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) is a problem that consists of a set of variables, which are each limited to some finite domain, and a set of constraints that restrict the possible values of the variables. A solution to a CSP is a set of assignments for each variable from its domain that simultaneously satisfy all constraints.

The Sudoku puzzle shown in Figure 4.1 provides a good illustration of a CSP. Solving a Sudoku puzzle consists of figuring out which single digit between 1 and 9 should be entered into the empty squares so that each row, column, and outlined 3x3 grid contain each digit once. For instance, the value in the 4th row and 3rd column from the top left can be deduced to be 6 by examining the initial values in the relevant row, column, and outlined 3x3 grid. The 4th row contains \{1,2,5,9\}, the 3rd column \{3,4,5,7,8\}, and the relevant grid \{2,5,9\}, together \{1,2,3,4,5,7,8,9\}. This determines that 6 is the only possible value for this square. After 6 is filled in in this square, the values for the remaining squares can be determined in a similar fashion. This deductive approach to solving CSPs will be introduced in section 4.2.2.

The components of a Sudoku puzzle are easy to formulate as a CSP:

- Variables: each square in the grid
- Variable domains: 1..9 for each empty square; the filled-in value for the remaining squares
- Constraints:

10This Sudoku puzzle was generated using the program Su Doku Solver: http://sudoku.sf.net.
Formally, a constraint satisfaction problem is a triple $\langle X, D, C \rangle$ where $X$ is a finite set of variables $\{x_1, x_2, \ldots, x_n\}$, $D$ is a function that maps each variable in $X$ to a finite set of possible values known as the domain $D_{x_i}$ for the variable $x_i$, and $C$ is a finite set of constraints on the values of the variables in $X$. Each constraint can constrain an arbitrary subset of the variables in $X$. For a detailed formal definition of a CSP, see for example Tsang (1993); Dechter (2003) or other introductory textbooks on constraint satisfaction or constraint programming.

The constraint satisfaction problems relevant for Extensible Dependency Grammar parsing (see chapter 5) are a subset of CSPs where the variables range over finite sets of non-negative integers. To illustrate, a sample CSP is shown in Figure 4.2. There are three

- each row contains each digit from 1..9 exactly once
- each column contains each digit from 1..9 exactly once
- each outlined 3x3 grid contains each digit once
variables \{X, Y, Z\}. Variables \(X\) and \(Y\) are finite integer variables with domains of 1..5 and 2..8 respectively. \(Z\) is a set variable which ranges over the powerset \(\mathcal{P}\{(6, 7)\}\). There are three constraints expressed in predicate logic that constrain the assignments of one or more variables.

<table>
<thead>
<tr>
<th>Variables and their domains:</th>
<th>Constraints:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X \in {1..5})</td>
<td>(X &gt; 3)</td>
</tr>
<tr>
<td>(Y \in {2..8})</td>
<td>(X + Y &gt; 10)</td>
</tr>
<tr>
<td>(Z \in \mathcal{P}{(6, 7)})</td>
<td>(Y \in Z)</td>
</tr>
</tbody>
</table>

Figure 4.2: Sample Constraint Satisfaction Problem

### 4.2 Solving Constraint Satisfaction Problems

A constraint satisfaction problem states the variables, their domains, and the constraints on the variable values. A solution to a CSP consists of one set of assignments for all variables that satisfies all the constraints. A CSP is satisfiable if there is at least one solution. The sample problem from the previous section (Figure 4.2) is satisfiable and has six solutions:
Figure 4.3: Solutions to CSP in Figure 4.2

Note that the CSP itself states the problem without specifying any particular method for finding solutions. In general, solving a CSP involves a combination of two approaches: 1) search and 2) local and global consistency techniques. The following sections introduce search (4.2.1) and consistency (4.2.2), along with constraint propagation and distribution (4.2.4), which combines the consistency techniques and search to solve CSPs efficiently.

4.2.1 Search

The following sections introduce methods for systematically searching through the space of possible CSP solutions: *generate-and-test* and *backtracking search*.

**Generate and Test**

A naïve approach to solving constraint satisfaction problems is to generate each possible combination of variable assignments given the variable domains and then test to see which assignment or assignments satisfy the constraints. There are as many potential combinations of variable assignments as the product of sizes of the variable domains, so for the
The sample problem in Figure 4.2, there are \(|\text{domain}(X)| \times |\text{domain}(Y)| \times |\text{domain}(Z)| = 5 \times 7 \times 4 = 140\) possible variable assignments.

The generate-and-test approach is, of course, extremely inefficient because it generates a complete set of variable assignments before considering any of the constraints. For example, a constraint such as \(X > 3\) from the sample problem immediately reduces the space of possible solutions by more than half, but the generate-and-test approach would spend time generating all possible assignments with \(X \leq 3\) and repeatedly checking whether the other constraints are satisfied, even though the value of \(X\) immediately excludes all of these assignments.

In a sample Extensible Dependency Grammar CSP for a three-word sentence (see chapter 5 for an introduction to these CSPs), there are approximately 20,000 boolean variables with a domain size of 2; 500 integer variables with a domain size of \(\sim\)100,000,000; and 1,500 set variables with a domain size of \(\sim\)100,000,000! Like XDG CSPs, most non-trivial CSPs are unsuited for the generate-and-test approach.

**Backtracking**

The backtracking approach finds solutions to a CSP by “incrementally extending a partial solution that specifies consistent values for some of the variables, towards a complete solution, by repeatedly choosing a value for another variable consistent with the values in the current partial solution” (Barták, 1999, p. 557). In a sense, the backtracking interleaves generation and testing steps from the generate-and-test approach. If a newly generated assignment is not a possible partial solution, then the search process backtracks to a previous state and continues the search with different assignments. Standard chronological
backtracking, which backtracks to the most recent decision when a partial assignment does not satisfy the constraints, can be visualized as a depth-first search, as in the tree shown in Figure 4.4. Each node is a point where a new variable is added to the partial solution. Each branch represents one of the possible values for this variable.

The generate-and-test approach would generate the entire tree and test the complete assignment found in each leaf. In contrast, the backtracking approach tests the partial assignment in each node before generating its children. If the partial assignment cannot satisfy the constraints, then there is no point in extending this partial assignment, so the search backtracks to a higher node in the tree where other partial assignments have yet to be explored. Figure 4.4 shows that the search can completely ignore the subtrees below $X \in \{1, 2, 3\}$ because these partial assignments do not satisfy the constraints. The search can immediately backtrack to the top node and continue analyzing variable assignments containing $X = 4$ or $X = 5$. One solution is found in the highlighted node $X = 4, Y = 7, Z = \{6, 7\}$. The remaining six solutions would be found in the unexpanded subtrees in this diagram.
Backtracking is more efficient than generate-and-test because it allows subtrees to be excluded from the search when partial assignments cannot possibly lead to solutions. However, the chronological backtracking algorithm still suffers from three major problems (Bartáč, 1999):

- thrashing: the search fails due to the same conflict in multiple subtrees
- redundant work: conflicting variable assignments are not stored, so partial assignments with these values are generated again in later subtrees
- late detection of conflicts: conflicts are not detected until the relevant variable values have been assigned, often deep in the search tree

Numerous algorithms have been developed that extend the chronological backtracking algorithm to reduce the impact of these problems. To avoid thrashing, a technique called backjumping (Gaschnig, 1979) uses information about the conflicting values when it is necessary to backtrack in the search tree. It finds the most recent point at which a conflicting variable was assigned and jumps back to this point rather than backtracking to the most recent variable assignment. This prevents the search from generating numerous subtrees with a known conflict. To avoid redundant work, backmarking and backchecking (Haralick and Elliot, 1980) store known conflicting assignments to prevent partial assignments with these conflicts from being generated in later stages and to reduce the amount of constraint checking that is necessary at each step.

These techniques only look back at previously assigned variables, so they all still suffer from the late detection of conflicts: until a variable assignment is made, constraints affecting that variable are not taken into consideration. One way to improve the backtracking
algorithm is to use heuristics to order the variables in the search tree (cf. Tsang, 1993, chapter 6). For example, variables that are involved in the highest number of constraints or variables with the smallest domains are considered earlier in the search because they are more likely to lead to failure. If failures are detected earlier, less of the search tree needs to be explored. In order to improve the running time of constraint solvers even further, it is necessary to take unassigned variables into account by looking forward in the search tree. The following section examines consistency techniques that make it possible to detect conflicts by looking forward at unassigned variables.

4.2.2 Consistency Techniques

In the backtracking techniques described in the previous section, a constraint is first taken into account after all the variables involved have been instantiated, which can lead to a lot of fruitless searching deep in the search tree due to the late detection of conflicts. Instead of waiting until variables are instantiated, consistency checking uses knowledge about the constraints to remove values from variable domains that could not possibly be part of a solution. Consistency techniques, which are known by many names such as constraint inference, constraint propagation, and local consistency enforcing, reduce the size of the problem and thus the size of the search tree by mapping the current CSP to a simpler CSP with the same set of solutions.

The terminology used in consistency checking is based on a representation of a CSP as a graph where the variables correspond to nodes and the edges correspond to constraints. For the most common algorithms, the CSP is converted into an equivalent binary CSP, where a
A binary CSP can be easily pictured as a graph, as shown in Figure 4.5.

The simplest consistency technique, node consistency, examines the unary constraints on each variable. For example, the constraint $X > 3$ from the sample problem could be used to reduce the domain size of $X$ without reference to any other variables or constraints. In the sample problem, node consistency would reduce the domain of $X$ to $X \in \{4, 5\}$.

Consistency techniques involving binary constraints are known as arc consistency because the constraints correspond to arcs between nodes in the graph representation. Arc consistency involves removing any values that are inconsistent given a pair of variables and one constraint between them. For example, if we examine the variable $Y$ (domain of 2..8) and the constraint $Y \in Z$, we can determine that the values 2..5, 8 can be removed from $Y$ because these values can never be in $P(\{6, 7\})$. When values are removed from one variable’s domain, it becomes necessary to recheck any other constraints associated with this variable in case the removal has any side effects. Since the late 1970s, many arc consistency algorithms have been developed with improving space and time complexity:
Arc consistency is not sufficient to solve CSPs on its own because it ensures only a limited amount of local consistency. Arc consistency can determine that a CSP is not satisfiable by eliminating all values within a particular variable’s domain, but a CSP may be arc consistent without having any solutions. Consider the graph in Figure 4.6: if $a, b, c \in \{1, 2\}$, then the three inequality constraints can hold individually, but there are no global solutions.

Consistency techniques can be extended to include more than two variables and multiple constraints, which is known as path consistency after the path in the graph between the variables. Path consistency can reduce the size of the CSP more than arc consistency, however consistency techniques beyond arc consistency are not often used in real applications because of their higher space and time complexity (Barták, 1999).
4.2.3 Combining Consistency and Search Techniques

The previous two sections have discussed two approaches to solving CSPs: systematically searching through possible variable assignments and using consistency techniques to reduce the size of the current problem. Efficient consistency techniques are not sufficient on their own to solve CSPs, but they can be integrated into complete search algorithms to reduce the size of the search space. In order to create more efficient algorithms, search algorithms can be combined with different degrees of consistency techniques.

The table in Figure 4.7 from Kumar (1992) gives an overview of algorithms that combine chronological backtracking with consistency techniques, ranging from no consistency checking (Generate and Test) to global arc consistency (Really Full Lookahead). The right-hand column uses the abbreviations introduced in the lefthand column to informally describe the combination of search and arc consistency techniques used in each approach. For example, Partial Lookahead consists of Forward Checking (FC) plus a greater degree of arc consistency than in forward checking or backtracking.

Consistency techniques can also be combined with improved backtracking algorithms such as backjumping and backmarking to create hybrid algorithms (Prosser, 1993). The following sections describe algorithms combining search and consistency techniques that form the theoretical foundation for the algorithms on conflict detection that will be presented in chapter 6.
**Forward Checking**

Forward checking extends standard chronological backtracking by performing a limited amount of consistency checking at each node in the search tree. For each unassigned variable, forward checking removes any values that are not consistent with the current partial solution. If a variable's domain no longer contains any values, then there is no need to expand this node in the search tree any further. Freuder and Wallace (1992) make the following observation:

“[D]espite the fact that [forward checking] can be viewed as an integration of consistency processing and standard backtracking, forward checking is really almost the complement of standard backtracking. Standard backtracking checks a value for consistency against previously chosen values. With forward checking, when we propose a value we already know it is consistent with the previously chosen values (or else the consistency processing would already have pruned it away). Now we test it against the domains of the remaining, uninstantiated variables” (p. 40).

The forward checking algorithm can be demonstrated using a sample map coloring problem from Ginsberg (1993) shown in Figure 4.8. There are five nodes $A–E$ and the arc between them represent shared borders on the map and there are three colors available—red, yellow,
blue—to color each node. A solution to this problem is a coloring where no two adjacent
nodes share the same color.

The CSP has five variables (A–E) that each have a domain containing three colors. There
is a constraint for each edge in the graph that states that these two nodes may not share the
same color. The CSP is defined as shown in Figure 4.9.

Figure 4.8: Map Coloring Problem from Ginsberg (1993)

Variables:

\[
\begin{align*}
A & \in \{\text{red, yellow, blue}\} \\
B & \in \{\text{red, yellow, blue}\} \\
C & \in \{\text{red, yellow, blue}\} \\
D & \in \{\text{red, yellow, blue}\} \\
E & \in \{\text{red, yellow, blue}\}
\end{align*}
\]

Constraints:

\[
\begin{align*}
A \neq C \\
A \neq D \\
A \neq E \\
D \neq E \\
B \neq D \\
B \neq E
\end{align*}
\]

Figure 4.9: CSP for Map Coloring Problem in Figure 4.8
Figure 4.10 shows part of the search tree to demonstrate the forward checking algorithm. At the top node, when the color red is chosen arbitrarily for node $A$, the color of $A$ can be used to infer that red is not a possible assignment for variables $C$, $D$, and $E$. Likewise, when yellow is assigned to $B$, the color yellow can be eliminated for $D$ and $E$. When the nodes below $B = yellow$ are expanded, the node $C = red$ can be skipped entirely because an earlier assignment ($A = red$) has already ruled out this possibility. Next, previous assignments restrict $D$’s color to blue. When $D$ is colored, it immediately becomes clear that there are no more available colors for $E$, so it is necessary to backtrack back to the color choice for $B$. The search continues in the same manner with $B = red$. Forward checking reduces the size of the search tree, but it also requires additional reasoning at each node to eliminate values for unassigned variables.

**Full Lookahead**

The full lookahead algorithm, also known as AC-L (*arc consistency lookahead*), checks arc consistency between all unassigned variables at each node along with the same checks against the partial assignment as in forward checking. This allows the problem to be reduced more than in forward checking, but naturally requires more overhead for constraint checking at each node. If full lookahead were used in Figure 4.10, the additional arc consistency checking between the unassigned variables $D$ and $E$ at the node $B = yellow$ would show that there are no possible assignments for $D$ and $E$ given $A = red$ and $B = yellow$. The algorithm would backtrack at this point instead of exploring possible values for $C$.

11 For map coloring problems like this one, there are equivalent permutations of the color assignments with red, yellow, or blue chosen arbitrarily for the first node colored.
Dependency-Directed Backtracking

Like forward checking, *dependency-directed backtracking* (Stallman, 1977) stores information about possible variable assignments that have been inferred from the current partial solution. In addition, it stores information about conflicts (also called *nogoods*) that have been noted earlier in the search, such as a pair of variable assignments that can never be in a solution. In the map coloring CSP example, a stored conflict might be that $A$ and $B$ cannot be red and yellow respectively. When the search reaches a point where it needs to backtrack, the stored inferences and conflict information allow the algorithm to backtrack to a point in the tree where it can modify the value of variables that are known to have caused conflicts, but without performing unnecessary work on partial solutions that are known to lead to conflicts. The drawbacks to dependency-directed backtracking are the large space
and time requirements for calculating the correct node to jump back to and maintaining information about conflicts.

**Dynamic Backtracking**

Dynamic backtracking (Ginsberg, 1993) is a variant of dependency-directed backtracking that uses polynomial space to store information about conflicts. It addresses the same disadvantages of standard chronological backtracking: thrashing and performing redundant work because known conflicts are not recorded. In particular, dynamic backtracking is intended to address drawbacks in chronological backtracking for CSPs that contain independent subproblems. In a CSP with independent subproblems, the variables could be explored in an order that interleaves choices from multiple subproblems. When a conflict is found in one of the subproblems and backtracking is necessary, backtracking can erase useful progress in the unrelated subproblems.

The basic insight is that storing information about which constraints led to the elimination of particular variable values can allow the search algorithm to jump back to a useful variable assignment and at the same time restructure the search tree to modify the order in which variables are considered. When conflicting variables can be moved lower in the search tree (i.e., further away from the root of the tree), there are two advantages: 1) less work is undone when backtracking, because it is not necessary to backtrack as high up the tree and 2) less work is redone in a new subtree before a conflict is discovered, because lower nodes are closer to complete variable assignments and thus solutions.

The algorithm stores *eliminating explanations*: information about which constraint led to the elimination of a particular variable value. The stored information is represented using
a table using polynomial space. For each variable, the approach keeps track of the current value of the variable and stores for each element in its domain notes about other variable assignments that conflict with this value. For example, in the map coloring CSP in Figure 4.8, the table to store information about variable assignments and conflicts would look like this (example taken from Ginsberg, 1993):

<table>
<thead>
<tr>
<th>node</th>
<th>color</th>
<th>eliminating explanations</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>red</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>yellow</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>blue</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After coloring several nodes, the table would be partially filled in as follows:

<table>
<thead>
<tr>
<th>node</th>
<th>color</th>
<th>eliminating explanations</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>red</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>yellow</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>blue</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This indicates that $A, B, C$ have been colored red, yellow, and blue. The eliminating explanation $A$ in the red column indicates that the current value of $A$ (red) has eliminated red as a possible assignment for node $C$ due to the constraint $A \neq C$.

If the current coloring is extended further, there comes a point when all possible colors have been eliminated for one of the nodes:
Here, there are no possible assignments for $E$, so it is necessary to backtrack. The eliminating explanations guide the backtracking process and make it possible to backtrack through the previous assignments without necessarily losing all intermediate work. From the eliminating explanations $A, B, D$ for node $E$, it is clear that the search should backtrack to reconsider the color of one of these variables. No alternate coloring of $D$ can satisfy the constraints, so it is necessary to backtrack to $B$. When a node’s coloring is removed, the table is updated accordingly. After recognizing that $C$’s color is not involved in the conflict and removing the coloring for $B, D, E$, the table is updated:

<table>
<thead>
<tr>
<th>node</th>
<th>color</th>
<th>eliminating explanations</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>red</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>yellow</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>blue</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>blue</td>
<td>A B D</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>A B D</td>
</tr>
</tbody>
</table>

The eliminating explanations involving $B$ and $D$ have been removed, however those involving $A$, which have been determined by the failed partial coloring above, have been retained. Dynamic backtracking makes it possible to keep the color of $C$ instead of removing its color when backtracking to $B$ and it preserves information about conflicts that have been established during the search process. The idea of eliminating explanations will resurface in section 6.2.1 with Explanation-Based Constraint Programming, an approach for finding conflicts in over-constrained CSPs (Jussien, 2001).
4.2.4 Constraint Propagation and Distribution

Constraint propagation and distribution interleave consistency checking and search techniques in order to solve CSPs efficiently. Instead of constructing the search tree using possible values of each variable as in Figure 4.4, the branches in the tree are constructed using new constraints that divide the remaining subproblem efficiently.

At each step, feasible consistency techniques are applied to reduce the size of the problem. If the consistency techniques have not yet led to a solution, the solver proceeds by considering two complementary problems: the CSP with an additional constraint $C$ and the CSP with the constraint $\neg C$. This is known as distributing the CSP with $C$ (Schulte and Smolka, 2008). Each of the two new problems is reduced as much as possible using consistency techniques and then split into complementary problems in the same fashion. This continues recursively until the consistency techniques either lead to a solution or prove that the variable assignments under consideration cannot lead to a solution.

Using distribution with $C$ and $\neg C$ provides a complete search of the problem space and can be represented as a binary search tree. Returning to the sample problem from Figure 4.2, one possible search tree for this problem is shown in Figure 4.11, which was produced using Gecode Interactive Search Tool (Gecode Team, 2006). Gecode is a general-purpose constraint solver that uses constraint propagation and distribution. The circular nodes represent a partial, consistent assignment and the diamond nodes represent a solution.

Part of the tree from Figure 4.11 is shown in detail in Figure 4.12. In the top node, consistency checking has reduced the variable domains from the initial problem to: $X \in$
\{4, 5\}, Y \in \{6, 7\}, Z \in \mathcal{P}(\{6, 7\}). At this point, the consistency techniques cannot reduce the problem any further, so distribution comes into play. The constraint involved in the top-level distribution step is \(Y = 6\) in the left branch and \(Y \neq 6\) in the right branch. Given the new constraint \(Y = 6\) in the left branch, consistency techniques can reduce the domain of \(X\) to \(\{5\}\) and the domain of \(Z\) to \(\{\{6\}, \{6, 7\}\}\). In each branch, consistency techniques and distribution are interleaved in each branch until solutions are found.

The constraint \(C\) chosen in each distribution step should be selected so that it is possible for the constraint propagation step to reduce the problem further once \(C\) or \(\neg C\) has been added. Typically, an as-of-yet unassigned variable \(x\) is chosen and the unary constraint \(C\) is constructed based on the domain of \(x\). For illustration, consider a domain for \(x\) consisting of \(\{\text{min}, \ldots, \text{median}, \ldots, \text{max}\}\). Some possibilities for \(C\) are \(x = \text{min}\), \(x = \text{max}\), \(x = \text{median}\), or \(x \leq \text{median}\).
The choice of variable and constraint in the distribution step is important because a well-chosen variable-value pair can greatly reduce the size of the search tree. A variable can be chosen randomly or by taking the first variable out of the list of remaining unassigned variables under consideration. One simple technique that often leads to smaller search trees is to choose the variable that currently has the smallest domain (Schulte and Smolka, 2008).

For specific types of problems, custom distribution techniques can be used to reduce the size of the search tree. The Extensible Dependency Grammar problems, which would take an extremely long time to solve using naïve distribution techniques, include distribution techniques that pick variables relevant to the structure of the dependency graphs.

### 4.3 Summary

Constraint satisfaction problems include a set of variables, variable domains, and a set of constraints. A solution to a CSP consists of an assignment for each variable that simultaneously satisfies all the constraints. Solutions can be found by searching systematically...
through the possible variable assignments or by applying consistency techniques that use
the constraints to reduce variable domains. Efficient constraint solvers use constraint prop-
agation and distribution, which use a combination of consistency and search techniques to
find solutions. For further reading, Tsang (1993), Dechter (2003), and Apt (2003) provide
general introductions to constraint satisfaction problems and constraint solving. See Tack
(2009) for more information about constraint propagation and its implementation in the
general-purpose constraint solver Gecode.
Chapter 5: Parsing as a Constraint Satisfaction Problem

This chapter serves as an introduction to Extensible Dependency Grammar (XDG) and the XDG constraint parser, which treats parsing as a constraint satisfaction problem (CSP). The first section touches on the earliest treatment of dependency parsing as a CSP, Constraint Dependency Grammar, and the following sections focus on XDG, which is the grammar formalism used in the grammar checking approach presented in chapter 10.

5.1 Constraint Dependency Grammar

The first formalization of dependency parsing as a constraint satisfaction problem was Constraint Dependency Grammar (CDG) (Maruyama, 1990a). A CDG grammar G is a 4-tuple consisting of:

- \( \Sigma \): a finite set of terminal symbols
- \( R \): a finite set of role IDs (such as governor)
- \( L \): a finite set of labels (such as subject, determiner)
- \( C \): a conjunction of unary and binary constraints that a parse should satisfy

A CDG parser first forms a constraint network using a “core” grammar that contains syntactic constraints. Next, constraint propagation in the form of arc consistency is used to
remove local inconsistencies. If ambiguity remains, additional (non-core) constraints such as semantic constraints can be added until an unambiguous parse is found.

An example core grammar from Maruyama (1990a) focuses on PP attachment. The first three components are:

- $\Sigma = \{V, NP, PP\}$
- $R = \{governor\}$
- $L = \{ROOT, OBJ, LOC, POSTMOD\}$

The set of constraints includes constraints such as:

- $\text{word}(\text{pos}(x)) = PP \Rightarrow (\text{word}(\text{mod}(x)) \in \{PP, NP, V\}, \text{mod}(x) < \text{pos}(x))$

which can be restated as: “A PP modifies a PP, an NP, or a V on the left.” A sample parse is shown in Figure 5.1 for the sentence “Put the block on the floor on the table in the room.”
5.1.1 Weighted Constraint Dependency Grammar

Weighted Constraint Dependency Grammar (WCDG) is an extension of CDG with weighted constraints (formalized in Schröder, 2002) that uses constraint weights to model the concept of gradation: there is a range of degrees of acceptability, grammaticality, and naturalness. Allowing for some constraints to be relaxed makes the parser more robust, while preferences, which are also stated as weighted constraints, can incorporate information from other levels of linguistic analysis to reduce the number of highly-ranked parses.

WCDG was mentioned in section 3.2.6 as the formalism underlying the constraint relaxation approach in Menzel and Schröder (1998, 1999). Parsing is treated as a partial constraint satisfaction problem. WCDG assigns each constraint a weight from 0.0–1.0 and multiples the weights of all constraints that are relaxed to give a parse a score. Allowing constraints to be relaxed causes the search space to increase dramatically, so WCDG uses heuristic search methods to find good, although not necessarily optimal, parses in a moderate amount of time (Schröder, 2002).

The WCDG approach appears quite attractive from an error diagnosis perspective and its developers have performed small pilot studies on its use in ICALL (Menzel and Schröder, 1998, 1999, see section 3.2.6), however the ad-hoc form of the existing German grammar (Foth et al., 2004, 2005), which focuses mainly on robust parsing for native German, does not provide the same precise or extensible characterization of German, in particular for word order, as Extensible Dependency Grammar.
5.2 Extensible Dependency Grammar

Extensible Dependency Grammar (XDG) takes a model-theoretic view of multiple (i.e., extensible) dimensions of linguistic analysis using dependency grammar-based representations. XDG was preceded by Topological Dependency Grammar (TDG) (Duchier and Debusmann, 2001b; Debusmann, 2001a), which focused on an analysis of German word order using only two dimensions: immediate dominance (ID) and linear precedence (LP). XDG is a “metagrammatical framework for dependency grammar” that extends TDG to an arbitrary number of dimensions (Debusmann, 2006). Each XDG instance defines a set of principles that specify the grammar.

The following sections will describe the XDG formalism, how dependency grammar can be defined using XDG multigraphs, the XDG analysis of word order, XDG’s treatment of parsing as a CSP, and will conclude with a brief discussion of its expressivity and complexity.

5.2.1 XDG Formalism

This section presents the XDG formalism by introducing the XDG definition for dependency graphs and multigraphs along with the components of an XDG grammar.

Dependency Graph

XDG defines a dependency graph as a graph where each node corresponds to a word in the input sentence. Each node has an associated attribute-value matrix (called a record) that stores features associated with it. Nodes in the dependency graph are connected by
directed, labeled edges. There are no other requirements on the shape of the dependency graph: it is not required to be connected, projective, or acyclic.

Throughout this thesis, dependency graphs will be shown in the style used in the XDG literature. An example graph is shown in Figure 5.2. The numbered words are shown underneath the graph and vertical dotted lines connect the sequential word IDs to nodes in the graph. The graph nodes are drawn in a tree-like format and labeled edges are drawn in straight lines between the nodes.

**Multigraphs**

The main model of XDG is the *multigraph*, a tuple of dependency graphs that share the same nodes. Each dependency graph is called a *dimension*. Debusmann (2006) defines *multigraphs* as follows:

**Definition 1** (Multigraph). A *multigraph* is a tuple $(V, D, W, w, L, E, A, a)$ consisting of:

1. a finite interval $V$ of the natural numbers starting from 1 called nodes

2. a finite set $D$ of dimensions
3. a finite set $W$ of words

4. the node-word mapping $w \in V \rightarrow W$

5. a finite set $L$ of edge labels

6. a finite set $E \subseteq V \times V \times D \times L$ of edges

7. a finite set $A$ of attributes

8. the node-attributes mapping $a \in V \rightarrow D \rightarrow A$

An example multigraph for the sentence *er hilft mir* ‘he helps me’ is shown in Figure 5.3. In order to make multigraphs easier to depict, the dimensions are each drawn separately with the nodes repeated even though the underlying representation contains a single node for each word. The two dimensions (ID, LP) are labeled to the left of each graph. In the ID dimension, the main verb *hilft* has two dependents, the subject *er* and the dative object *mir*. In the LP dimension, the set of edge labels corresponds to topological fields (see chapter 2) and each node receives an additional label such as $n$ or $v12$. The main verb appears in the left bracket ($v12$) and the subject and object appear in the Vorfeld ($vf$) and Mittelfeld ($mf$) respectively. Not shown in the diagram are the attributes associated with each node, which will be described in the following section covering the components of an XDG grammar.

**XDG Grammar**

XDG is defined as a description language for multigraphs: an XDG grammar describes a set of grammatical multigraphs for a given language. The grammar consists of three parts: the dimensions, the principles, and the lexicon.
An XDG grammar defines the types of dimensions such as ID (immediate dominance), LP (linear precedence), SEM (semantics), or IS (information structure). Each dimension has a name, a set of edge labels, and a set of attributes.

The principles “state the well-formedness conditions of the XDG models” (Debusmann, 2006). Principles can specify traditional grammatical constraints (selection, agreement, etc.) along with the shape of the dependency graphs in each dimension and the relationship between dependency graphs in two different dimensions. Many general-purpose principles are provided by default for XDG grammar writers. For example, a grammar may use the tree principle to require that the graph in the ID dimension is a connected tree or the projectivity principle to require the graph in the LP dimension to be projective. The valency principle can ensure that each word has all its required arguments and the agreement principle can check that subjects and verbs agree.
The lexicon contains a set of lexical entries. Each lexical entry specifies the word and attributes for each dimension. A sample lexical entry for the 3rd person singular verb *hilft* ‘helps’ is shown in Figure 5.4. This particular entry for *hilft* refers to the use of *hilft* in declarative main clause, i.e., *hilft* as it would appear in the second position in a verb-second clause. In the ID dimension, the word *hilft* has the agreement features ‘third’ and ‘singular’ and it requires as dependents one subject (subj), one dative object (iobj). It may be modified by 0 or more adverb phrases (adv*). In the LP dimension, *hilft* receives the label ‘v12’ (verb in the first or second position) and requires as a dependent one element in the Vorfeld ‘vf’*. Other dependents appear as Mittelfeld ‘mf’ elements.

Using the provided high-level principles, XDG grammars have been developed for fragments of Arabic, Czech, Dutch, English, French, and German (Debusmann, 2006).

### 5.2.2 Dependency Grammar and Multigraphs

Debusmann (2006) describes how concepts from dependency grammar can be stated as principles on multigraphs. As mentioned in the previous section, principles can be formulated that restrict a graph’s structure to a directed acyclic graph or to a tree. For example, the principle that requires a dimension to be a tree would state that: 1) the graph is acyclic,
2) there is exactly one root node, and 3) each non-root node must have exactly one head. Each principle uses combination of low-level constraints such as inequality or set membership on nodes and their attributes that express a high-level grammatical constraint such as agreement. Principles can define projectivity, the relationship between lexical entries and instantiations of lexical items, selection, agreement, word order, etc.

5.2.3 Word Order in XDG

As shown in the figures in section 5.2.1, XDG uses a novel treatment of word order that “distinguishes two orthogonal yet mutually constraining structures: a syntactic dependency tree and a topological dependency tree” (Duchier and Debusmann, 2001c, p. 180). The idea of separating an unordered syntactic dependency tree from an ordered word order tree goes back to the theory of word order domains (Reape, 1994). First proposed as part of Topological Dependency Grammar, a precursor to XDG, Duchier and Debusmann’s framework is intended to address some of the challenges many grammar formalisms have had modeling word order for so-called freer word-order languages such as German.

For word order in German, the two most relevant dimensions of an XDG multigraph are the immediate dominance (ID) dimension, which looks like a traditional dependency tree with grammatical relations such as subject and object between the nodes, and the linear precedence (LP) dimension, which uses concepts from the topological field analysis of German word order. The ID dimension is an non-ordered dependency tree with no constraints on projectivity, and ID constraints such as selection and agreement make no reference to word order. An LP dependency tree is a projective, ordered tree that relies on information from the ID dimension to determine which topological field each word should be assigned to.
The basic idea is that certain nodes are allowed to climb up the tree in order to create a projective analysis. The LP dependency tree ends up as a projective flattening of the ID tree. Principles limit which nodes may climb (such as past participles or adverbs) and certain nodes (such as finite verbs or the head noun in a noun phrase) act as barriers to climbing so that, for instance, arguments cannot climb out of a subordinate clause.

At the clause level, the edge labels used in the LP dimension correspond to topological fields (see chapter 2). Each lexical entry determines the topological fields (such as \( v_{12} \) or \( mf \)) that can be assigned to each word using the on feature as shown in Figure 5.4. Within other phrase types such as noun phrases, XDG has proposed topological field-like labels such as noun Vorfeld \((nvf)\) for determiners and noun Nachfeld \((nrf)\) for genitive modifiers of nouns.

XDG’s order principle defines a strict partial order on the set of edge labels of each daughter. The principle is applied to each node in the LP graph independently. In order for a node to be ordered among its daughters, the on feature is used to give it an additional label in the LP dimension, for example \( v_{12} \) for hilft as in Figure 5.3. The strict partial order for this example would be \( vf \prec v_{12} \prec mf \). This order constraint along with the LP out feature for hilft (see Figure 5.4) would prevent sentences such as er mir hilft or hilft er mir, since there needs to be exactly one Vorfeld element.\(^\text{12}\)

The example sentence mir hat er gestern geholfen lit. ‘me has he yesterday helped’ in Figure 5.5 shows the ID and LP dimensions for a sentence with a non-projective ID dimension.

\(^\text{12}\)These are not ungrammatical sequences in German, but the lexical entry for hilft under consideration from Figure 5.4 is the form that appears in non-embedded, declarative clauses.
and where climbing can be seen in the LP dimension. In the ID dimension, the dative object mir ‘me’ and the adverb gestern ‘yesterday’ are dependents of the main non-finite verb geholfen ‘helped’. In the LP dimension, mir and gestern are the children of a higher node in the tree hat ‘has’, effectively having climbed up in the tree.\textsuperscript{13} The non-finite verb geholfen appears in the topological field vcf for the verbal complex that appears in the right bracket.

A detailed description of the order, climbing, and barrier principles can be found in Debusmann (2001a).

\subsection{5.2.4 XDG Parsing as a Constraint Satisfaction Problem}

An XDG grammar imposes constraints on the appearance of grammatical utterances for a language without specifying or requiring a particular parsing approach. An XDG parser can use any approach that finds parses that satisfy the constraints or proves that there is

\footnote{It is easiest to describe this process in a derivation manner, i.e., that nodes climb up in the tree, but there is no derivational process involved. The relevant principles merely relate edges in the ID dimension to edges in the LP dimension without specifying any kind of derivation or movement.}
no parse for a particular sentence. Duchier (1999), which presents the earliest formulation of XDG-style parsing as a constraint satisfaction problem, summarizes the approach as follows:

“Our approach stands in sharp contrast to most extant parsing techniques. We abandon the generative view; we no longer build partial parses by combination of smaller ones, as is the case in e.g. chart parsing. Rather, we give a global well-formedness condition and proceed to enumerate its models.

“We choose dependency grammar (DG) as our framework and axiomatize the notion of a syntactically well-formed dependency tree in a manner that is particularly well-suited to constraint-based processing: we propose to regard dependency parsing as a finite configuration problem that can be formulated as a constraint satisfaction problem (CSP).

“This view takes advantage of the fact that for a sentence of length \( n \), there are finitely many possible [dependency] trees involving just \( n \) nodes. Out of this large number, we must select those that are grammatical. We do not attempt this through explorative generation, but rather via model elimination. What constraint programming affords us is effective model elimination through constraint propagation.”

“The key to success, presented here, is to reduce parsing to a problem that constraint programming (CP) is good at, namely configuration.”

(Duchier, 1999, p. 115–116)

In order to treat parsing as a constraint satisfaction problem, it is necessary to model multigraphs and principles using finite domain variables. The following sections describe how multigraphs are modeled using finite sets of integers and how principles are modeled using finite domain and finite set constraints.

**Multigraphs**

Multigraphs are modeled using finite sets of integers. A multigraph consists of set of records with one record for each node. Each node stores its lexical features along with
Figure 5.6: XDG Record for hilft in er hilft mir

information about its attributes and its position in the dependency tree in each dimension.

For example, the record for the node hilft from the parse in Figure 5.3 would look like

Figure 5.6.

An integer or set variable is used to store each of the features. Feature values like the tuple
(third sg) are encoded with integer values. A brief overview of the features found for hilft in Figure 5.6:

- **index** stores the position of the node in the sentence
- **word** stores the word itself
- **id/lp** dimensions are each stored in a separate record containing:
  - **entry**: the complete lexical entry for the word
  - **attrs**: the attributes of the word in this parse
  - **model**: the structure of this dependency parse
Graphs are modeled using the *model* record stored in each dimension. To make it easy to develop new graph-related principles, each node stores a number of set features that describe the structure of the graph: the node itself, its immediate mothers, its immediate daughters, all of its parents, all of its children, the nodes equal to or above this node, the nodes equal to or below this node, the mother nodes with edge labels, the daughter nodes with edge labels, the nodes above this node with edge labels, and the nodes below this node with edge labels. The diagram in Figure 5.6 shows only the feature *daughtersL*, which stores the daughter nodes by edges label, from which the entire graph can be constructed.

XDG principles refer to the features in the multigraph’s node records in order to post constraints on the nodes and edges of a graph. Each principle consists of low-level constraints such as equality, implication, and set membership that form a high-level constraint such as agreement. As an example, a simplified agreement principle applied to a single pair of words is shown in Figure 5.7 for the words *word1* and *word2*. It requires that *word1.attr* and *word2.attr* are equal in the case where *word2* is the *rel* daughter of *word1*.

\[
((word2 \in word1.daughters) \rightarrow ((rel \in relations) \rightarrow (word1.attr == word2.attr)))
\]

**Figure 5.7: Core Constraints in XDG Agreement Principle**

This agreement principle uses five low-level constraints in order to express the high-level agreement constraint. The general-purpose XDG agreement principle takes arguments including the relevant dimension, name of the relation, and the attributes that should agree. An XDG grammar would use the *agreement* principle multiple times to specify different
types of agreement, e.g., for person-number agreement between subjects and verbs or case-gender-number agreement between determiners and nouns.

**Constraint Parser Implementation**

The XDG Development Kit (XDK) (Debusmann et al., 2004) provides a constraint-based parser and a graphical user interface for grammar development and parsing. XDK is implemented in the Oz programming language (Smolka, 1995) using the Mozart Programming System (Mozart Consortium, 1999–2008). Mozart provides so-called *search engines* that perform constraint propagation and distribution to find solutions to constraint satisfaction problems (e.g. Schulte, 1997; Tack, 2002).

The XDG parser has three main stages (Debusmann, 2006): i) create node records for each word in the input sentence, ii) map each word to the corresponding list of lexical entries, and iii) post principles. The principles each have a priority that determines the order in which they are posted. This ranking helps optimize the search for the search engine. For instance, agreement constraints refer to relations between words that cannot be established until selection constraints have been posted, so posting agreement constraints before selection constraints could lead to extraneous searching. Rather than relying on the distribution mechanisms provided by the Mozart search engines, which wouldn’t have knowledge about which features in the node records are relevant, XDK principles that require distribution specify distribution on particular attributes explicitly.
5.2.5 Expressivity and Complexity

Debusmann (2006) proves that it is possible to construct a weakly-equivalent XDG for every CFG as long as the CFG does not generate the empty string. It is also possible to model the non-context-free languages \( \{a^n b^c c^n | n \geq 1\} \), \( \{n^1 \ldots n^k v^1 \ldots v^k | k \geq 1\} \) (cross-serial dependencies), and \( \{\sigma(n^1 \ldots n^k)v^1 \ldots v^k | k \geq 1, \sigma \text{ is a permutation}\} \) (scrambling).

In the general case, solving finite domain constraint satisfaction problems is an NP-complete problem. In an unpublished manuscript, Debusmann (2007) uses a formulation of XDG in terms of first-order logic to prove upper and lower bounds for three different recognition problems with a given grammar \( G \) and string \( s \):

Given a grammar \( G \) and a string \( s \), is \( s \) in \( L(G) \)?

1. universal recognition problem: both \( G \) and \( s \) are variable
2. fixed recognition problem: \( G \) is fixed and \( s \) is variable
3. instance recognition problem: the principles are fixed, and the lexicon and \( s \) are variable

The lower bound for the universal and instance recognition problems is NP-hard and the upper bound is in NP. The fixed recognitions problem’s lower bound is PSPACE-hard and the upper bound is PSPACE.

5.3 Summary

Extensible Dependency Grammar is a metagrammatical dependency grammar framework that treats parsing as a constraint satisfaction problem. An XDG grammar consists of a set of dimensions, a lexicon, and a set of principles that define traditional grammatical constraints such as agreement and also the shape of the dependency analysis. XDG dependency...
analyses are modeled across multiple dimensions as multigraphs, a tuple of dependency graphs that share the same words. In order to frame parsing as a constraint satisfaction problem, multigraphs and features-attribute matrices are modeled using finite set variables. Principles state constraints on the boolean, finite domain, and finite set variables in the multigraph model.
Chapter 6: Detecting Conflicts in Overconstrained CSPs

This chapter presents previous work in two related areas: finding partial solutions to over-constrained problems and figuring out which constraints lead to the lack of solution. There are a number of contexts in which partial solutions are desirable: a problem is over-constrained and has no solutions, the problem is too difficult to solve completely but a partial solution is also useful, or it is necessary to find the best solution given limited resources such as limited time or memory.

In the context of grammar checking using an XDG parser, it is necessary to find partial solutions and to pinpoint conflicting constraints in order to provide useful feedback to a learner. An ungrammatical input sentence results in an overconstrained XDG CSP, which has no solutions. A CSP solver, such as the Mozart/Oz solver used by the XDG Grammar Development Kit (Debusmann et al., 2004), will simply return a failure for ungrammatical input without providing any further information, meaning the only feedback that could be provided to the learner is simply grammatical or ungrammatical. Algorithms for finding partial solutions to CSPs and identifying conflicting constraints make it possible for the grammar checker to provide detailed feedback.

Section 6.1 presents modifications to algorithms described in chapter 4 that make it possible for them to find partial solutions to overconstrained CSPs and section 6.2 describes two
approaches to finding the source of conflicts in overconstrained CSPs, explanation-based constraint programming and QUICKXPLAIN. Section 6.3 evaluates the suitability of these approaches for use in the Fledgling grammar checker.

### 6.1 Partial Constraint Satisfaction Problems

Freuder and Wallace (1992) define partial constraint satisfaction as “finding values for a subset of the variables that satisfy a subset of the constraints” (p. 21). In other words, some of the constraints are “weakened” in order to allow more solutions. They present modifications to standard constraint satisfaction algorithms that are designed for maximal constraint satisfaction, which means that the algorithms search for solutions that satisfy the maximal number of constraints possible. In particular, they introduce modified algorithms for backtracking, backjumping, backmarking, arc consistency, and forward checking.

Their approach is restricted to binary, finite CSPs: CSPs where each constraint involves at most two variables and each variable’s domain is a finite set of values. This allows a constraint to be represented as a set of permitted pairs of values. In developing new algorithms, their goal is to minimize the number of constraint checks, i.e., the number of times the algorithm checks whether a new variable value satisfies a constraint involving it and other variables in the current partial variable assignment under consideration.

As mentioned in section 4.2.2, binary CSPs can be represented as a graph where the variables correspond to nodes and the edges correspond to constraints. The CSP in Figure 6.1, taken from Freuder and Wallace (1992), describes a problem where a robot is given the task of selecting a shirt, slacks, and shoes for an outfit. The shirt can be green or white, the

---

14This is just one of many metrics that can be used to rank partial solutions.
slacks can be denim, blue, or gray, and the shoes can be Cordovans or sneakers. Constraints restrict the robot to “fashionable” choices as shown in the constraints on each edge in the diagram. This problem is overconstrained, so there are no solutions.

The following sections will use this problem to illustrate two algorithms for solving partial constraint satisfaction problems: branch and bound for chronological backtracking and an extended version of forward checking, which combines backtracking with arc consistency techniques.

### 6.1.1 Search Techniques

The branch and bound algorithm adapts the backtracking algorithm (see section 4.2.1) to find partial solutions to overconstrained CSPs. The backtracking search for the robot CSP in Figure 6.1 is shown in Figure 6.2. In backtracking, a failure is noted as soon as one
constraint cannot be satisfied and to save unnecessary work, any further choices below this point in the search tree are not explored. The backtracking algorithm initially chooses Cordovans and then tries to pair them with denim slacks, which leads to a failure. Next, it tries blue slacks, which also leads to a failure. Then, it tries gray pants, which go with the Cordovans, but then no shirt choices satisfy the constraints. It backtracks to the top of the tree and tries again with sneakers, but again, there are no solutions.

The branch and bound algorithm uses the same depth-first tree traversal as backtracking, but in order to find partial solutions, it redefines the notions of failure and best solution. As mentioned above, the best solution has been defined as the solution that violates the fewest number of constraints. Because the branch and bound algorithm needs to consider partial solutions, the it abandons a branch only if it cannot possibly contain any solutions better than the current best solution. This means that a failure is noted only when the current branch cannot find any solutions that violate fewer constraints than the current best
solution. In order to do this, branch and bound keeps track of how many constraints the current partial solution does not satisfy \((d)\) and how many constraints are not satisfied in the best solution found thus far in the search \((N)\).

Figure 6.3 shows the branch and bound algorithm applied to the robot CSP. Instead of abandoning the leftmost denim branch as soon as it notices a conflict, it instead stores the number of violated constraints \((d = 1)\) and continues down this branch. With the green shirt, a total of three constraints are violated \((d = 3)\), so the current best solution \((\text{Cordovans, denim slacks, green shirt})\) has \(N = 3\). The search continues by substituting the green shirt for a white shirt, which leads to a solution where only one constraint is violated \((d = 1, N = 1)\). Cordovans with blue slacks have \(d = 1\), which is not smaller than the current value of \(N\), so there is no reason to continue exploring this branch and it backtracks to try out the gray slacks. Clearly, since this problem is overconstrained and the only solution better than \(d = 1\) would be \(d = 0\), the rest of the search is unable to find any better solutions. It could potentially find solutions that are just as good under \((\text{sneakers, blue slacks})\) or \((\text{sneakers, gray slacks})\), but there’s no need to continue the search when the already established complete variable assignment \((\text{Cordovans, denim slacks, white shirt})\) is just as good.

Search algorithms such as backjumping and backmarking can also be modified in similar ways for partial constraint satisfaction problems by adding bounds on the number of constraints that can be violated in a solution and adapting the steps that determine how far to jump back in the tree or whether to continue extending a partial solution.
### 6.1.2 Consistency Techniques

Algorithms that involve consistency techniques such as arc consistency and forward checking can also be modified for use in solving partial constraint satisfaction problems. The forward checking algorithm (see section 4.2.3) uses a limited amount of arc consistency to eliminate inconsistent values for unassigned variables at each node in the search tree. With a partial constraint satisfaction problem, arc consistency cannot be used to eliminate values outright unless there is already a known lower bound $N$ on the number of violated constraints in a partial solution. Without such a bound, forward checking can use inconsistency counts for each variable value in order to store the number of times it has been involved in an inconsistent arc consistency check thus far in a branch of the search tree.
The search tree can be pruned by abandoning partial solutions that match or exceed the number of inconsistency counts found in an earlier branch. The Cordovans branch of the search tree shown in Figure 6.4 shows how inconsistency counts are tracked for as-of-yet unassigned variables. At the top node, the slacks values denim and blue and the shirt color green are recorded as conflicting with the choice of Cordovans. The denim branch below it shows that green is known to conflict with two previous assignments, both Cordovans and denim. The shirt color assignments below denim indicate the number of violated constraints $d$ and the current lower bound $N$, which is 1 for (Cordovans, denim, white). The choice of blue slacks can do no better than this assignment, so this branch can be pruned immediately. The gray slacks branch shows that Cordovans plus gray slacks leaves no possible values for shirt color than can violate fewer constraints than our lower bound, so this branch can also be pruned.
6.2 Detecting Conflicts in Overconstrained CSPs

The partial constraint satisfaction algorithms from the previous section make it possible to find a partial solution to an overconstrained CSP, but they do not necessarily point to a reason why the CSP is overconstrained. This section first discusses explanation-based constraint programming, an approach based on dynamic backtracking that is well-suited for finite domain constraints but not finite set constraints, and then turns to the approach that will underlie the Fledgling grammar checker, QUICKXPLAIN, a general-purpose conflict detection algorithm for overconstrained CSPs.

6.2.1 Jussien: Explanation-Based Constraint Programming

Explanation-based constraint programming (Jussien, 2001) extends dynamic backtracking (Ginsberg, 1993, see section 4.2.3) in order to provide explanations for failure in overconstrained constraint satisfaction problems. There are several types of explanations. One type of explanation is a contradiction explanation (also called a nogood), a subset of the constraints in a CSP that lead to a failure. In the next section, the algorithm QUICKXPLAIN will be seen to provide contradiction explanations.

A second type of explanation is the eliminating explanation, which was mentioned previously in the description of dynamic backtracking. An eliminating explanation notes which other variable assignment(s) eliminated a single value \(a\) from the domain of a variable \(v\). Contradiction explanations can be derived from eliminating explanations: when all the values from a variable’s domain have been eliminated, it is possible to construct a contradiction explanation from the eliminating explanations for that variable.
In order to keep the space complexity for his algorithm polynomial, Jussien (2001) keeps only one eliminating explanation for variable value. It is necessary for the algorithm to forget explanations that are no longer relevant to the current variable assignment. He keeps one explanation at a time for each variable value, so the amount of space required is limited by the number of variables and the size of their domains.

Jussien (2001) notes that:

> “Indeed, constraint solvers always know, although it is scarcely explicitly, why they remove values from the domain of the variables. By making that knowledge explicit, quite precise and interesting explanations can be computed” (p. 3).

His explanation-based algorithm is implemented by modifying the propagation functions for each constraint (e.g., $\text{equals}(A, B)$, $\text{lessthan}(A, B)$, $\text{alldifferent}(A, B, C, ...)$, etc.) in the constraint solver. When a constraint is added to a CSP, the customized functions update the explanations for all the variables involved in the constraint. Jussien and Barichard (2000) note that while determining the right explanation for values eliminated by global constraints such as $\text{alldifferent}(A, B, C, ...)$ can be difficult, even “very simple explanations can still give very interesting results” (p. 6).

A contraction explanation points to constraint(s) that led to a failure, which means that the explanations in an explanation-based constraint system can be used to dynamically remove conflicting constraints in order to find partial solutions. There are two considerations when a constraint is removed: values that were eliminated by the removed constraint need to be restored and the effects of constraint propagation need to undone so that the solver’s state looks like the removed constraint had never existed. Jussien and Barichard (2000) explain...
how the propagation functions for each constraint can be extended to handle restored variable values. The stored eliminating explanations can be used to prevent the solver from thrashing in order to reduce the amount of work that needs to be performed to restore the state of the solver when a constraint is removed.

The difficulties in handling constraint removal with constraint propagation have also been considered in the realm of dynamic CSPs, CSPs where constraints can be dynamically added and removed (Bessière, 1991; Debruyne, 1996). Generally, adding constraints is easy, just a matter of constraint propagation, but removing them is difficult. As will be seen in chapter 10, the generic constraint solver Gecode (Gecode Team, 2006) has no mechanism for removing constraints—once a constraint has been added, there is no way to recover the previous state.

Explanation-based constraint programming has been implemented for the now-obsolete 1.2 version of the choco constraint programming library (CHOCO Team, 2010) for a number of variables and constraints with finite integer and real domains. There do not appear to be plans to update this for current versions of choco. For use in error detection in overconstrained XDG CSPs, the main drawback to explanation-based constraint programming is that it has not been implemented for set variables and set constraints. It would be non-trivial to extend the approach from finite domain constraints to finite set constraints. Because the domains of set variables can be so much larger (n!), the space needed to store eliminating explanations would become much larger and it is also likely that it would be challenging to extend the set constraint propagations functions to efficiently determine eliminating explanations.
6.2.2 Junker: QuickXplain

QuickXplain is a “non-intrusive conflict detection algorithm” that makes it possible to detect conflicting constraints in CSPs without modifying the constraint solver (Junker, 2001). This means that QuickXplain is able to work with any constraint propagation algorithms. It works by recursively partitioning the original CSP into smaller CSPs containing subsets of the original constraints and testing the consistency of the subsets until a minimal conflict can be found. Junker observes:

“Although the principle employed in QuickXplain is quite intuitive, this algorithm is coming as a surprise. We believed for a long time that conflict detection for constraint propagation either has to be intrusive and causing overhead or it has to be very slow.”

Previous iterative conflict detection approaches needed $O(n \times k)$ constraint checks to find a conflict of size $k$ in a set of $n$ constraints (de Siqueira and Puget, 1988; Bakker et al., 1993), whereas QuickXplain needs only $O(n \times \log(k + 1) + k^2)$, so QuickXplain is much faster if $k$ is small compared to $n$ (Junker, 2001).

Since the QuickXplain algorithm will be used by the Fledgling grammar checker presented in chapter 10, the algorithm will be presented in its entirety here so that the implementation in later chapters is clear. Figure 6.5 shows the algorithm as published in Junker (2001) with the addition of the explicit partial order parameter from Junker (2004).

QuickXplain takes three arguments: 1) the background constraints $C$, which will not be considered as part of the current conflict set, 2) the foreground constraints $U$, which can end up in the current conflict set, and 3) a partial order $\prec$ on all constraints $C \cup U$. The function CONSISTENT$(C)$ determines whether a CSP containing the constraints $C$
Algorithm 6.2.1: $\text{QUICKXPLAIN}(C, U, \prec)$

1. if not $\text{ISCONSISTENT}(C)$ then return $\emptyset$
2. if $U = \emptyset$ then throw exception ‘no conflict’
3. let $a_1, \ldots, a_n$ be an enumeration of $U$ that respects $\prec$
4. $k := 0$
5. while $\text{ISCONSISTENT}(C)$ and $k < n$
6. $C_k := C$
7. $k := k + 1$
8. $C := \Pi(C \cup \{a_k\})$
9. if $\text{ISCONSISTENT}(C)$ then throw exception ‘no conflict’
10. $X := \{a_k\}$
11. let $i$ be $\text{SPLIT}(k - 1)$
12. $U_1 := \{a_1, \ldots, a_i\}$
13. $U_2 := \{a_{i+1}, \ldots, a_{k-1}\}$
14. if $U_2 \neq \emptyset$ then
15. $C := C_i$
16. $C := \Pi(C \cup X)$
17. let $\Delta_2$ be the result of $\text{QUICKXPLAIN}(C, U_2, \prec)$
18. $X := \Delta_2 \cup X$
19. if $U_1 \neq \emptyset$ then
20. $C := C_0$
21. $C := \Pi(C \cup X)$
22. let $\Delta_1$ be the result of $\text{QUICKXPLAIN}(C, U_1, \prec)$
23. $X := \Delta_2 \cup X$
24. return $X$

Figure 6.5: $\text{QUICKXPLAIN}$ Algorithm from Junker (2001)
is satisfiable, the function $\text{SPLIT}(k)$ returns a value between 1 and $k$ for the recursive split, and the function $\Pi(C)$ performs constraint propagation as new constraints are added.

The loop in lines 4–8 performs a search for the first failed constraint in $C \cup U$. It starts with a CSP containing the background constraints $C$ and one by one adds the foreground constraints from $U$ until a conflict is found. The constraint that led to the failure, $a_k$, is then added to the current conflict set $X$. Next, the constraints $\{a_1, \ldots, a_{k-1}\}$ are divided into two partitions and $\text{QUICKXPLAIN}$ is called recursively on each half in lines 14–23 and to see if any other constraints are involved in the conflict with $a_k$. In the first recursive call, which considers the later constraints $U_2$, all the constraints from $U_1$ and constraint $a_k$ are put the background $C$ before $\text{QUICKXPLAIN}$ is called. In the second recursive call for the earlier constraints $U_1$, $a_k$ and any constraints just found in line 18 are added to the background. This ensures that $\text{QUICKXPLAIN}$ finds a single conflict set that is centered around $a_k$ and that it prefers conflict sets that contain constraints found earlier in the partial order $\prec$. If $\prec$ is a total order, there is a unique preferred conflict. If any additional constraints are found (i.e., $a_k$ is not responsible for the failure alone but a combination of multiple constraints led to the conflict), they are added to the conflict set $X$ and the algorithm completes by returning $X$.

### 6.3 Usability for Overconstrained XDG CSPs

As described in the introduction to this chapter, XDG CSPs for ungrammatical input sentences will be overconstrained and thus have no solutions, and the CSP solver used by the XDG Grammar Development Kit (XDK, Debusmann et al., 2004) returns a failure without
giving any further information about what led to the failure. It is necessary to determine which constraints cause a CSP to be overconstrained and to provide feedback about the type of error. Relaxing those constraint makes it possible to find a partial solution.

XDG CSPs, which will be described in more detail in chapter 10, contain thousands of variables and constraints, many of which involve set variables with large domain sizes. The size and nature of these CSPs makes constraint propagation and distribution indispensable, which means that the algorithms presented in section 6.1 that are centered around traditional chronological backtracking are not well-suited to finding partial solutions in a reasonable amount of time. Even relying on a default constraint distribution approach rather than XDG’s custom distribution leads to unacceptably long solving times.

Turning from partial constraint satisfaction algorithms to conflict detection algorithms, the motivations behind explanation-based constraint programming—recording explanations in order to provide feedback to developers and users of CSP-based applications—align quite well with the XDG grammar checking task, however the algorithm requires non-trivial extensions to propagation for every single constraint type and the existing implementation handles only a limited number of constraints for boolean, integer, and real variables. Unfortunately, the algorithm has not been extended to set variables and constraints and is therefore not applicable for XDG CSPs.

On the other hand, QUICKXPLAIN’s non-intrusive approach makes it well-suited for use with XDG CSPs, because it is simple to implement QUICKXPLAIN using a modern constraint solver such as Gecode (Gecode Team, 2006). As will be explored in chapter 10,

\[^{15}\text{This is not only a problem for grammar checking. During grammar development, errors in a principle definition or lexical entry result in a failure message with no further feedback.}\]
QUICKXPLAIN’s use of background (hard) and foreground (soft) constraints make it possible to focus the conflict detection algorithm on targeted constraints that can be directly tied to grammatical error types. Additionally, the constraint ordering parameter can be used to prefer certain types of conflicts over others. QUICKXPLAIN is also quite fast for XDG CSPs, since the number of conflicting constraints is quite small compared to the total number of constraints.

6.4 Summary

Partial constraint satisfaction and CSP conflict detection algorithms provide the information needed by an XDG-based grammar checker for a detailed error diagnosis. Information about conflicting constraints can be used to pinpoint the type of grammatical error(s) made in the input sentence and relaxing these constraints can provide partial analyses that enrich the feedback provided to a learner.

Partial constraint satisfaction algorithms (Freuder and Wallace, 1992) and explanation-based constraint programming (Jussien et al., 2000; Jussien and Barichard, 2000; Jussien, 2001) can fulfill some of these needs, however they are not able to handle XDG CSPs, which in particular contain large numbers of set constraints. The non-intrusive conflict detection algorithm QUICKXPLAIN (Junker, 2001, 2004) is able to work with modern constraint solvers to efficiently find conflicting constraints in XDG CSPs.
Part II

Error-Annnotated Learner Corpus
Chapter 7: EAGLE Corpus

This chapter describes the Error-Annotated German Learner Corpus (EAGLE), the first corpus of beginning learner German with grammatical error annotation.\textsuperscript{16} The only available error-annotated corpus of learner German, FALKO Siemen et al. (2006), focuses on advanced learners, and EAGLE has been created to complement this with a corpus of beginning learner data. In particular, EAGLE is expected to contain a wider range of syntactic errors such as agreement and word order errors.

The corpus uses an error annotation format that extends the multi-layer standoff format proposed by Lüdeling et al. (2005) to include incremental target hypotheses for each error. In this format, each annotated error includes information about the location of tokens affected by the error, the error type, and the proposed target correction. The multi-layer standoff format makes it possible to annotate ambiguous errors with more than one target correction and to annotate the multiple, overlapping errors common in beginning learner productions.

The data collected for the corpus is described in section 7.2. The EAGLE error annotation scheme for grammatical errors and the multi-layer standoff format with incremental target hypotheses are introduced in section 7.3. Section 7.4 presents the interannotator agreement.

\textsuperscript{16}This chapter is adapted and extended from Boyd (2010a).
for the essay subcorpus and section 7.5 describes the division into training and testing data used for the grammar development and testing in chapters 9–10.

7.1 Motivation for Collecting Learner Data

Corpora of learner language provide useful data for research in language acquisition and the development of natural language technology. Learner productions provide insight into the language acquisition process and annotated learner corpora allow researchers to easily search for particular phenomena. Annotated data is also useful for developing and customizing tools such as part-of-speech taggers, spell checkers, and grammar checkers for non-native speakers (cf. Granger, 2003; Meurers, 2009). For this work, data annotated with grammatical errors is required for the evaluation of the grammar checking approach presented in Chapter 10.

7.2 Data

The learner language data in the EAGLE corpus consists of responses to course-related activities from students in the second and third courses of The Ohio State University’s introductory German sequence. Two main types of data were collected: online workbook responses and hand-written final exam essays. The two types of data were chosen to include both typed and hand-written language produced with and without access to reference materials.
Translate into German:

To whom do these articles of clothing belong?

Sample responses:

Wem hat diesen Kleidungsstücke
Wer gehört diese Kleidungen?
Wem gehören diese Kleidungsstücke?
Wem gehört diesem Kleidungs?
Wem gehört die Kleidungsstücke?
Wer gehören diese Kleidungsstucke?
Wem gehört diesem Kleidungstücke?
Wem gehören dieser Kleidungsstücke zu?
Wem gehören diese Kleidungsstücke?
Wem gehören dieser Kleidungsstüke?
Wem gehören diesem Kleidungsstücke?
Wem gehören diese Artikel der Kleidung?

Figure 7.1: Sample Exercise from Online Workbook

7.2.1 Online Workbook Data

The online workbook subcorpus contains data collected from the Deutsch: Na Klar! Online Workbook, 4th Edition (Briggs, 2003). Responses were collected from 50 learners (38 in the second course and 12 in the third course) during Spring Quarter 2007 at The Ohio State University. The online workbook contains a wide variety of activities including translation exercises, cloze questions, and build-a-sentence questions, etc. which the learners completed outside of class with access to reference materials. A translation exercise with sample learner responses is shown in Figure 7.1.
The online workbook responses range from answers to multiple choice questions to short essays. In order to focus on data suited for grammatical error annotation, the EAGLE corpus contains responses to only those activities where the learners are instructed to respond in complete sentences. In the activities where responses were automatically assessed by the online workbook, students often made multiple submissions until they reached the target answer. Each of these responses is stored separately in the corpus with a code indicating the order in which the responses were submitted.

In total, there are 59,068 tokens in 6,986 responses to 412 activities. When duplicate responses to the same activity are removed (since many students arrived at the same target answer for a given activity), there are approximately 33,000 tokens in 3,500 responses containing a total of 7,500 sentences.

7.2.2 Essay Data

The essay subcorpus contains hand-written essays from 81 learners (43 in the second course and 38 in the third course) collected during Fall Quarter 2006 at Ohio State University. The learners were given one topic in the second course and could choose from two topics in the third course. The essays were written as part of a timed exam without access to reference materials. The hand-written data was keyed in and the subcorpus contains 12,412 tokens in 81 essays with an average of 16 sentences per essay.

Due to the anonymous data collection, it is not possible to determine whether any of the same learners appear in both the workbook and essay subcorpora, but it is unlikely that the two learner groups overlap.
7.2.3 Preprocessing

The collected data was tokenized using Stefanie Dipper’s German tokenizer (Dipper, 2008) and then anonymized to remove all potentially identifying personal names, streets, cities, and states. In order to maintain coherence in longer responses, each anonymized item receives a code such as “CITY-4” or “FIRSTNAME-13” that is used consistently throughout the corpus.

7.3 Error Annotation

The EAGLE error typology and annotation format focus on the annotation of grammatical errors present in the learner data. Before the grammatical error annotation begins, non-word spelling errors are corrected as described in section 7.3.1. Then, the grammatical error typology described in section 7.3.2 is applied using the multi-layer standoff format described in section 7.3.3.

Each sentence in the corpus is annotated independently without regard to context. If there is no context in which the sentence could be uttered, a series of one or more corrections is annotated that transforms the ungrammatical sentence into a grammatical one. Each correction includes information about which tokens are affected by the error, the type of error, and the proposed target correction.

7.3.1 Non-Word Spelling Errors

Non-word spelling errors were identified with the assistance from Aspell (Aspell, 2008) and corrected to either a word with the smallest edit distance or to a literal translation in
the case of English or other foreign words. A sample of the types of spelling errors found in EAGLE is shown in Table 7.2.

The spelling errors in the corpus have been classified using the spelling error annotation scheme CLASSY proposed in Rimrott (2005). CLASSY focuses on non-word spelling errors with one spelling error such as a morphological error or capitalization error. Rimrott created a corpus of spelling errors produced by learners using the E-Tutor (Heift, 2010, see section 3.2.7). Because Rimrott found that only 3.6% of her spelling errors contained multiple errors, the CLASSY system and her further analysis ignore cases of multiple errors.

The EAGLE corpus has a slightly higher percentage of spelling errors with multiple errors than Rimrott’s corpus, with 5.8% of spelling errors containing multiple errors. This is most likely due to the fact that some of the activities used for the EAGLE corpus collection are less constrained (e.g., longer essays) than the activities used in Rimrott’s corpus, which were tasks to build a sentence from lemmas or translate a sentence.

CLASSY classifies spelling errors by whether they appear to be competence (I) or performance (II) errors and then competence errors are classified in three further dimensions:
• Linguistic subsystem:
  o O: orthographic
  o P: phonological
  o M: morphological
  o L: lexical

• Language influence:
  o N: paralingual (neither English nor German)
  o B: ambilingual (both English and German)
  o G: intralingual (only German)
  o E: interlingual (only English)

• Edit distance:
  o 1: single violation
  o 2: multiple violation

For the classification of EAGLE spelling errors, a third performance-related category activity design (III) was added to the top-level classification to account for a type of error that was caused by the activity in the Quia online workbook: in several activities, students were asked to read hand-written words such as names of unfamiliar professions from low-quality images and it was not uncommon for students to make spelling errors. This is in effect a type of performance error, but is more closely related to activity design rather than the student’s competence or performance in German, so it is treated separately from typical performance errors.
In addition, we noticed that many errors that the CLASSY scheme classified as performance errors due to frequency cutoffs appeared to be competence errors in EAGLE. To adjust for this difference, we lowered one of the frequency cutoffs that determines whether a frequent error is classified as a competence error. Instead of three occurrences from one student as in the original CLASSY scheme, two occurrences by one student are sufficient to classify an error as a competence error.

The EAGLE spelling errors were classified by two annotators using the CLASSY scheme. The two annotators began the spelling error annotation process by jointly annotating a random sample of approximately 5% of the errors in order to become familiar with the CLASSY scheme. The remaining errors were classified independently and an analysis of interannotator agreement can be found below. The annotators discussed any disagreements to produce a final classification for each item.

The distribution of spelling errors from Rimrott’s corpus and EAGLE is shown Figure 7.1. The most striking differences between Rimrott’s corpus and EAGLE are in the much higher occurrences in EAGLE of errors IOE1 (competence, orthographic, interlingual, single edit) and IPE2 (competence, phonological, interlingual, multiple edits) in EAGLE. The IOE1 errors in the EAGLE are primarily capitalization errors where the learners failed to capitalize German nouns and it is unclear why Rimrott’s learners and the EAGLE learners would differ so drastically in this regard. It can potentially be attributed to different teaching methods. The higher rate of IPE2 errors in EAGLE is likely due to differences in the types of activities used when gathering the two corpora. In E-Tutor, all activities have a target

\footnote{Many thanks to Marion Zepf for her assistance with the spelling error annotation.}
answer with provided vocabulary. EAGLE contains a higher number of less restricted activity types, in particular short essays in the online workbook and the exam essay questions where the student’s vocabulary choices are not constrained at all. For exam essays, the lack of reference materials also contributes to a higher number of misspellings.

**Interannotator Agreement for CLASSY**

Interannotator agreement was calculated for the subset of the corpus annotated independently while ignoring the activity design (III) errors, resulting in a total of 977 type errors. The edit distance feature can be calculated automatically, so the evaluation considers 17 error classes: ILB, ILE, ILG, ILN, IMB, IME, IMG, IMN, IOB, IOE, IOG, ION, IPB, IPE, IPG, IPN, and II.

If each of the 17 classes is treated independently, Cohen’s $\kappa$ is 0.574 and Krippendorf’s $\alpha$ is 0.572, which indicate only a moderate level of agreement. Artstein and Poesio (2009) suggest that $\alpha$ above 0.7 or 0.8 is required for good-quality annotation, so it appears that improvement in the CLASSY annotation guidelines or further annotator training is needed in order to achieve a high level of agreement. If partial credit is given for the competence subclasses such that each component (competence, linguistic subsystem, and language influence) is given equal weight, $\kappa$ is 0.783 and $\alpha$ is 0.582. The linguistic subsystem subclassification appears to cause the most disagreement; if the linguistic subsystem is ignored, $\kappa$ is 0.839 and $\alpha$ is 0.685. Examining only the competence vs. performance distinction, $\kappa$ is 0.861 and $\alpha$ is 0.687. These results indicate moderate to acceptable agreement between the two annotators.

19The vocabulary may be provided in German or in English, which is used for translation exercises.
<table>
<thead>
<tr>
<th>Error Type</th>
<th>Rimrott Count</th>
<th>Rimrott %</th>
<th>EAGLE Count</th>
<th>EAGLE %</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILB1</td>
<td>6</td>
<td>0.006</td>
<td>2</td>
<td>0.001</td>
</tr>
<tr>
<td>ILB2</td>
<td>7</td>
<td>0.007</td>
<td>11</td>
<td>0.007</td>
</tr>
<tr>
<td>ILE1</td>
<td>8</td>
<td>0.008</td>
<td>53</td>
<td>0.033</td>
</tr>
<tr>
<td>ILE2</td>
<td>17</td>
<td>0.017</td>
<td>70</td>
<td>0.043</td>
</tr>
<tr>
<td>ILG1</td>
<td>5</td>
<td>0.005</td>
<td>22</td>
<td>0.014</td>
</tr>
<tr>
<td>ILG2</td>
<td>3</td>
<td>0.003</td>
<td>15</td>
<td>0.009</td>
</tr>
<tr>
<td>ILN1</td>
<td>88</td>
<td>0.086</td>
<td>117</td>
<td>0.073</td>
</tr>
<tr>
<td>ILN2</td>
<td>61</td>
<td>0.059</td>
<td>30</td>
<td>0.019</td>
</tr>
<tr>
<td>IMB1</td>
<td>11</td>
<td>0.011</td>
<td>30</td>
<td>0.019</td>
</tr>
<tr>
<td>IMB2</td>
<td>2</td>
<td>0.002</td>
<td>5</td>
<td>0.003</td>
</tr>
<tr>
<td>IME2</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0.002</td>
</tr>
<tr>
<td>IMG1</td>
<td>143</td>
<td>0.139</td>
<td>116</td>
<td>0.072</td>
</tr>
<tr>
<td>IMG2</td>
<td>151</td>
<td>0.147</td>
<td>81</td>
<td>0.050</td>
</tr>
<tr>
<td>IMN1</td>
<td>1</td>
<td>0.001</td>
<td>2</td>
<td>0.001</td>
</tr>
<tr>
<td>IMN2</td>
<td>2</td>
<td>0.002</td>
<td>1</td>
<td>0.001</td>
</tr>
<tr>
<td>IOB1</td>
<td>2</td>
<td>0.002</td>
<td>6</td>
<td>0.004</td>
</tr>
<tr>
<td>IOE1</td>
<td>1</td>
<td>0.001</td>
<td>222</td>
<td>0.138</td>
</tr>
<tr>
<td>IOE2</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0.004</td>
</tr>
<tr>
<td>IOG1</td>
<td>2</td>
<td>0.002</td>
<td>4</td>
<td>0.002</td>
</tr>
<tr>
<td>IOG2</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0.006</td>
</tr>
<tr>
<td>ION1</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>0.012</td>
</tr>
<tr>
<td>ION2</td>
<td>1</td>
<td>0.001</td>
<td>1</td>
<td>0.001</td>
</tr>
<tr>
<td>IPB1</td>
<td>36</td>
<td>0.035</td>
<td>95</td>
<td>0.059</td>
</tr>
<tr>
<td>IPB2</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0.005</td>
</tr>
<tr>
<td>IPE1</td>
<td>91</td>
<td>0.089</td>
<td>80</td>
<td>0.050</td>
</tr>
<tr>
<td>IPE2</td>
<td>2</td>
<td>0.002</td>
<td>104</td>
<td>0.064</td>
</tr>
<tr>
<td>IPG1</td>
<td>68</td>
<td>0.066</td>
<td>180</td>
<td>0.112</td>
</tr>
<tr>
<td>IPG2</td>
<td>10</td>
<td>0.010</td>
<td>34</td>
<td>0.021</td>
</tr>
<tr>
<td>IPN1</td>
<td>15</td>
<td>0.015</td>
<td>21</td>
<td>0.013</td>
</tr>
<tr>
<td>IPN2</td>
<td>2</td>
<td>0.002</td>
<td>2</td>
<td>0.001</td>
</tr>
<tr>
<td>II1</td>
<td>278</td>
<td>0.271</td>
<td>177</td>
<td>0.110</td>
</tr>
<tr>
<td>II2</td>
<td>14</td>
<td>0.014</td>
<td>67</td>
<td>0.042</td>
</tr>
<tr>
<td>III1</td>
<td>–</td>
<td>–</td>
<td>11</td>
<td>0.007</td>
</tr>
<tr>
<td>III2</td>
<td>–</td>
<td>–</td>
<td>7</td>
<td>0.004</td>
</tr>
<tr>
<td>Total</td>
<td>1027</td>
<td></td>
<td>1613</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.1: Classification of Non-Word Spelling Errors in EAGLE
<table>
<thead>
<tr>
<th>Error Type</th>
<th>Example</th>
<th>Detailed Error Description</th>
</tr>
</thead>
</table>
| Word Form        | Ja, Ich zeige ihn ihnen.  
yes, I show him them  
Target: Ja, ich zeige ihn ihnen. | Capitalization                              |
| Selection        | Hast du der Reiseprospekt<sub>nominative</sub>?  
have you the travel brochure<sub>nominative</sub>  
Target: Hast du den Reiseprospekt? | Verb - NP Complement Case                   |
| Agreement        | Du arbeiten in Liechtenstein.  
you work<sub>1st/3rd plural</sub> in Liechtenstein  
Target: Du arbeitet in Liechtenstein. | Subject-Verb Agreement                       |
| Word Order       | Welcher Job diese Dinge verlangen würde?  
which job these things require would  
Target: Welcher Job würde diese Dinge verlangen? | Finite Verb Position                        |
| Punctuation      | Gehört dir diese Jacke  
belongs you this jacket  
Target: Gehört dir diese Jacke? | Missing Sentence-Final Punctuation          |

Figure 7.3: Types of Errors Annotated

### 7.3.2 Error Typology

The EAGLE error typology was designed for errors made by beginning learners of German and was informed by two previous classification schemes from Rogers (1984) and Juozulynas (1994), who addressed errors by advanced and intermediate learners of German respectively. The error typology includes five main types of errors: selection, agreement, word order, word form (errors within single words that are not non-word spelling errors), and punctuation. Examples of each type of error are shown in Figure 7.3.

Detailed error annotation schemes for these types are shown in Figures 7.4–7.8. Each type of error is subcategorized by grammatical features of the words, phrases, or topological fields (Höhle, 1986a, see section 2.1.6) affected by the error. The complete error typology
is explained in detail with examples for each error type in the EAGLE annotation manual Boyd (2010b).

Selection

The selection error typology uses the part-of-speech of the head to organize the error types. Heads such as verbs and prepositions select arguments and the listed error types cover the ways in which an argument can deviate from the expected form.

There are two exceptions to the head-based categorization, the determiner error S7, which handles determiners with no head nouns, and the sentence error S5, which checks responses top-down. S5 looks for the presence of a main clause and main finite verbs in each response. S5.C is used when the response is so unclear that the annotator is unable to propose a target correction.

Agreement

Agreement errors are categorized by the type of agreement under consideration: subject-verb, determiner-adjective-noun, etc. More than one type of agreement can be violated in the same error, so the sentence *ich seht ihn* ‘I see₂,₃pl him’ would contain a single subject-verb agreement error in person and number annotated as *A₁.A₁.B*.

Word Order Errors

Word order errors are subcategorized by topological fields for clause-level word order errors and by phrase type for phrase-level errors. Given the similarities between English and
S1. Verb
   A. Complement
      i. NP complement - incorrect case
      ii. PP complement - incorrect preposition
      iii. PP complement - incorrect case with correct preposition
      iv. Two-way PP complement with verb of state/location - incorrect preposition or case
      v. Two-way PP complement with verb of motion - incorrect preposition or case
         a. NP-dependent preposition choice (e.g., in die Schweiz)
      vi. VP complement - haben/sein error
      vii. VP complement - incorrect non-finite verb form
      viii. VP zu infinitive complement - zu missing
      ix. VP clausal complement - incorrect complementizer
      x. Incorrect correlative
      xi. Incorrect complement type
      xii. Complement should be adverbial
      xiii. Missing
      xiv. Extra
   B. Separable prefix - impossible form
   C. Reflexive
      i. Missing
      ii. Extra
      iii. Incorrect case

S2. Preposition
   A. Complement
      i. Incorrect case
      ii. Missing
   B. Missing
      i. In paired construction (e.g., von/bis)

Figure 7.4: Selection Error Typology
S3. Noun

A. Determiner
   i. Missing
   ii. Extra (e.g. die die)
   iii. Extra with job/description or with als
   iv. Pronoun instead of determiner (alles, wem)
   v. Pronominalized determiner instead of determiner (eins)
   vi. Adverb instead of determiner (sehr instead of viel)

B. Complement
   i. NP complement - incorrect case
   ii. PP complement - incorrect preposition
   iii. PP complement - incorrect case with correct preposition
   iv. Incorrect complement type

C. Modifier
   i. Genitive NP modifier - incorrect case

S4. Adjective

A. Complement
   i. NP complement - incorrect case
   ii. PP complement - incorrect preposition
   iii. PP complement - incorrect case with correct preposition
   iv. Incorrect complement type

B. Comparative clause
   i. Incorrect type (wie/als or wie/als missing)
   ii. Non-parallel after wie/als

Figure 7.4: Selection Error Typology (cont.)
S5. Sentence

A. Main clause
   i. Missing
   ii. Extra coordinating conjunction
B. Finite verb
   i. Missing in main clause
   ii. Missing in subordinate clause
   iii. Extra in main clause
   iv. Imperative form with subject
   v. Infinitive with subject (only sein)
C. Target unclear
   i. Main clause
   ii. Subordinate clause
D. Subordinate clause (not a verb complement)
   i. Incorrect subordinating conjunction
   ii. Missing subordinating element

S6. Conjunction

A. Coordinating
   i. Missing coordinating conjunction
   ii. Mismatched determiner or adjective
   iii. Missing phrase in coordination
   iv. Unlike coordinated phrases
B. Subordinating
   i. Incorrect tense

S7. Determiner/Inflected Adjective

A. Missing head noun
B. Relative pronoun with no antecedent

Figure 7.4: Selection Error Typology (cont.)
<table>
<thead>
<tr>
<th>A1. Subject-Verb</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Person</td>
<td></td>
</tr>
<tr>
<td>B. Number</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A2. Determiner-Adjective-Noun</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Gender</td>
<td></td>
</tr>
<tr>
<td>i. Of gendered noun (e.g., Freundin)</td>
<td></td>
</tr>
<tr>
<td>B. Number</td>
<td></td>
</tr>
<tr>
<td>C. Case</td>
<td></td>
</tr>
<tr>
<td>D. Definiteness</td>
<td></td>
</tr>
<tr>
<td>E. Attributeness</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A3. Relative Pronoun-Antecedent</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Gender</td>
<td></td>
</tr>
<tr>
<td>B. Number</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A4. Negative</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Double negative</td>
<td></td>
</tr>
<tr>
<td>B. Incompatible (schon with noch nicht)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A5. Subject-Predicate with Copula</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Number</td>
<td></td>
</tr>
</tbody>
</table>

| A6. Reflexive-Subject |  |

<table>
<thead>
<tr>
<th>A7. Appositives</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Gender</td>
<td></td>
</tr>
<tr>
<td>B. Number</td>
<td></td>
</tr>
<tr>
<td>C. Case</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.5: Agreement Error Typology
O1. Finite verb
   A. In a main clause
   B. In a subordinate clause

O2. Non-finite verb
   A. Reordered outside clause

O3. Separable prefix

O4. *Mittelfeld*
   A. Arguments
   B. Adjuncts

O5. *Vorfeld*
   A. Contains interrogative pronoun in question

O6. Prepositional phrase
   A. Involving *was für*

O7. Noun phrase
   A. *Kein . . . mehr*

O8. Adverb phrase
   A. *Noch nicht*

Figure 7.6: Word Order Error Typology
<table>
<thead>
<tr>
<th>Topic</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1. Capitalization</td>
<td></td>
</tr>
<tr>
<td>W2. Inflection</td>
<td></td>
</tr>
<tr>
<td>A. Noun</td>
<td></td>
</tr>
<tr>
<td>B. Verb</td>
<td></td>
</tr>
<tr>
<td>C. Adjective</td>
<td></td>
</tr>
<tr>
<td>W3. Single-letter real word spelling error with different POS</td>
<td></td>
</tr>
<tr>
<td>A. <em>regen</em> for <em>regnen</em></td>
<td></td>
</tr>
<tr>
<td>B. <em>such</em> for <em>sich</em></td>
<td></td>
</tr>
<tr>
<td>C. <em>dass</em> for <em>das</em></td>
<td></td>
</tr>
<tr>
<td>D. <em>Hat</em> for <em>Hut</em></td>
<td></td>
</tr>
<tr>
<td>E. . . .</td>
<td></td>
</tr>
<tr>
<td>W4. Collocations</td>
<td></td>
</tr>
<tr>
<td>A. Dates (e.g., <em>im Montag</em>)</td>
<td></td>
</tr>
<tr>
<td>B. Descriptions (e.g., <em>von Leder</em>)</td>
<td></td>
</tr>
<tr>
<td>C. Locations (e.g., <em>nach for bei</em>)</td>
<td></td>
</tr>
<tr>
<td>D. Transitions (e.g., <em>dafür for deswegen</em>)</td>
<td></td>
</tr>
<tr>
<td>E. Sport (e.g., <em>Schlittschuh gehen</em>)</td>
<td></td>
</tr>
<tr>
<td>W5. Dictionary errors</td>
<td></td>
</tr>
<tr>
<td>A. <em>erste</em> for <em>zuerst</em></td>
<td></td>
</tr>
<tr>
<td>B. <em>Marke neu</em> for <em>nagelneu</em></td>
<td></td>
</tr>
<tr>
<td>C. <em>äußerste</em> for <em>draußen</em></td>
<td></td>
</tr>
<tr>
<td>D. <em>nach</em> for <em>nachdem</em></td>
<td></td>
</tr>
<tr>
<td>E. . . .</td>
<td></td>
</tr>
<tr>
<td>W6. Miscellaneous errors</td>
<td></td>
</tr>
<tr>
<td>A. Doubled word</td>
<td></td>
</tr>
<tr>
<td>B. <em>nicht</em> vs. <em>keine</em></td>
<td></td>
</tr>
<tr>
<td>C. Extra punctuation (copy-paste from activity)</td>
<td></td>
</tr>
<tr>
<td>D. Extra space between separable prefix and verb-final verb</td>
<td></td>
</tr>
<tr>
<td>E. Compound formed with space</td>
<td></td>
</tr>
<tr>
<td>F. Numbers formed with space</td>
<td></td>
</tr>
<tr>
<td>G. Adjectives derived from place names formed incorrectly</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.7: Abbreviated Word Form Error Typology
P1. Comma
   A. Missing comma
      i. Between coordinated elements
      ii. Between main and subordinate clauses or other clauses
          where a comma is required
      iii. Other
   B. Extra comma
      i. Between coordinated elements
      ii. Other

P2. Other punctuation
   A. Missing
   B. Extra

P3. Sentence final punctuation
   A. Incorrect
   B. Missing
   C. Extra

Figure 7.8: Punctuation Error Typology
German word order for prepositional phrases, noun phrases, and adverb phrases, the EAGLE typology proposed at the beginning of the annotation process contained only clause-level error categories. A small number of phrase-level errors were indeed found in the corpus and the phrase-level categories were subsequently added.

**Word Form Errors**

Word form errors include capitalization errors and other errors centered on single words that are difficult or awkward to describe as selection or agreement errors. For example, the response in (24) shows a case where the student (among other errors) has typed *such* instead of *sich* ‘herself’. This appears to be a keyboarding error given the proximity of *I* and *U*, however the linguistically-based EAGLE typology would identify *such* as the imperative of the verb *suchen* ‘search’. With this approach, a likely keyboarding error would be annotated as two errors: an extra imperative verb plus a missing reflexive pronoun. To avoid this, the category W3 was created for real word spelling errors that differ by a single letter and result in a word with a different POS. The idea behind the error category W3 is that these errors could potentially be identified using a spell checking approach rather than a full-fledged grammar checker.

(24) * Sie will **such** nicht die Stelle **bewerben**.
    she wants₃,sg search₃,imp not the position apply
    ‘She does not want to apply for the position.’

The word form error typology also contains collocation errors, errors that appear to be due to dictionary lookup (words that are homonyms in English but not in German), and miscellaneous errors. The word form category may be expanded by annotators as needed.
to cover new errors, but the goal of the EAGLE annotation scheme is to treat the word form category as a last resort for errors that cannot be described using the other error categories.

**Punctuation Errors**

The punctuation error typology is not intended to be exhaustive and focuses on two types of errors: sentence-final punctuation errors and comma errors. EAGLE contains only responses to activities where the response should contain one or more complete sentences, so it is assumed that all responses should contain sentence-final punctuation. English and German differ noticeably in the presence of commas between subclauses, so comma errors are also expected to be frequent.

**Addition to Error Codes**

There is one addition to the error codes: * indicates that the head was corrected instead of the dependent in selection errors. It is mainly used for haben/sein errors (S1.A.vi.). Example (25) shows a haben vs. sein error that requires the syntactic head (the finite auxiliary) to be corrected rather than the main non-finite verb.

(25) * Ich bin eine Fahrkarte gekauft.
     I am a ticket bought
     ‘I am (correct: have) bought a ticket.’

**7.3.3 Error Annotation Format**

The EAGLE grammatical error annotation uses a multi-layer standoff format first proposed for learner error annotation by Lüdeling et al. (2005) for the FALKO corpus of advanced
learner German (Siemen et al., 2006). This format is chosen in order to account for situations where a) errors span multiple words, b) learners make multiple overlapping errors in a single sentence, and c) errors are ambiguous. Standoff annotation allows multiple overlapping errors to be annotated easily and multiple layers allow for multiple target corrections to be specified in the case of ambiguities.

As in Lüdeling et al. (2005), each type of error encompasses three layers in the annotation: location, description, and target. The location layer identifies which words, phrases, or clauses are affected, the description layer specifies the particular type of error such as a subject-verb agreement error, and the target layer gives the target correction that corresponds to the error description. The target correction makes explicit the annotator’s hypothesis about the learner’s intended utterance and shows the correction for the specified error.

Example (26) shows a sentence with multiple errors that will be used in the following sections to illustrate the annotation format. It contains a noun phrase *dieses Hunden* ‘this dog’ where the determiner and the noun disagree in gender, case, and number and a verb complement *Wen* ‘whom’ in the wrong case. First, the agreement error will be considered. Figure 7.9 shows the appearance of the standoff annotation layers for agreement errors in this example.

(26) *Wen gehört dieses Hunden?*

*Wen* acc belong 3 sg *dieses neut,nom,acc,sg Hunden* masc,dat,pl

‘Whom does this dogs belong?’

156
If we want to annotate the agreement error in *dieses Hunden*, we identify the affected tokens, determine the type of error, and give a target correction as in Figure 7.10 below.

<table>
<thead>
<tr>
<th>Tokens</th>
<th>Wen</th>
<th>gehört</th>
<th>dieses</th>
<th>Hunden?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td></td>
<td></td>
<td></td>
<td>l</td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td>Det-Adj-Noun Agreement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target</td>
<td></td>
<td></td>
<td>dieser Hund</td>
<td></td>
</tr>
</tbody>
</table>

The top-most layers provide information about the original tokens, automatic tokenization, and automatic spelling correction. If there are errors in the tokenization or spelling correction, the annotator can specify corrections in the “Adj” (adjusted) spelling and tokenization layers. For each top-level error type, there is a set of three layers in the annotation that correspond to the location of the error, description of the error type, and target correction specified by the annotation. The current annotation uses the layers shown in Table 7.2.

### 7.3.4 Annotation Issues

Performing grammatical error annotation is a complicated task involving the selection of an error span, an error description type, and a target correction that reflects the annotated
<table>
<thead>
<tr>
<th>Annotation Layer(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token</td>
<td>anonymized tokens</td>
</tr>
<tr>
<td>Token Notes</td>
<td>notes about tokens for additional annotation of proper nouns such as TITLE or other non-word spelling error related issues such as accidental run-ons</td>
</tr>
<tr>
<td>Tokenizer</td>
<td>information about token type from the tokenizer</td>
</tr>
<tr>
<td>Norm Spelling</td>
<td>automatic spelling corrections</td>
</tr>
<tr>
<td>Adj Tokenizer</td>
<td>hand corrections to the tokenization</td>
</tr>
<tr>
<td>Adj Norm Spelling</td>
<td>hand corrections to the spelling correction</td>
</tr>
<tr>
<td>WF Loc, WF Desc, WF Target</td>
<td>word form errors</td>
</tr>
<tr>
<td>Sub Loc, Sub Desc, Sub Target</td>
<td>selection errors</td>
</tr>
<tr>
<td>Agr Loc, Agr Desc 1, Agr Target 1, Agr Desc 2, Agr Target 2</td>
<td>agreement errors with two extra description/target layers for ambiguous errors</td>
</tr>
<tr>
<td>Order Main Loc, Order Main Desc, Order Main Target</td>
<td>verb-related order errors in the main clause</td>
</tr>
<tr>
<td>Order Sub1 Loc, Order Sub1 Desc, Order Sub1 Target</td>
<td>verb-related order errors in the first-level subordinate clause (further embedding would require Order Sub2 Loc, etc., but this wasn’t necessary for the EAGLE data)</td>
</tr>
<tr>
<td>Order NV1 Loc, Order NV1 Desc, Order NV1 Target</td>
<td>non-verb-related order errors such as errors within NPs or the the Mittelfeld</td>
</tr>
<tr>
<td>Punc Loc, Punc Desc, Punc Target</td>
<td>punctuation errors</td>
</tr>
</tbody>
</table>

Table 7.2: Annotation Layers in EAGLE
error type. The following sections discuss annotation difficulties and the EAGLE annotation approach for: choosing the span of an error, choosing target corrections, performing incremental analysis, dealing with ambiguous errors, and annotating overlapping errors.

**Error Spans**

Selecting the tokens involved in an error can lead to unnecessary disagreement between annotators. For instance, when guidelines are not specific, one annotator might select only the determiner in a case agreement error while a second annotator might select both the determiner and the noun. Word order errors are particularly susceptible to disagreement: when two phrases appear before the finite verb instead of one, an annotator could select: 1) one of the phrases that appears before the finite verb, 2) the finite verb, or 3) both the phrase and the finite verb, since one target correction would exchange their positions in the sentence.

In order to reduce these kinds of disagreements in the selection of the tokens affected by an error, the span annotated for an error is determined by the error type and specified explicitly in the EAGLE annotation manual Boyd (2010b). The motivation is to have the annotators rely on well-defined linguistic units—single tokens, entire phrases, or entire topological fields—instead of expecting them to consistently select subsets of tokens that do not form linguistically coherent units.

- Word Form: a single token or a span of tokens (in the case of a tokenization or collocation error) is selected.
- Selection: the head and each token in the argument phrase is selected.
• Agreement: the entire affected phrase is selected.

• Word Order: the entire affected phrase (for an error within non-verb phrase) or the entire affected topological field is selected.
  
  – Non-Mittelfeld Clause-Level Errors: the whole clause, Vorfeld or verbal complex is selected.

  – Mittelfeld Errors: the entire Mittelfeld is selected.

• Punctuation: each token is selected individually.

When selecting the span of an error involving a noun phrase, the entire noun phrase including any modifying appositives or relative clauses is selected. For agreement errors involving a phrase and a single token, such as subject-verb agreement errors or relative pronoun-antecedent agreement errors, the single token (finite verb, relative pronoun) is selected along with the entire phrase that is not in agreement.

Target Corrections

EAGLE annotators choose target corrections for words, phrases, and topological fields, proposing incremental corrections rather than choosing a target correction for the whole sentence in one step. For most error types, the annotation focuses on linguistic properties of the lexical items present in the data. Annotation proceeds bottom-up by considering the properties of words and relations between them. For instance, determiner-adjective-noun agreement is examined whenever a noun phrase with a determiner or adjective is found in a response; if a sentence does not contain any such noun phrases, there is no need to consider determiner-adjective-noun agreement. The exceptions to this are the word order errors that
examine the positions of topological fields in each clause in a top-down fashion and the “Sentence” selection error, which also checks top-down for the presence of main clauses and finite verbs in each sentence.

The target corrections follow the following guidelines:

- Target corrections should be chosen to minimize the number of total errors that will be annotated in this sentence. This is the main point in the annotation process where the annotator considers global rather than local requirements. The annotator should consider the entire sentence including not yet annotated errors in order to choose the most appropriate target correction.

- Target corrections should be chosen without regard to meaning. Grammatical non-sense sentences are acceptable targets.

- As much as possible, all content words in the original sentence should be preserved in the target corrections.

- In selection errors, the correction should modify the dependent(s) rather than the head in the phrase affected by the error. For instance, an error where the case of a verb’s NP argument is incorrect, the correction should fix the case of the NP rather than propose a synonymous verb that would fit in this context.

- Agreement errors can be corrected only to available inflection values of the words in the response. For example, *den Schweiz, which agrees in number (singular) and case (accusative) but not gender (masc. vs. fem.), can be corrected to die Schweiz (fem. acc. sing.) but not der Schweiz (fem. dat. sing.).
Correcting *den Schweiz to der Schweiz in the context *aus den Schweiz as an agreement error overlooks the fact that the dative is specified by aus, not by any context internal to the NP. Instead, two errors should be annotated: one that corrects the gender agreement error between den and Schweiz and one that corrects the selection error between aus and the NP.

- Ambiguous errors:
  - More than one target correction should be specified for an ambiguous agreement error (see the following section on ambiguous errors for more detail).
  - If a majority of the words in an NP agree in the feature under consideration (case, gender, number, or definiteness), the error is not ambiguous and only one target correction needs to be marked.

- Subsequent errors can assume the previous errors have been corrected as shown in their target corrections.

- Word order determines the type of final punctuation, i.e., punctuation should be corrected to match word order (as in questions) rather than vice versa.

The following sections describe how these guidelines fit into the annotation of multiple, ambiguous, and overlapping errors.

Incremental Analysis

Because responses from beginning learners often contain multiple errors (cf. Heift, 2003a), the EAGLE annotation scheme extends the basic annotation format of Lüdeling et al.
(2005) to include error numbering, which is specified in the location layer for each error. The error numbering allows the annotator to specify a series of incremental corrections, each with its own detailed error description, that convert the learner’s response into a grammatical target. Each step assumes that previous corrections have been made, which makes it possible to address phrase-internal errors, such as agreement errors, before considering selection or word order. For example, all of the words in a noun phrase need to have the same number, gender, and case before it is possible to determine whether that noun phrase is grammatical as a particular complement of a verb.

In example (26) from the previous section, the subject *dieses Hunden* ‘this dogs’ needs to be internally consistent before an annotator can determine whether the subject agrees with the verb *gehört* ‘belong’. In this instance, the phrase *dieses Hunden* ‘this dogs’ would be annotated as containing a determiner-noun agreement error with the target correction *dieser Hund* ‘this dog (nom sg)’ and once this is complete, the subject-verb agreement can then be examined and be determined to be grammatical. After examining the subject-verb agreement, we can turn to the other verb complement from the example, *Wen* ‘whom’. Instead of an accusative complement *Wen*, the verb *gehört* requires a dative complement *Wem* ‘to whom’. The annotation including both the agreement error and this verb complement case error is shown in Figure 7.11 below.
When the two target corrections in Figure 7.11 are applied to the original sentence, we arrive at a grammatical target sentence:

(27) Wem gehört dieser Hund?  
    to whom$_{dat}$ belong$_{3,sg}$ this$_{masc,nom,sg}$ dog$_{masc,nom,sg}$  
    ‘To whom does this dog belong?’

For example (26), the order in which in the errors are annotated is not important because they do not overlap, but in many instances, the order in which the errors are annotated plays an important role in making it possible to annotate errors that depend on previous target corrections. We will return to the issue of overlapping errors after discussing ambiguous errors in the next section.

**Ambiguous Errors**

The annotation of ambiguous errors with the EAGLE annotation scheme is driven by the fact that it is not possible to determine the student’s knowledge or intent from the EAGLE corpus data, so linguistic features are used instead to distinguish different types of errors. This is especially relevant for distinguishing many types of selection and agreement errors.
Example (26) illustrates how an ambiguous error, such as a large percentage of agreement errors, can cause difficulties in creating consistent annotation. Considering only the noun phrase from the previous example, an annotator could have just as easily corrected *dieses Hunden* ‘this dogs’ to *diesen Hunden* ‘these dogs (D pl)’, which would have had both the incorrect number and case as the subject of the sentence. This would have led to further corrections to reach a grammatical target. In order to avoid these kinds of inconsistencies, an annotator chooses the target that minimizes the total number of errors annotated for the given sentence. Thus, instead of trying to minimize the edit distance between the learner response and the target correction, as in many existing error annotation schemes, the EAGLE annotation tries to minimize the total number of annotated errors.

In cases where the ambiguity is not resolved by the surrounding context, the multi-layer annotation allows for multiple targets to be specified. Because ambiguities most often arise in agreement errors, the EAGLE annotation scheme includes two additional layers in the agreement type for a second error description and a second error target. The additional layers are shown in the example in Figure 7.12, which is described in detail in the next section.

**Overlapping Errors**

A final issue common in learner language productions is overlapping errors. Since different types of errors are annotated in different layers, the multi-layer standoff format makes it simple to annotate such errors. Example (28) shows what the multi-layer standoff format looks like for a response with multiple overlapping errors. This example, which combines
errors from several actual learner responses, contains four errors: 1) a subject-verb agreement error, 2) a noun phrase argument in the wrong case, 3) a word order error in the subordinate clause, and 4) a word order error in the main clause. The EAGLE multi-layer standoff annotation for example (28) is shown in Figure 7.12. In order to show overlapping word order error spans, the word order error layers have been divided into two sets of layers for the main and subordinate clauses.

(28) Als Sofie last dem Fahrplan, sie Reisepläne machte.
when Sofie read the timetable, she made travel plans.
‘As Sofie read the timetable, she made travel plans.’

7.3.5 EAGLE Corpus Annotation

The Partitur (‘musical score’) Editor from the EXMARaLDA (Extensible Markup Language for Discourse Annotation) Project (Schmidt, 2001) was used to perform the annotation. The workbook subcorpus was annotated by a single annotator (the author) and the essay subcorpus was annotated by two German computational linguistics students. The frequencies of the top-level error categories are summarized in Figure 7.13 and the most frequent errors are shown in Figure 7.14. A detailed error analysis is presented in chapter 8.

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>2377</td>
</tr>
<tr>
<td>Agreement</td>
<td>1299</td>
</tr>
<tr>
<td>Word Order</td>
<td>603</td>
</tr>
<tr>
<td>Word Form</td>
<td>770</td>
</tr>
<tr>
<td>Punctuation</td>
<td>949</td>
</tr>
<tr>
<td>Total</td>
<td>6013</td>
</tr>
</tbody>
</table>

Figure 7.13: Errors by Top-Level Category in EAGLE

166
Als Sofie den Fahrplan las, machte sie Reisepläne.
<table>
<thead>
<tr>
<th>Error Category</th>
<th>Error Description</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Punctuation</td>
<td>Missing Sentence-Final</td>
<td>532</td>
</tr>
<tr>
<td>Agreement</td>
<td>Subject-Verb – Number</td>
<td>416</td>
</tr>
<tr>
<td>Selection</td>
<td>Verb – NP Complement Case</td>
<td>393</td>
</tr>
<tr>
<td>Agreement</td>
<td>Det-Adj-Noun – Gender</td>
<td>340</td>
</tr>
<tr>
<td>Word Form</td>
<td>Capitalization</td>
<td>335</td>
</tr>
<tr>
<td>Selection</td>
<td>Sentence – Main Finite Verb Missing</td>
<td>312</td>
</tr>
<tr>
<td>Selection</td>
<td>Preposition – Complement Case</td>
<td>254</td>
</tr>
<tr>
<td>Order</td>
<td>Finite Verb – Main Clause</td>
<td>238</td>
</tr>
<tr>
<td>Agreement</td>
<td>Det-Adj-Noun – Number</td>
<td>197</td>
</tr>
<tr>
<td>Selection</td>
<td>Noun – Determiner Missing</td>
<td>184</td>
</tr>
<tr>
<td>Agreement</td>
<td>Subject-Verb – Person</td>
<td>181</td>
</tr>
<tr>
<td>Agreement</td>
<td>Det-Adj-Noun – Case</td>
<td>152</td>
</tr>
<tr>
<td>Selection</td>
<td>Verb – Missing Complement</td>
<td>148</td>
</tr>
</tbody>
</table>

Figure 7.14: Most Frequent Grammatical Errors in EAGLE

7.4 Interannotator Agreement for EAGLE Annotation Scheme

The EAGLE annotation scheme involves numerous decisions on the part of the annotator—token span selection, error type, and target correction—and each sentence may be annotated with zero to many errors. The combination of these decisions leads to difficulties in establishing acceptable agreement for the annotation scheme in its entirety, however it is possible to calculate interannotator agreement for a simplified version of the annotation that remains meaningful for the intended use of the corpus: the evaluation of a grammar checker for beginning learners of German.

For the purpose of grammar checker evaluation, the focus is on the error description layer of the annotation. The interannotator agreement calculations will take the sentence as the base unit of annotation and consider two different perspectives:
• Grammaticality: a binary decision on the sentence’s grammaticality

• Error Descriptions: the set of error descriptions annotated for a given sentence

The MASI (measuring agreement on set-valued items) distance metric from Passonneau et al. (2006) is used with Krippendorf’s \( \alpha \) to calculate the scores shown in the following sections. MASI is 1 when two sets are identical and 0 when they are disjoint. The distance between two sets takes into consideration the size of the intersection of the two sets and their absolute sizes.

\[
MASIDist(S_1, S_2) = \frac{|S_1 \cap S_2|}{\max(|S_1|, |S_2|)}
\]

The \( \alpha \) values reported below were calculated using a script\(^{20}\) by Tom Lippincott, which has in the meanwhile been incorporated into NLTK (Bird et al., 2009).

The essay subcorpus, which was annotated by two native German-speaking computational linguistics students, contains 1277 sentences in 81 essays.\(^{21}\) Five essays were used for training, leaving 1205 sentences to be annotated independently and thus taken into consideration for the following interannotator agreement calculations. As mentioned earlier in this chapter in section 7.3.1 on non-word spelling error annotation, values of \( \alpha \) above 0.7 can be considered acceptable for complicated annotation schemes such as discourse annotation, although values above 0.8 generally indicate good-quality annotation (Artstein and Poesio, 2009).

\(^{20}\)http://cswww.essex.ac.uk/Research/nle/arrau/Lippincott/agreement/agreement.py

\(^{21}\)Many thanks for to Cornelius Fath and Stefanie Wolf for their work on the essay subcorpus annotation.
Table 7.3: Interannotator Agreement for Grammaticality

<table>
<thead>
<tr>
<th>Error Type</th>
<th>α</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grammaticality</td>
<td>0.781</td>
</tr>
<tr>
<td>Selection</td>
<td>0.725</td>
</tr>
<tr>
<td>Agreement</td>
<td>0.812</td>
</tr>
<tr>
<td>Word Order</td>
<td>0.809</td>
</tr>
<tr>
<td>Word Form</td>
<td>0.680</td>
</tr>
<tr>
<td>Punctuation</td>
<td>0.806</td>
</tr>
</tbody>
</table>

7.4.1 Grammaticality

Grammaticality decisions are derived from the number of errors annotated in a sentence: zero errors is considered grammatical and one or more errors is considered ungrammatical. Agreement is also calculated for each top-level error type: do annotators agree on whether this sentence contains any agreement errors, any order errors, etc.

Interannotator agreement is shown in Table 7.3. The agreement on grammaticality is 0.781, which shows acceptable agreement. The agreement for the presence of agreement, word order, and punctuation errors indicates good-quality annotation and the analysis of interannotator agreement for selection errors also shows acceptable agreement. The interannotator agreement for word form errors, which are a collection of single-word and collocation errors that did not fit well in the other categories, is just below the level of acceptability at 0.68.
### Table 7.4: $\alpha$ for Error Description by Error Typology Depth

<table>
<thead>
<tr>
<th>Error Desc.</th>
<th>$\alpha$ by Error Typology Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>0.566 0.501 - -</td>
</tr>
<tr>
<td>Selection</td>
<td>0.605 0.550 0.533 0.473</td>
</tr>
<tr>
<td>Agreement</td>
<td>0.759 0.726 0.660 0.655</td>
</tr>
<tr>
<td>Word Order</td>
<td>0.788 0.701 0.612 0.610</td>
</tr>
<tr>
<td>Word Form</td>
<td>0.612 0.580 0.550 0.546</td>
</tr>
<tr>
<td>Punctuation</td>
<td>0.793 0.774 0.761 0.679</td>
</tr>
</tbody>
</table>

#### 7.4.2 Error Descriptions

The interannotator agreement for error descriptions is calculated by comparing the set of error description annotations made on each sentence by each annotator. For example, the sentence shown in Figure 7.12 would have a set of four annotated error descriptions: S1.a.i, A1.a, O1.a, and O1.b. Because the error typology is hierarchical, it is possible to compare error descriptions at different depths, so the set above could be compared at the top level (S, A, O, O), at the second level (S1, A1, O1, O1), at the third level (S1.a, A1.a, O1.a, O1.b), etc. The top-level scores differ from the binary scores in the previous section because the error description sets allow for more than one error of the exact same type.

$\alpha$ is shown by top-level category in Table 7.4. The two annotators show an acceptable level of agreement for agreement, word order, and punctuation errors up to the second level of the annotation scheme and also in the third level of annotation for punctuation. Word form errors and selection errors cause more difficulties and the level of agreement is not currently acceptable at any level of the annotation scheme. When all errors are taken into account, $\alpha$ is only 0.566 for the top-level categories. Some causes of disagreement are ambiguity...
between agreement and selection errors mentioned in section 7.3.4 and unclear guidelines for the word form category, where annotators were allowed to add new subcategories as they saw fit.

7.5 Creating Training and Testing Data

The workbook corpus was divided into training and test data using a random 60/40 split for the responses to each activity. This means that training and test data contain answers to the same activities and there may be correct or errorful answers that are present in both the training and test data. This is a realistic division of training/test data for a course workbook context where many students are responding to identical, structured prompts with envisioned target answers. The training data contains 4,709 responses and the test data 3,172 responses.

The essays were similarly split with 60% of the essays for each course in the training data and 40% in the test data. Because the essay prompts are less structured, there are no duplicate responses between the essay training and test data. There are 820 sentences in the training data and 456 sentences in the test data. Due to time limitations, there is not yet an adjudicated version of the essay subcorpus, so the annotation from one annotator, who appeared to be more consistent during the annotation training process, is used in the training and testing data.

7.6 Summary

The EAGLE corpus is the first corpus of grammatical error-annotated data for beginning learners of German. The corpus consists of responses to online workbook activities and
final exam essays from students in the second and third introductory German courses at The Ohio State University. The EAGLE error typology, first mentioned in (Boyd, 2010a) and documented in detail in (Boyd, 2010b), was developed to create a linguistically-grounded error annotation scheme for beginning learners of German. Additionally, the non-word spelling errors from the corpus have been annotated using the CLASSY annotation scheme (Rimrott, 2005).
Chapter 8: Error Analysis for the EAGLE Corpus

A brief overview of the distribution of errors in the EAGLE corpus was provided at the end of chapter 7. This chapter takes a detailed look at the types and distribution of errors found in the EAGLE corpus and the influence of this analysis on the development of the grammar (chapter 9) and grammar checker (chapter 10).

8.1 Overview of Error Distribution

The EAGLE corpus contains 5,706 sentences, 3,444 (∼60%) of which have been annotated with one or more errors including punctuation errors. Disregarding punctuation errors, approximately 54% of sentences contain at least one error. Table 8.1 shows the frequency and distribution of errors in each of the top-level categories for the workbook subcorpus, essay subcorpus, and overall. On average, each sentence has been annotated with approximately one error, which initially appears a little low than expected for beginning learners, who are known to make multiple errors per sentence (Heift, 2003a). However, recall that the workbook subcorpus contains multiple attempts by the same student for the same question, for which the student has received some (limited) feedback from the Quia workbook. This leads to a higher percentage of correct answers than would be found in a corpus gathered
Table 8.1: Workbook vs. Essay Errors by Top-Level Category in EAGLE

![Image of Table 8.1: Workbook vs. Essay Errors by Top-Level Category in EAGLE](image)

from a non-interactive workbook. In the essay subcorpus, where the students received no immediate feedback about their responses, there are approximately 1.5 errors per sentence.

The differences between the distribution of errors between the workbook and essay subcorpora can be explained by differences in the types of activity and situation in which the responses were written (typed answers to untimed prompts with access to reference materials vs. a timed, hand-written exam with no access to reference materials). The workbook subcorpus contains mostly single-sentence responses to structured prompts while the essay subcorpus contain a multi-paragraph responses to an open-ended question such as What did you do this quarter? or Where would you like to study abroad?

It is likely that the lack of access to reference materials and the open-ended nature of the prompts led to the higher percentage of word order errors and word form errors in the essay subcorpus. The workbook exercises frequently provide the word order (e.g., build a sentence using these ordered lemmas) or give model question/answers that help with word order. With restrictions on vocabulary and access to reference materials in the workbook

---

The metadata collected would make it possible to identify the initial responses to each question in the workbook subcorpus for separate analysis.
subcorpus, students are also less likely to make word form errors. The mode of data entry (typed workbook vs. hand-written essays) also affects the nature of word form errors found. For instance, the word form errors such for *sich* and *rau* for *Frau* appear to be purely keyboarding errors, whereas students writing essays in an exam situation are more careful about providing punctuation, especially sentence-final punctuation, which is often missing in single-sentence workbook responses. Sentence-final punctuation errors alone account for 616 errors in the workbook subcorpus while the essay subcorpus contains only three such errors.

### 8.1.1 Comparison to Previous Analyses

Differing annotation schemes make it impossible to perform a direct comparison, but two existing studies of written errors made by learners of German provide some information about the distribution of errors made by more advanced learners of German. Juozulynas (1994) examined written errors made by second-year students and found that 80% of errors are not semantic in nature, indicating that a syntactic parser could detect the majority of errors for intermediate learners. He found the following distribution of errors: 29% syntactic, 24% morphology, 15% spelling, 12% punctuation, and 20% semantics. Rogers (1984) examined 26 essays by students who had had at least four years of German in secondary school and created a detailed error typology covering syntactic, morphological, lexical, orthographic, and English transfer errors. At the advanced level, she found that approximately 35% of errors were syntactic in nature, 35% morphological, 15% lexical, 10% orthographic, and 5% due to complete transfer of an English expression.
8.2 Detailed Error Analysis

In this section the distribution errors for each top-level error type will be examined in turn. The figures extend the error typology presented in chapter 7 with error counts for each type. In a few places, *Other* categories have been added to the typology in order to account for cases that remain partially classified or unclear.\(^{23}\)

8.2.1 Selection Errors

Figure 8.1 presents the error counts for selection errors. As mentioned in chapter 2, English-speaking beginning learners of German are expected to make errors in verb and preposition subcategorization (see section 2.1.5), both of which involve the case system in the selection of subjects and arguments. Since English-speaking learners are often unfamiliar with noun gender and declension in general, many errors are expected when the case system is involved.

Nearly half of the selection errors appear in the Verb - Complement (S1.A) category, demonstrating the expected difficulties with subcategorization and the case system. Noun arguments are the most frequent verb arguments, so it is no surprise the S1.A.i is the most frequent error. Prepositional phrase arguments are also frequent and students have difficulties with two-way prepositions (S1.A.iv–v) with verbs of motion vs. state/location and PP arguments where the preposition case is selected by the verb (S1.A.ii). Errors in the

\(^{23}\) The Exmaralda Partitur editor used in the annotation does not have an integrated mechanism for handling the selection of the error type, so annotators entered error codes by hand. As a result, there are a few unclear error annotations due to typos or where the annotator has not fully specified the error type, e.g., Selection - Verb Complement error (SA.1) without further specification. The annotation has undergone several rounds of consistency checking, however a small number of errors has yet to be addressed.
non-finite verb form with auxiliary verbs (infinitive vs. past participle) and in the choice of auxiliary with the perfect tense haben/sein (S1.A.vi) are also common.

Another frequent source of errors in selection is sentences missing main finite verbs (S5.B.i), which is due to nonsense responses or incomplete sentences in the online workbook, which were annotated with S5.B.i if no main finite verb could be identified. In many cases, it appears that students submitted partial responses to receive hints from the workbook feedback modules before completing the sentence. In other cases, it looks like students were taking advantage of the fact that instructors gave credit for attempting workbook activities and made a no real attempt at the activity: a single learner in the German 103 course entered $k$ as a response to 55 activities and a German 102 learner entered . (a single period) as a response to 21 activities. Case selection with preposition complements (S2.B.i), missing(extra) determiners (S3.A.i–iv), and unclear targets (S5.C) are several additional frequent error types.

8.2.2 Agreement Errors

Agreement error frequencies are shown in Figure 8.2. In an agreement error such as $Du_{2,s}$ $lachen_{3,pl}$. where more than one feature does not agree, a single instance of an error can be annotated with more than one error category (e.g., A1.A, A1.B). The overall agreement error count in Table 8.1 refers to the number of agreement error instances rather than the total number of times agreement error codes are annotated. For subject-verb and determiner-adjective-noun errors, Figure 8.2 shows the number of instances in parentheses for the parent category and the counts of each individual error code in the subcategories.

$^{24}$Sentence - Main clause - Missing (S5.A.i) is used only when a subordinate clauses appears without a main clause.
S1. Verb

A. Complement
   i. NP complement - incorrect case ........................................ 393
   ii. PP complement - incorrect preposition ................................. 28
   iii. PP complement - incorrect case with correct preposition ........ 4
   iv. Two-way PP complement with verb of state/location -
      incorrect preposition or case ........................................... 145
   v. Two-way PP complement with verb of motion -
      incorrect preposition or case .......................................... 41
      a. NP-dependent preposition choice (e.g., in die Schweiz) ....... 13
   vi. VP complement - haben/sein error .................................... 79
   vii. VP complement - incorrect non-finite verb form ..................... 120
   viii. VP zu infinitive complement - zu missing .......................... 16
   ix. VP clausal complement - incorrect complementizer ............... 3
   x. Incorrect correlative ...................................................... 2
   xi. Incorrect complement type ............................................... 109
   xii. Complement should be adverbial ...................................... 2
   xiii. Missing ................................................................. 18
   xiv. Extra ........................................................................... 3
   xv. Other ............................................................................. 60
   B. Separable prefix - impossible form ..................................... 6
   C. Reflexive
      i. Missing ........................................................................... 18
      ii. Extra ............................................................................... 3
      iii. Incorrect case ............................................................. 6

S2. Preposition

A. Complement
   i. Incorrect case .................................................................. 254
   ii. Missing ............................................................................ 11
   iii. Other ............................................................................... 11
   B. Missing ............................................................................. 7
      i. In paired construction (e.g., von/bis) ................................. 4

Figure 8.1: Selection Error Frequency
### S3. Noun

#### A. Determiner
- i. Missing .................................................. 184
- ii. Extra (e.g. die die) ........................................... 54
- iii. Extra with job/description or with als .................... 28
- iv. Pronoun instead of determiner (alles, wem) ................. 28
- v. Pronominalized determiner instead of determiner (eins) ..... 1
- vi. Adverb instead of determiner (sehr instead of viel) ........ 3
- vii. Other .......................................................... 1

#### B. Complement
- i. NP complement - incorrect case ................................ 3
- ii. PP complement - incorrect preposition .......................... 6
- iii. PP complement - incorrect case with correct preposition .... 0
- iv. Incorrect complement type ....................................... 6

#### C. Modifier
- i. Genitive NP modifier - incorrect case ........................... 14

### S4. Adjective

#### A. Complement
- i. NP complement - incorrect case ................................ 0
- ii. PP complement - incorrect preposition .......................... 3
- iii. PP complement - incorrect case with correct preposition .... 0
- iv. Incorrect complement type ....................................... 8

#### B. Comparative clause
- i. Incorrect type (wie/als or wie/als missing) ...................... 4
- ii. Non-parallel after wie/als ....................................... 6

---

**Figure 8.1: Selection Error Frequency (cont.)**
S5. Sentence

A. Main clause
   i. Missing .................................................. 35
   ii. Extra coordinating conjunction ...................... 1

B. Finite verb
   i. Missing in main clause ................................. 312
   ii. Missing in subordinate clause ....................... 10
   iii. Extra in main clause ................................. 26
   iv. Imperative form with subject ........................ 2
   v. Infinitive with subject (only *sein*) ................. 3

C. Target unclear
   i. Main clause ............................................. 7
   ii. Subordinate clause .................................... 2
   iii. Other .................................................. 111

D. Subordinate clause (not a verb complement) ............ 3
   i. Incorrect subordinating conjunction ................ 9
   ii. Missing subordinating element ..................... 1

S6. Conjunction

A. Coordinating
   i. Missing coordinating conjunction .................. 18
   ii. Mismatched determiner or adjective ............... 7
   iii. Missing phrase in coordination .................... 1
   iv. Unlike coordinated phrases .......................... 0

B. Subordinating
   i. Incorrect tense ........................................ 23

S7. Determiner/Inflected Adjective

A. Missing head noun ........................................ 40

B. Relative pronoun with no antecedent ................... 0

S8. Other ..................................................... 33

Figure 8.1: Selection Error Frequency (cont.)

181
As nearly all sentences require subject-verb agreement and include noun phrases requiring determiner-adjective-noun agreement, there are many errors as students practice noun phrase and verb inflection. Number errors appear more frequently than person errors in subject-verb agreement, which may be due in part to the fact that students frequently use the citation form of verbs (the infinitive, which is the same as the 1st/3rd person plural form for all verbs except to be) with singular subjects, but there are also many occurrences of plural subjects with 3rd singular verbs. Some of the occurrences of singular subjects with plural verbs may be due to the structure of the activities, which often provide vocabulary prompts in the citation form. A few agreement error types such as appositives (A7) were envisaged when the EAGLE error typology was developed, but the few instances of apposition in EAGLE, most of which involve proper names, are all grammatically correct.

8.2.3 Word Order Errors

The distribution of word order errors is shown in Figure 8.3. Word order error types related to topological fields may overlap, so it is sometimes not possible to separate a verb placement error from a Mittelfeld ordering error with clear target corrections for each, so a single word order error span may be annotated with multiple error types. As in the agreement error types, Table 8.1 shows the number of instances of word order errors while Figure 8.3 shows the number of occurrences of each individual error. There are a total of 567 instances of error types O1–O4 with 591 annotated error types.

The placement of verbs in sentences (O1–O3) is clearly difficult for beginning learners of German. The most frequent error, finite verb position in a main clause, may be the most frequent as compared to the other verb forms merely because finite verbs in main
<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject-Verb</td>
<td>(503)</td>
</tr>
<tr>
<td>A. Person</td>
<td>181</td>
</tr>
<tr>
<td>B. Number</td>
<td>416</td>
</tr>
<tr>
<td>Determiner-Adjective-Noun</td>
<td>(747)</td>
</tr>
<tr>
<td>A. Gender</td>
<td>340</td>
</tr>
<tr>
<td>i. Of gendered noun (e.g., Freundin)</td>
<td>4</td>
</tr>
<tr>
<td>B. Number</td>
<td>197</td>
</tr>
<tr>
<td>C. Case</td>
<td>152</td>
</tr>
<tr>
<td>D. Definiteness</td>
<td>19</td>
</tr>
<tr>
<td>E. Attributeness</td>
<td>47</td>
</tr>
<tr>
<td>Relative Pronoun-Antecedent</td>
<td></td>
</tr>
<tr>
<td>A. Gender</td>
<td>13</td>
</tr>
<tr>
<td>B. Number</td>
<td>1</td>
</tr>
<tr>
<td>Negative</td>
<td></td>
</tr>
<tr>
<td>A. Double negative</td>
<td>0</td>
</tr>
<tr>
<td>B. Incompatible (schon with noch nicht)</td>
<td>1</td>
</tr>
<tr>
<td>Subject-Predicate with Copula</td>
<td></td>
</tr>
<tr>
<td>A. Number</td>
<td>7</td>
</tr>
<tr>
<td>Reflexive-Subject</td>
<td>8</td>
</tr>
<tr>
<td>Appositives</td>
<td>0</td>
</tr>
<tr>
<td>A. Gender</td>
<td></td>
</tr>
<tr>
<td>B. Number</td>
<td></td>
</tr>
<tr>
<td>C. Case</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>19</td>
</tr>
</tbody>
</table>

*A single error may be annotated with one or more subtypes, so the individual counts overlap. The number of instances is shown in parentheses.

*Ditto.*

Figure 8.2: Agreement Error Frequency
clauses occur in every sentence while subordinate clauses and non-finite verb forms are less frequent in beginning learner productions. The next most common type of error, also closely tied to the topological field model, is in the order of elements in the *Mittelfeld* (O4), which is split relatively evenly between errors in argument order and in adjunct order. There are only a handful of occurrences of the remaining word order error types.

### 8.2.4 Word Form Errors

As much as possible, grammatical errors are annotated using the selection, agreement, and word order error types, however there are many remaining errors in the form of individual words or collocations. Some error types, such as capitalization (W1) or inflection (W2) relate directly to the form of single words with very little reference to the surrounding context, e.g., the first word in a sentence should be capitalized or the superlative predicate adjective form requires *am* (*am kleinsten* ‘the smallest’), whereas the remainder are more context-dependent.

As the collection of error types is not structured according to any linguistic principles, the analysis of these error types does not point to any properties of German that pose problems for learners. Beginning learners make a wide range of errors in areas that are difficult when learning any new language: collocations, confusion when looking up English homonyms in the dictionary, orthographic conventions, etc.

### 8.2.5 Punctuation Errors

The grammar checker presented in chapter 10 ignores punctuation, however it seemed prudent to annotate a limited number of types of punctuation errors during the annotation
<table>
<thead>
<tr>
<th>Error Type</th>
<th>Subcategories</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1. Finite verb</td>
<td>A. In a main clause</td>
<td>238</td>
</tr>
<tr>
<td></td>
<td>B. In a subordinate clause</td>
<td>76</td>
</tr>
<tr>
<td>O2. Non-finite verb</td>
<td></td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>A. Reordered outside clause</td>
<td>3</td>
</tr>
<tr>
<td>O3. Separable prefix</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>O4. <em>Mittelfeld</em></td>
<td>A. Arguments</td>
<td>91</td>
</tr>
<tr>
<td></td>
<td>B. Adjuncts</td>
<td>82</td>
</tr>
<tr>
<td>O5. <em>Vorfeld</em></td>
<td>A. Contains interrogative pronoun in question</td>
<td>1</td>
</tr>
<tr>
<td>O6. Prepositional phrase</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>A. Involving <em>was für</em></td>
<td>4</td>
</tr>
<tr>
<td>O7. Noun phrase</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>A. <em>Kein ... mehr</em></td>
<td>1</td>
</tr>
<tr>
<td>O8. Adverb phrase</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>A. <em>Noch nicht</em></td>
<td>2</td>
</tr>
<tr>
<td>O9. Other</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

*aSpans containing errors O1–O4 may overlap, so some spans are annotated with two error types when it is not possible to clearly separate two error types. For errors O1–O4 there are 591 individual error annotations corresponding to 567 spans.*
<table>
<thead>
<tr>
<th>Section</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1. Capitalization</td>
<td>335</td>
</tr>
<tr>
<td>W2. Inflection</td>
<td></td>
</tr>
<tr>
<td>A. Noun</td>
<td>13</td>
</tr>
<tr>
<td>B. Verb</td>
<td>14</td>
</tr>
<tr>
<td>C. Adjective</td>
<td>25</td>
</tr>
<tr>
<td>W3. Single-letter real word spelling error</td>
<td>59</td>
</tr>
<tr>
<td>A. <em>regen</em> for <em>regnen</em></td>
<td></td>
</tr>
<tr>
<td>B. <em>such</em> for <em>sich</em></td>
<td></td>
</tr>
<tr>
<td>C. <em>dass</em> for <em>das</em></td>
<td></td>
</tr>
<tr>
<td>D. <em>Hat</em> for <em>Hut</em></td>
<td></td>
</tr>
<tr>
<td>E. ...</td>
<td></td>
</tr>
<tr>
<td>W4. Collocations</td>
<td>65</td>
</tr>
<tr>
<td>A. Dates (e.g., <em>im Montag</em>)</td>
<td></td>
</tr>
<tr>
<td>B. Descriptions (e.g., <em>von Leder</em>)</td>
<td></td>
</tr>
<tr>
<td>C. Locations (e.g., <em>nach</em> for <em>bei</em>)</td>
<td></td>
</tr>
<tr>
<td>D. Transitions (e.g., <em>dafür</em> for <em>deswegen</em>)</td>
<td></td>
</tr>
<tr>
<td>E. Sport (e.g., <em>Schlittschuh gehen</em>)</td>
<td></td>
</tr>
<tr>
<td>F. ...</td>
<td></td>
</tr>
<tr>
<td>W5. Dictionary errors</td>
<td>121</td>
</tr>
<tr>
<td>A. <em>erste</em> for <em>zuerst</em></td>
<td></td>
</tr>
<tr>
<td>B. <em>Marke neu</em> for <em>nagelneu</em></td>
<td></td>
</tr>
<tr>
<td>C. <em>Außerste</em> for <em>draußen</em></td>
<td></td>
</tr>
<tr>
<td>D. <em>nach</em> for <em>nachdem</em></td>
<td></td>
</tr>
<tr>
<td>E. ...</td>
<td></td>
</tr>
<tr>
<td>W6. Miscellaneous errors</td>
<td>123</td>
</tr>
<tr>
<td>A. Doubled word</td>
<td></td>
</tr>
<tr>
<td>B. <em>nicht</em> vs. <em>keine</em></td>
<td></td>
</tr>
<tr>
<td>C. Extra punctuation (copy-paste from activity)</td>
<td></td>
</tr>
<tr>
<td>D. Extra space between separable prefix and verb-final verb</td>
<td></td>
</tr>
<tr>
<td>E. Compound formed with space</td>
<td></td>
</tr>
<tr>
<td>F. Numbers formed with space</td>
<td></td>
</tr>
<tr>
<td>G. Adjectives derived from place names formed incorrectly</td>
<td></td>
</tr>
<tr>
<td>H. ...</td>
<td></td>
</tr>
<tr>
<td>W7. Other</td>
<td>13</td>
</tr>
</tbody>
</table>

Figure 8.4: Abbreviated Word Form Error Frequency
### P1. Comma

<table>
<thead>
<tr>
<th>A. Missing comma</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>i. Between coordinated elements</td>
<td>17</td>
</tr>
<tr>
<td>ii. Between main and subordinate clauses or other clauses where a comma is required</td>
<td>49</td>
</tr>
<tr>
<td>iii. Other</td>
<td>17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B. Extra comma</th>
</tr>
</thead>
<tbody>
<tr>
<td>i. Between coordinated elements</td>
</tr>
<tr>
<td>ii. Other</td>
</tr>
</tbody>
</table>

### P2. Other punctuation | 2 |

| A. Missing | 4 |
| B. Extra | 18 |

### P3. Sentence final punctuation

| A. Incorrect | 87 |
| B. Missing | 532 |
| C. Extra | 0 |

---

**Figure 8.5: Punctuation Error Frequency**

Process. The punctuation error distribution is shown in Figure 8.5. There are several significant differences in punctuation usage between English and German, in particular in comma usage, which is demonstrated by the frequent errors in the P1 error type. However, by far the most frequent error type, missing sentence-final punctuation (P3.B), is due to the structure of the online workbook, which leads to the submission of partial or nonsense answers without final punctuation (see explanation in section 8.2.1).

### 8.3 Relevant Phenomena for Grammar Development and Grammar Checking

On the one hand, the grammatical sentences in the EAGLE corpus indicate the grammatical phenomena that need to be covered by a grammar check for beginning learners of
German. These include the standard phenomena in German grammar mentioned in chapter 2: subject-verb agreement, noun-adjective-determiner agreement, verb subcategorization frames with NP and PP, separable verb prefixes, relative clauses, predicate nouns and adjectives, etc. On the other hand, the types of errors show where a grammar needs to be rich enough in order to distinguish grammatical constructions from error, for instance, it would not be sufficient to specify NP verb arguments without reference to case, as is the case in many robust statistical parsers of German. Such a parser could not distinguish an accusative NP from a dative NP, which would make it impossible to detect cases where a learner has made a mistake in the case of an NP argument. The grammar will focus on the most frequent phenomena and the most frequent errors in order to provide the best coverage possible given the restricted amount of development time available while working on this thesis, i.e., person-months rather than the many person-years that have gone into the most successful hand-written grammars (e.g., XLE (Crouch et al., 2006), LinGO English Resource Grammar (Flickinger, 2000)

8.3.1 Sentence-Level Analysis

The Fledgling grammar checker needs to identify whether parses contain a main clause with a finite verb in order to diagnose errors in S5, in particular the very frequent S5.B.i error.
8.3.2 Verb Subcategorization Frames

To diagnose verb complement selection errors, verb subcategorization frames should include NP complements (case), PP complements (preposition and case), adverbial complements including case required with two-way prepositions, *haben/sein* with past participles, reflexives (case), and non-finite VP complements (infinitive vs. past participle). Clausal complements and errors related to clausal complements are rare in EAGLE, so as long as the parser accepts clausal complements, detailed information about clausal complement selection may be omitted at the cost of several errors not being detected.

8.3.3 Noun Phrases

The most frequent errors in terms of selection in noun phrases is whether a determiner is required. Several classes of nouns are needed: common count and mass nouns, proper nouns, etc.

8.3.4 Subject-Verb Agreement

The grammar should be able to detect person and number agreement between subjects and verbs for error A1.

8.3.5 Determiner-Adjective-Noun Agreement

Likewise, the grammar should be able to detect determiner-adjective-noun agreement in noun phrases. Determiners, adjectives, and nouns need to have gender, number, case, and definiteness features to make it possible to detect agreement errors in A2. The attributeness
error, which is used for errors where beginning learners who have not yet learned about adjective endings use a predicate adjective form in an attributive instance (\textit{der klein}_{\textit{pred}} Mann ‘the small man’), can be identified when an extra word error points to an adjective in a noun phrase.

8.3.6 Relative Pronoun-Antecedent Agreement

The grammar needs to enforce agreement in gender and number between relative pronouns and antecedents for error A3.

8.3.7 Reflexive-Subject Agreement

Agreement between reflexive pronouns and subjects in person and number is also necessary for error A6.

8.3.8 Topological Field Order

The grammar needs to identify the main topological fields (\textit{Vorfeld}, left sentence bracket, \textit{Mittelfeld}, right sentence bracket, \textit{Nachfeld}) in order to detect errors O1–O3.

8.3.9 Mittelfeld Order

The Fledgling grammar checker will detect only a limited number of Mittelfeld ordering errors, in particular those involving pronoun verb arguments, which have a relatively fixed order in the Mittelfeld in the kinds of sentences produced by beginning learners. Properly characterizing word order in the Mittelfeld for all NPs and adverbials requires information
about semantics, pragmatics, and information structure that is outside the scope of the syntax-focused Fledgling.

8.4 Summary

The EAGLE corpus contains a wide range of errors that show the difficulties beginning learners face in mastering the German morphology and word order, which differ greatly from English. Errors are found in nearly all of the envisaged categories in the error typology and the typology was extended during the annotation process to handle unexpected types of errors, in particular word form errors such as collocation and dictionary errors. The frequent types of errors indicate the grammatical phenomena that need to be included in a grammar used for grammar checking. This includes detailed information about verb subcategorization frames, subject-verb agreement, noun phrase agreement, and topological fields for word order errors.
Part III

Error Detection and Diagnosis
This chapter describes the customizations and extensions to existing demonstration XDG grammars needed in order to provide the coverage and detailed analysis required for the EAGLE corpus for the Fledgling grammar checker.

9.1 Grammar Development

The first task is to extend the coverage of existing XDG demonstration grammars in order to provide sufficient coverage of the grammatical phenomena in the EAGLE corpus. Section 9.1.1 describes the coverage of existing demonstration grammars and section 9.1.2 describes the grammar extensions and modifications, illustrated with parses of constructed examples and sentences from EAGLE.

9.1.1 Coverage of Existing XDG German Grammars

The starting point for the grammar development for the Fledgling grammar checker is an existing XDG demonstration grammar softproj (Bader et al., 2004), which is an extension of a grammar (diplom) developed in (Debusmann, 2001b). These demonstration grammars are provided as part of the XDG Development Kit (Debusmann et al., 2004). The diplom grammar covered a wide range of phenomena related to the verbal cluster. The
softproj grammar simplifies some of the verb classes and adds (with various degrees of completeness) many additional phenomena: free relative clauses, verbal complements of nouns, comparative clauses, passives, imperatives, noun classes and semantic noun constraints, adverbs and some semantic adverb constraints, subordinating and coordinating conjunctions, copula verbs and predicate nouns, questions, placeholder es and explicative es.

As the development of the demonstration grammars has mainly been intended to show how XDG can handle complex grammatical phenomena, the focus has been on complex verb clusters and other constructions that rarely appear in beginning learner language. Aside from good coverage of the topological field model, which is needed for any German sentences and thus a thorough account of word order, many basic features of German grammar as described in chapters 2 and 8 are missing.

9.1.2 Grammar Extensions and Modifications

In order to reach a high level of coverage of the grammatical phenomena in the EAGLE corpus, many new phenomena have been added to the grammar and existing phenomena have been modified to improve the precision of the grammar for the purpose of error diagnosis. The following sections describe the modifications made to verb subcategorization frames, adjuncts, relative clauses, noun classes, predicate adjectives, and coordinating conjunctions.
Figure 9.1: ‘I am annoyed with him.’

**Verb Subcategorization Frames**

In verb subcategorization frames, coverage of the following complements has been added: reflexive pronouns, PP complements with preposition + case, locative PPs, and movement/direction PPs. *da-* and *wo* compounds with prepositions can also be PP complements. Figure 9.1 shows the analysis of a verb requiring an accusative reflexive and a PP argument as *über* with the accusative. Figure 9.2 shows the same verb with the PP argument realized as a *wo*-compound.

Support for the syntactic (although not the predicate-argument structure) analysis of raising and control verbs has been added. Figure 9.3 shows a raising verb. The future with *werden* and the use of separable prefixes with imperatives and *zu* (e.g., *anzurufen*) has also been implemented as demonstrated in Figure 9.4.
Figure 9.2: ‘What are you annoyed about?’

Figure 9.3: ‘It seems to be raining.’
Verb Adjuncts

Support for adjunct PPs separate from verb arguments has also been added using a constraint that is overly simplistic, but adequate for EAGLE coverage: two-way prepositions always take the dative in adjunct PPs. Figure 9.5 shows a parse with an adjunct PP.

Relative Clauses

Free relative clauses that were covered in softproj have been removed from the grammar to reduce ambiguity. Agreement in gender and number has been added between relative pronouns and their antecedents, as the softproj approach was not precise enough for error detection in this area. In the EAGLE grammar, relative pronouns are able to be the objects of PPs and wo-compounds are also supported as relative pronouns. A relative clause is shown in Figure 9.6.
Figure 9.5: ‘He is sleeping in his apartment.’

Nouns

The noun class has been split into mass and count nouns and there is a separate class that contains proper names. The titles Herr, Frau, and Herr Magister expect a single following proper name. Was für ‘what kind of’ and was für ein are supported as a type of determiner. In order to simplify the analysis, was für (ein) is converted into a single token during preprocessing (see section 9.3).
Figure 9.6: ‘Should I read the job posting that interests me the most?’
Predicate Adjectives

Support for predicate adjectives including superlatives with *am* and positive and comparative adjectives with *so...wie* ‘as...as’ and *als* ‘than’ clauses respectively has been implemented. Figure 9.8 shows a predicate comparative adjective (*adj.cop*) with an *als* clause.

Coordinating Conjunctions

The treatment of coordinating conjunctions *softproj* is not very general, with a separate entry of each conjunction for each type of coordinated phrase, but it is sufficient for most of the sentences produced by beginning learners. In the EAGLE grammar, coordinating conjunctions have been extended in order to allow coordination of predicate adjectives, non-finite verb phrases, prepositional phrase arguments, and appositive proper names. The
maximum number of phrases allowed in the coordination has been increased from three (as in softproj) to six in order to be able to parse the longest coordinate phrases found in the EAGLE training data.

The implementation of coordination is limited to coordination of complete phrases of the same type, which causes problems for a number of responses in EAGLE. The corpus contains very little unlike coordination or non-constituent coordination, but a frequent type of coordination not covered in the EAGLE grammar is finite VPs coordination, e.g. *Er geht in den Wald und wandert.* ‘He goes into the forest and hikes.’ Because the XDG dependency analysis treats the subject just like any other verb argument, *geht in den Wald* is analyzed as a sentence missing a subject rather than a complete finite VP, so coordination is more
ich habe eine Gabel einen Teller einen Löffel und ein Messer

Figure 9.9: ‘I have a fork, a plate, a spoon, and a knife.’
difficult to implement than with non-finite VPs or other phrases such as NPs, PPs, and AdjPs.

**Other Additions**

Accusative NPs for a limited list of nouns derived from the training data (*Tag, Abend, Woche, etc.*) can appear as temporal adverbials (e.g., *jeden Abend*). Figure 9.10 shows a parse with *jeden Tag* ‘every day’ as an adverbial. Limited implementation of appositives, restricted to only proper names, has been implemented and in PPs, coverage of *als* + predicative noun has been added.
9.1.3 Evaluating XDG Principles

In almost all cases, the existing XDG principles are powerful enough for the EAGLE grammar implementation. The main XDG principles used to implement the EAGLE grammar are agreement, agreement Subset, government, relative, and order (see Debusmann, 2006). The relative principle is sufficient for nominative, accusative, dative relative pronouns, but genitive relative pronouns are not restricted to relative clauses by the principle. As a result, genitive relative pronouns are not restricted to relative clauses, which leads to spurious parses where a genitive relative pronoun that modifies a noun phrase can appear incorrectly as a genitive modifier, e.g., *der Mann dessen Sohn ‘*the man whose son’ is parsed as a correct noun phrase.

9.1.4 Customizing Constraints for Error Diagnosis

One of the goals for the XDG demonstration grammars is to provide concise analyses of grammatical phenomena. For the purposes of error diagnosis, conciseness is not always desired as it can prevent a grammar checker from distinguishing particular types of errors. For example, the softproj grammar uses a single constraint to check subject-verb agreement and determiner-adjective-noun agreement in person, gender, and number.25 An informal statement of this constraint is as follows:

Require agreement in person, gender, and number between all pairs of tokens in subj, adj, or det relations.

25Since a verb has no gender assignment, it agrees with any noun in gender.
When a conflict detection algorithm such as QuickXplain identifies that this constraint cannot be satisfied in an ungrammatical sentence, it would be necessary to examine the lexical entries of the words involved to identify which kind of agreement was involved. For error diagnosis, a much more straightforward approach is to split this constraint into two separate agreement constraints:

1) Require agreement in person, gender, and number between all pairs of tokens in subj. (*subject-verb agreement*)

2) Require agreement in person, gender, and number between all pairs of tokens in adj, or det relations. (*det-adj-noun agreement*)

The drawback to splitting apart concise constraints is that a less concise XDG grammar generates a CSP with more variables and constraints. With the current EAGLE grammar, each XDG agreement principle generates five low-level constraints for each pair of words in the input sentence, so the size of the CSP can grow dramatically when constraints are separated. An increase in the number of constraints leads to longer running times for CSP solvers and for conflict detection algorithms such as QuickXplain, whose running time depends on the number of constraints. A grammar developed for error diagnosis needs to find a balance between causing the total number of constraints to explode and being able to distinguish errors easily in order to provide targeted feedback.

For the EAGLE grammar, constraints were split apart where needed in order to distinguish the main categories in each section of the error typology, e.g., verb selection (S1) is split from preposition selection (S2) and Mittelfeld ordering errors (O4) are split from finite verb ordering errors (O1). Most selection and agreement constraints in the EAGLE grammar are naturally linked to one specific error type, such as the single government constraint that requires that reflexive verb argument is in the right case, corresponding to a single error.
type, S1.C.iii. Most selection and agreement constraints that cover more than one error type, such as the subject-verb/determiner-adjective-noun mentioned, are easy to separate into individual constraints for each error type.

A few constraints correspond to parallel error types in multiple subcategories. For instance, a single constraint handles the grammatical case of all preposition arguments, which means that arguments of adjunct prepositions and the arguments in PP complements of verbs, nouns, and adjectives (S2.A.i, S1.A.iii/iv/v, S3.B.iii, and S4.A.iii respectively) all correspond to same constraint. This means that the error diagnosis module can identify that a preposition argument is in the wrong case, but without closer analysis of a parse, it can’t distinguish these error types.

A second exception is the word order order principle in the XDG principle library, which is a monolithic constraint that states precedence constraints on all fields (labels) in the lp dependency tree at once. The noun phrase fields \( \text{nvf, nmf, nxf} \) do not overlap with the clause-level topological fields \( \text{vf, v12, mf, ...} \), so it is sufficient and concise to use a single ordering constraint:

\[
\text{nvf} \prec \text{nmf} \prec \text{n} \prec \text{nxf} \prec \text{vf} \prec \text{v12} \prec \text{mf} \ldots
\]

This single constraint makes it difficult to distinguish topological field ordering errors (O1–O5) from NP/PP-internal ordering errors (O6–O8) and from distinguishing different types of topological field errors. The order principle can be stated only once per dimension, so in order to accommodate a set of separate ordering constraints, a new principle has been added, orderplus, which makes it possible to add additional ordering constraints alongside one occurrence of the original order principle.
9.2 Extending the Lexicon

After implementing additional grammatical phenomena, the second task is to extend the lexicon. The softproj lexicon is a very small demonstration lexicon with only 425 lexical entries for 240 unique tokens. In contrast, EAGLE contains approximately 3100 unique tokens for 2000 lemmas.

9.2.1 Lexical Coverage

First, all closed class lexical items in EAGLE were added by hand including: auxiliary and modal verbs, separable prefixes, prepositions, determiners, and pronouns. Next, as described in the following sections, lexical entries for open class items were generated using a morphological generator and a database of verb subcategorization frames.

9.2.2 Morphological Generation

A list of all lemmas in EAGLE was generated using the morphological analyzer/generator Morphy (Lezius, 2000) from a list of tokens. This list of lemmas was then processed by Morphy’s generator to generate all inflected forms of each word.

XDG lexical entries were generated for all forms in Morphy’s output, which includes morphological analysis and forms for uninflected and inflected lemmas. The uninflected parts-of-speech include coordinating and subordinating conjunctions, adverbs, and proper
names. The inflected parts-of-speech are verbs, adjectives, and nouns. With the exception of Konjunktiv I (subjunctive of indirect discourse) forms generated for verbs, all forms generated by Morphy are added to the lexicon.

Separable verb prefixes pose a problem for a lexicon generated using list of tokens and a morphological analyzer: the list of lemmas generated from the EAGLE corpus may only include the separated forms of a separable prefix verb, e.g., only er ruft an ‘he is calling’ is observed and never er hat angerufen ‘he called’. To make sure that all potential separated forms of separable prefix verbs are in the lexicon, lexical entries are generated for separable prefix verbs identified by Morphy for the following separable prefixes: ab, an, auf, aus, bei, durch, ein, entlang, fern, her, hin, mit, nach, vor, weg, zu, zurück, zusammen.

The XDG tools can handle large vocabularies (hundreds of thousands of entries), but to reduce the time needed to read in the grammar, only those tokens that appear in EAGLE are added to the EAGLE grammar for the evaluation presented in chapter 11, which reduces the number of lexical entries from over 200,000 to about 20,000.

Of particular importance for the EAGLE grammar (or any precision grammar) are verb subcategorization frames, which are not provided by Morphy. The subcategorization frames are derived from a subcategorization frame database (Buchholz, 1996) as described in the following section.

---

26 Genitive forms of proper names are currently ignored because they are so rare in EAGLE, but Morphy does generate them and they could be incorporated into future versions of the EAGLE grammar.

27 About a third of these entries are for separated forms of separable prefix verbs that never actually occur in EAGLE, however without deeper analysis of EAGLE, it is not possible to filter out these unused verbs.
9.2.3 Verb Subcategorization Frame Database

The verb subcategorization frame database (Buchholz, 1996) is used to automatically generate verb frames for the verb tokens generated by Morphy. The subcategorization frames included in EAGLE are limited to the 200 most frequent frames (out of a total of 1487) in the Buchholz database, which limits the number of spurious parses for infrequent frames. The automatically generated verb frames incorporate all nominal arguments (although no distinctions are made among the different types of nominal arguments as in the Buchholz database), PP arguments, reflexives, locative and directional adverbials, and the type of auxiliary (haben vs. sein). Clausal argument specifications and other types of currently are currently ignored since the listed arguments provide sufficient coverage for the EAGLE training data. The EAGLE grammar in its current form overgenerates by allowing any verbs to take clausal arguments, however the number of clausal arguments in EAGLE is so small that this is not problematic for the grammar checker.

9.2.4 Additional Vocabulary

Morphy was not familiar with all nouns found in EAGLE and the Buchholz verb database did not include all of the necessary verbs and frames for EAGLE, so a small number of lexical entries was added by hand for tokens in EAGLE. A few were added by hand due to changes in standard German orthography since Morphy was developed, e.g., vergiß vs. vergiss ‘forget (imp.)’. In total, approximately 350 lexical entries have been added by hand.
for \(\sim 150\) common nouns, \(\sim 50\) proper nouns, \(\sim 50\) verbs, \(\sim 100\) adjectives,\(^{28}\) and a handful of adverbs.

A number of subcategorization frames was added to the Buchholz database in order to generate the lexical entries needed for the EAGLE training data. Some frequent verbs were unexpectedly missing from the database (e.g., "vergessen ‘forget’, bewerben ‘apply’"), but in other cases it was only necessary to add a new frame entry (e.g., for "kosten ‘cost’, lernen").

### 9.3 Preprocessing

In order to reduce the time needed for grammar development, punctuation has been ignored and several phenomena are handled in preprocessing steps. In most cases, patterns are converted to specially marked tokens.

#### 9.3.1 Sentence-Initial Sentence Equivalents

The *Satzäquivalente* (sentence equivalents) *Ja* ‘yes’, *Nein* ‘no’, *Ach* ‘oh’, and *Entschuldigung* ‘sorry’ followed by a comma at the beginning of a sentence are currently ignored.

#### 9.3.2 Contractions

Preposition/determiner contractions are split into the preposition and a marked determiner token. *im* ‘on the’ is converted to *im _m* and _m_ has a lexical entry identical to *dem* ‘the’.

\(^{28}\)Somewhat unexpectedly, Morphy was not able to generate all attributive forms for the irregular adjectives *hoch* ‘high’, *gut* ‘good’, and *groß* ‘large’.
There are 16 supported contractions with three marked lexical entries for \( m \), \( s \), and \( r \).\(^{29}\) The \( am \) splitter identifies superlative \( am \), which is not split to \( an m \). The pronoun \( es \) is rarely contracted in EAGLE, but there are occurrences of \( gehts \) ‘goes it’ (as in \( wie geht’s \) ‘how are you’), which is split into \( geht es \).

### 9.3.3 Numbers, Adverbials, and Other Patterns

Several patterns are converted to generic lexical entries. Cardinal numbers written as digits are all converted to EIN (1) or NUM (2–) and ordinal numbers are converted to NUMTH. Several adverb phrases related to time are recognized and converted to ADVERBIAL: \( um \ldots Uhr \) ‘at \ldots o’clock’, \( zwischen \ldots und \ldots Uhr \) ‘between \ldots and \ldots o’clock’, \( von \ldots bis \ldots Uhr \) ‘from \ldots to \ldots o’clock’.

Prepositional phrases with \( pro \) ‘per’ are converted to a generic PRO_UNIT token and the nouns are limited to those in the training data: \( Tag \) ‘day’, \( Woche \) ‘week’, \( Jahr \) ‘year’, \( Kurs \) ‘course’, \( Nacht \) ‘night’. A more general treatment will be needed in the future.

As mentioned in section 9.1.2, \( was für (ein) \) ‘what kind of (a)’ is converted into a single token, with a separate lexical entry for each form of \( ein \). \( Wie viel \) ‘how much’ and \( wie oft \) ‘how often’ are also converted into single tokens.

Finally, the prepositional phrase \( zu X \) ‘to X’ is analyzed by default as a directional PP, so the frequent collocation \( zu Hause \) ‘at home’ is converted to \( zu Hause \) and treated as a

\(^{29}\)There are cases where the use of a contraction is obligatory or leads to a slight semantic differences, however these differences are not included in the syntax-focused EAGLE error typology and are not intended to be detected by the Fledgling grammar checker.
locative adverbial. This allows the grammar to distinguish the directional PP nach Hause ‘(towards) home’ from the locative PP zu Hause.

### 9.3.4 Capitalization

In order to reduce ambiguity and to reduce the number of lexical entries, capitalization is normalized at the beginning of a sentence. If the first word in a sentence is found capitalized in the lexicon, it is left capitalized, otherwise the first letter is changed to lower case.

There are a few infrequent nouns that have much more common lowercase verb, adverb, and pronoun counterparts, so at the beginning of a sentence, the following words, all taken from the training data, are converted to lower case: Hast, Haben, Gehen, Würden, Nachmittags, Morgens, Soll, Gehören, Sein, Koch.

Remaining problems are the pronoun/possessive pronoun pairs Sie/sie ‘you (formal) / she, they’ and Ihr/ihr ‘your (formal) / you (plural)’. Both can be distinguished by examining the form of the following word, which is always the finite verb. Ihr/ihr can be distinguished by determining whether the following token is 2nd person plural verb, and Sie/sie by looking for a 3rd person singular verb, which makes it possible to distinguish 3rd person singular sie but not Sie ‘you (formal)’ from sie ‘they’. The verb patterns are currently detected using regular expressions, which are sufficient for EAGLE but are not really adequate for general use. Both these patterns and the infrequent nouns mentioned in the previous paragraph could be detected with a part-of-speech tagger in the future.
9.4 Establishing Coverage

To test the coverage of the EAGLE grammar, the grammatical sentences from the workbook subcorpus were extracted from the training data. The EAGLE grammar currently returns at least one parse for \(\sim83\%\) of the 2866 sentences. 61 sentences are skipped due to missing vocabulary and 430 sentences cannot be parsed. See section 11.4.1 for a discussion of the issues affecting coverage of grammatical sentences for the EAGLE grammar.

9.5 Summary

The existing XDG softproj demonstration grammar provided a solid starting point for a grammar with coverage for the EAGLE corpus. The grammar was extended to cover the grammatical constructions found in EAGLE, in particular the coverage of verb arguments. The constraints in the grammar

The morphological analyzer/generator Morphy (Lezius, 2000) and a verb subcategorization frame database (Buchholz, 1996) were used to automatically generate a large lexicon from a list of lemmas in EAGLE. The lexicon was extended by hand to cover tokens these tools were unable to analyze.
Chapter 10: Detecting Conflicts and Determining Explanations for Learner Errors

**QUICKXPLAIN**, introduced in section 6.2.2, is used to identify conflicting constraints in XDG CSPs that lead to detailed error diagnoses. Section 10.1 discusses the role of **QUICKXPLAIN** in error detection and diagnosis for the Fledgling grammar checker and section 10.2 presents the customizations for XDG CSPs. Section 10.3 describes the implementation of **QUICKXPLAIN** for XDG using the constraint solver Gecode (Gecode Team, 2006) with example output. Sections 10.4 – 10.5 describe the mapping between conflicting constraints and EAGLE types and how constraint ordering can be used to prefer particular explanations.

### 10.1 **QUICKXPLAIN** in Error Detection and Diagnosis

An error is *detected* in a CSP when a constraint solver determines that the CSP is unsatisfiable and therefore has no solutions. Beyond an *unsatisfiable* message, a traditional constraint solver is unable to provide any further information about type of error found or to yield any partial solutions (i.e., partial parses) to the problem. An error is *diagnosed* when a reason for the failure is discovered and can be provided as feedback for a human user. Once errors are diagnosed, the constraints causing the errors can be relaxed so that it is possible to additionally provide one or more partial solutions to the original CSP.
An error can be detected by determining whether a CSP is satisfiable. If so, there are no errors and there is no need for further processing. If the CSP is unsatisfiable, a conflict detection algorithm such as QUICKXPLAIN can be used to determine which constraints are causing the CSP to be unsatisfiable. As explained in section 6.2.2, the QUICKXPLAIN algorithm finds a set of conflicting constraints by recursively examining subsets of constraints from an overconstrained CSP. If the CSP is satisfiable, there are no conflicts and QUICKXPLAIN returns no conflict. If the CSP is unsatisfiable, QUICKXPLAIN returns a set of conflicts that form a minimal conflict, i.e., a CSP containing these constraints is never satisfiable.

Although the set of constraints found by QUICKXPLAIN is a minimal conflict, it is not necessarily the only conflict in the original CSP. Removing or relaxing the constraints in the first conflict set from the CSP may not necessarily lead directly to a satisfiable CSP. In order to find all minimal conflicts (i.e., all errors) in a CSP, QUICKXPLAIN can be used iteratively. QUICKXPLAIN finds a minimal conflict, these constraints are relaxed, and QUICKXPLAIN searches again for a conflict.

The output of applying QUICKXPLAIN iteratively is an ordered list of conflict sets and the set of solutions to the CSP that contains all the constraints from the original CSP except those in conflict sets. The constraints in the set of conflicts can point to a reason behind the failure and lead to an appropriate diagnosis, such as an error type from an error typology, and the solutions to the relaxed CSP can inform feedback provided to the user.
10.2 Customizing QuickXplain for XDG CSPs

When diagnosing errors in ungrammatical learner sentences using XDG CSPs, the goal is to search for government, agreement, or order constraints that have been violated. Many properties of XDG CSPs, such as the appearance of the multigraph parse or the node records that contain lexical information, do not need to come into consideration. The CSP representation includes low-level constraints that specify everything from the position of the word in the sentence to appearance of the parse (labeled tree, labeled directed acyclic graph, etc.) In order to ensure that the CSP solutions are indeed XDG parses, constraints that define the appearance of the parse or copy the lexical entry for a word into the node record should always be enforced. If QUICKXPLAIN considered conflicts among these constraints, it would mean looking for errors the structure of an XDG analysis rather than in the grammatical constraints that Fledgling is targeting.

The QUICKXPLAIN algorithm already has a natural way of handling constraints that is easy to use for XDG CSPs: the main QUICKXPLAIN method expects to be provided with a set each of background and foreground constraints. In the initial call to QUICKXPLAIN, an XDG CSP can already be divided into hard constraints, which should never be relaxed, and soft constraints, which can lead to an error diagnosis if they are not satisfied. Section 10.2.1 defines the division of XDG CSP constraints into hard and soft constraints for the purposes of error diagnosis with QUICKXPLAIN.
\((\text{word2} \in \text{word1.daugthers}) \Rightarrow ((\text{rel} \in \text{relations}) \Rightarrow (\text{word1.attr} == \text{word2.attr}))\)

Figure 10.1: Core Constraints in XDG Agreement Principle

10.2.1 Hard and Soft Constraints

In order to diagnose the error types in the EAGLE error typology, only a small fraction of constraints needs to be considered as having the potential to be involved in a conflict related to a grammatical error. As the running time of \textsc{QuickXplain} depends on the number of constraints under consideration,\(^{30}\) reducing the number of constraints as much as possible makes \textsc{QuickXplain} for XDG CSPs faster.

As explained above, many XDG constraints enforce the appearance of an XDG parse and should always be considered as hard constraints. Constraints that correspond to relevant grammatical error types such as agreement constraints should be soft constraints. Figure 10.1, repeated from Figure 5.7, shows the constraints involved in the agreement principle. Recall from chapter 5 the agreement principle uses five constraints in the CSP to express the high-level notion of agreement. If we wish to relax this agreement constraint because of an error, we could consider all five constraints from the grammar in turn, however it is sufficient to consider only the outermost implication, which is underlined in Figure 10.1.

As a result, when looking for an agreement error between \textit{word1} and \textit{word2} in a \textit{rel} relation, \textsc{QuickXplain} needs only to consider the underlined implication. If this constraint is

\(^{30}\)The running time is \(O(n \times \log(k + 1) + k^2)\), where \(n\) is the number of constraints and \(k\) is the size of conflict set.
not part of a conflict set, there is no agreement error; if it is part of a conflict set due to an agreement error, removing the implication relaxes the entire high-level agreement constraint so that a partial analysis can be found.

For each type of principle involved in an XDG CSP, where numerous low-level constraints are used to express high-level grammatical constraints, it is possible to pinpoint a small number of low-level constraints, which will be called key constraints, that are the outermost or central constraints for a principle.

For agreement and agreementSubset, there is one key constraint for each pair of words and relation under consideration. For example, a constraint on person/number agreement between subjects and verbs is specified as an agreement between the agr variable for any pair of words in subj relation. Because it is not known in advance which words might appear in a subj relation, the XDG CSP states the constraint on every pair of words in the sentence. If a pair is in the subj relation, they should agree; otherwise, the constraint does not apply.

For out, which checks for the presence of required arguments, there are two constraints for each type of argument or adjunct that can appear as the child of each word in the sentence. For government, which checks the form of arguments that are present, there is one key constraint for each type of argument or adjunct (reflexive, NP, argument PP, adjunct PP, adverb, etc.) expected or permitted by each word. For order and orderPlus, there is one key constraint for each word that specifies the order for it and its immediate children. For relative, there is one constraint for each word in the sentence that is relevant when this word is a relative pronoun.
Restricting the soft constraints to these key constraints greatly reduces the size of the search space and thus the running time for QUICKXPLAIN. As an example, the nine-word sentence *ich habe einen Anzug, sechs Baumwollhemden, und drei Mützen* ‘I have a suit, six cotton shirts, and three caps’ contains 311,500 variables and 415,000 constraints. When the soft constraints are restricted to key constraints, there are only 91,000 constraints that QUICKXPLAIN needs to consider.

10.2.2 Processing Stages

Multiple stages of processing, which each gradually allow more soft constraints, allows Fledgling to report more precise error diagnoses. Particularly problematic is the out constraint, which checks that each word has the correct arguments and adjuncts. This constraint is involved in missing and extra word errors. When words are missing, it is easy to identify which word is missing in which argument, e.g., when a noun that requires a determiner does not have one and subsequently only that noun is reported as containing an error. When there is an extra word, there is no suitable place to attach the word in the parse, so QUICKXPLAIN returns a conflict set that includes all possible arguments expected by all words in the sentence, since there is no single preferred solution.

When there are extra words, relaxing the out constraints allows for any word in the sentence to be the parent of any other word in any relation, so the CSP solutions are not useful parses of the input sentence. Because of this, out constraints should not be relaxed until all other errors in the sentence have been identified. Otherwise, most errors could be diagnosed as an out violation, since the very nature of an ungrammatical sentence is that some words do not fit into the parse. The agreement, government, and order constraints
can pinpoint errors very precisely and relaxing one results in minimal changes to the set of parse solutions.

Since the main concern is limiting out constraint relaxation, Fledgling uses two stages. In the first stage, it looks for errors related to government, agreement, relative, and order principles. If no solutions are found when these constraints can be relaxed, it adds the out constraints to the set of soft constraints.

10.3 Implementation in Gecode

The QUICKXPLAIN algorithm as described in section 6.2.2 was implemented using the general-purpose constraint solver Gecode (Gecode Team, 2006, version 3.4.0) by extending Gecode’s FlatZinc interpreter. FlatZinc is a low-level CSP modeling language that is supported by many constraint solvers (Nethercote et al., 2007). The FlatZinc file specifies variables, constraints, and solver instructions such as XDG’s custom constraint distribution.

The existing Gecode FlatZinc interpreter reads in the CSP in FlatZinc format and immediately posts (i.e., adds and propagates) the parsed constraints in the computation space, the basic representation of a CSP used by Gecode’s search engines.Undoing propagation is difficult and not necessary for traditional constraint solving, so Gecode has no mechanism to remove constraints from a computation space, which is a problem for QUICKXPLAIN, since QUICKXPLAIN needs to examine CSPs containing subsets of the original constraints.

In order to make it possible for QUICKXPLAIN to create CSPs using subsets of constraints, the Gecode FlatZinc interpreter was modified to create a constraint store as the constraints
are read in. Each computation space used by the new FlatZinc-QuickXplain interpreter contains a constraint store. Instead of posting the constraints as they are read in, FlatZinc-QuickXplain parses them and stores them in the constraint store. Each time QuickXplain needs to examine a new subset of constraints, the original space is cloned and the desired constraints are posted.

QuickXplain rarely examines a CSP with the same subset of constraints twice, so experiments with a cache of computation spaces showed that maintaining a cache of spaces does not improve the running time. The cloning operation is so much more expensive than constraint posting/propagation, in part because of the time taken to clone the constraint store, that creating clones to store in a cache takes more time than cloning the original space and reposting constraints as needed.

As presented in Junker (2001), QuickXplain’s search for a constraint that leads to a failure is performed linearly (cf. lines 4–8 in Figure 6.5. Foreground (i.e., soft) constraints are added one by one until the problem has no solutions. Posting a constraint is a fast operation, however the repeated checks for satisfiability were quite slow for problems with a large number of constraints like XDG CSPs. To improve the running time, the linear search from the original algorithm was replaced with a recursive binary search. The binary search requires more cloning than the linear search, which could keep posting constraints to the same space, but it requires many fewer satisfiability checks.

An extended version of the basic FlatZinc-QuickXplain algorithm, named QuickXplain-XDG, is created to add the customizations described in section 10.2: dividing constraints into hard and soft constraints and associating soft constraints with particular tokens and error types. The FlatZinc export generated by the XDG Development Toolkit (Debusmann
et al., 2004) includes information that links the low-level variables found in the CSP to features and feature values in the multigraph model. These link low-level constraints to high-level principles, which are in turn linked to error types as described in section 10.4.

When Q\textsc{UICK}X\textsc{PLAIN}-XDG finds a conflict set, it outputs the relevant constraints and associated variables and principles, removes these constraints from the original CSP, and checks again for conflicts. It will loop through conflict detection and constraint relaxation steps until the CSP is satisfiable.

10.3.1 Example Output for Q\textsc{UICK}X\textsc{PLAIN}-XDG

In this section, examples from the training will illustrate the output of Q\textsc{UICK}X\textsc{PLAIN}-XDG. A sentence from the training data is shown in (29). It contains an agreement error in the noun phrase \textit{einem Kneipe}.

(29) * Er war in einem Kneipe.
   he was in a\textsubscript{masc/neut,dat,sg} bar\textsubscript{fem,dat,sg}

The output from Q\textsc{UICK}X\textsc{PLAIN}-XDG for this sentence, shown in Figure 10.2, includes one conflict set for this sentence that specifies the XDG principle (dimension, principle type, principle name specified in the grammar) and information about the tokens and features involved. The token/feature information reports the fact that agreement was not satisfied when token 4 is analyzed as the \texttt{det} daughter of token 5. The low-level constraint, which is copied from directly from the FlatZinc input, is also shown for reference. It corresponds to the implication key constraint described for agreement in section 10.2.1.

\footnote{The XDG FlatZinc output also contains a number of unused variables, which FLATZINC-Q\textsc{UICK}X\textsc{PLAIN} automatically removes. When present, they lead to a large number of equivalent solutions and increase the running time unnecessarily.}
Conflict #1:
Constraint: 29634 constraint bool_right_imp(fdbool14133, fdbool14136, true);
Principle: id principle.agreement agreement.det-adj-noun
Tokens: 4, 5 (id model daughtersL det)

Figure 10.2: QUICKXPLAIN-XDG Output for (29)

An example with a selection error (S1.A.i) is shown in (30). The verb *zeige* ‘show’ expects one accusative and one dative argument rather than two accusative arguments. The QUICXPLAIN-XDG output is shown in Figure 10.3.

(30) * Ich zeige sie sie.
   I show\(_{1,sg}\) her\(_{acc}\) her\(_{acc}\)

This conflict set shows that the *sie* that does not fit into *zeige*’s expected arguments could have been intended as the subject of *zeige*, in which case there is a subject-verb agreement error, or that it could possibly be the missing dative *iobj* argument, in which case there is a selection error in the case of an argument. After these constraints are relaxed, one of the parses found is shown in Figure 10.4. Analyzing *sie* as the subject would lead to further errors for *ich*, so the parses found analyze one of the *sie* tokens as an indirect object.

An example with a word order error in the position of the finite verb is given in (31). The QUICXPLAIN-XDG output is provided in Figure 10.5. An error in the position of the finite verb is equivalent to a *Mittelfeld* ordering error because, in effect, the finite verb appears in the middle of the *Mittelfeld* rather than in the position between the *Vorfeld* and *Mittelfeld*.

   after the shopping I eat in the restaurant
Conflict #1:

Constraint: 14281 constraint bool_right_imp(fdbool3753, fdbool3756, true);  
Principle: id principle.agreement agreement.subj-verb  
Tokens: 2 (id model daughtersL subj), 3

Constraint: 14576 constraint bool_right_imp(fdbool3989, fdbool3992, true);  
Principle: id principle.agreement agreement.subj-verb  
Tokens: 2 (id model daughtersL subj), 4

Constraint: 81386 constraint bool_right_imp(fdbool55775, fdbool55776, true);  
Principle: id principle.government government.casewithargtype-other  
Tokens: 2 (id model daughtersL iobj), 3

Constraint: 81563 constraint bool_right_imp(fdbool55893, fdbool55894, true);  
Principle: id principle.government government.casewithargtype-other  
Tokens: 2 (id model daughtersL iobj), 4

Figure 10.3: QUICKXPLAIN-XDG Output for (30)

Figure 10.4: Parse after Constraint Relaxation for (30)
Conflict #1:

Constraint: 21101 constraint array_seq([fs3667, fs3668, fs3669, fs3670, s3671, fs3672, fs3673]);
Principle: lp principle.orderPlus mforderpro

Constraint: 21963 constraint array_seq([fs3715, fs3716, fs3717, fs3718, fs3719]);
Principle: lp principle.orderPlus mforder

Figure 10.5: QUICKPLAIN-XDG Output for (31)

Relaxing the Mittelfeld ordering constraints (which are separated into pronoun and non-pronoun ordering) in the conflict allows the word ich to appear before the finite verb in the parse shown in Figure 10.6.

A final example with two errors is shown in (32). In this example, there is a subject-verb agreement error between ich and hörst and a determiner-adjective-noun agreement error between meinem and Neffe.

   I hear_{2,sg} often from my_{dat} Neffe_{nom}

10.4 Associating Conflicts with Explanations

The QUICKPLAIN-XDG output includes one or more sets of conflicting constraints and information about the principles and tokens associated with each constraint. To map this information to error types in the EAGLE error typology, each constraint is associated with one or more constraints in the typology. The mapping is shown in Table 10.1.
Figure 10.6: Parse after Constraint Relaxation for (31)

Conflict #1:
Constraint: 20439 constraint bool_right_imp(fdbool4873, fdbool4876, true);
Principle: id principle.agreement agreement.subj-verb
Tokens: 1, 2 (id model daughtersL subj)

Conflict #2:
Constraint: 92309 constraint bool_right_imp(fdbool62369, fdbool62372, true);
Principle: id principle.agreement agreement.npcase
Tokens: 5, 6 (id model daughtersL det)

Conflict #3:
Constraint: 105399 constraint bool_right_imp(fdbool72841, fdbool72844, true);
Principle: id principle.agreement agreement.refl
Tokens: 1, 2 (id model daughtersL subj)

Conflict #4:
Constraint: 188275 constraint bool_right_imp(fdbool133489, fdbool133490, true);
Principle: id principle.government government.casewithprep
Agreement: 4 (id model daughtersL piobj), 6

Figure 10.7: QUICKXPLAIN-XDG Output for (32)
Figure 10.8: Parse after Constraint Relaxation for (32)

Most constraints can be mapped to a single error type, but several (incorrect case in PP argument, missing extra arguments) are not specific enough on their own to point to a single error type. In these cases, the token information and information from the relaxed parse can be used to distinguish the error types in most cases. Missing and extra arguments, which require further classification, are discussed in the following section.

10.4.1 Missing and Extra Arguments

Missing and extra arguments require additional analysis because when the out principle is not satisfied, it could be due to missing arguments, extra arguments, or incorrect argument
<table>
<thead>
<tr>
<th>XDG Principle</th>
<th>Error Code(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>government</td>
<td>S1.A.i</td>
<td>nominative case for verb subject</td>
</tr>
<tr>
<td>government</td>
<td>S1.A.i</td>
<td>non-nominative case for acc/dat/gen objects</td>
</tr>
<tr>
<td>government</td>
<td>S1.A.ii</td>
<td>case involving conjunction</td>
</tr>
<tr>
<td>government</td>
<td>S1.A.iii/iv/v, S2.A.i, S3.B.iii, S4.A.iii</td>
<td>incorrect preposition in PP argument</td>
</tr>
<tr>
<td>government</td>
<td>S1.C.iii</td>
<td>incorrect case in PP argument</td>
</tr>
<tr>
<td>government</td>
<td>S2.A.i</td>
<td>reflexive case selection</td>
</tr>
<tr>
<td>government</td>
<td>S6.A.ii</td>
<td>coordinating conjunction mismatch</td>
</tr>
<tr>
<td>agreement Subset</td>
<td>S1.A.vi</td>
<td>haben/sein error</td>
</tr>
<tr>
<td>agreement</td>
<td>S1.A.xi</td>
<td>subject selection for raising verbs</td>
</tr>
<tr>
<td>agreement</td>
<td>S1.B</td>
<td>impossible separable prefix</td>
</tr>
<tr>
<td>agreement</td>
<td>A2.A, A2.B</td>
<td>agreement within NP in gender/number</td>
</tr>
<tr>
<td>agreement</td>
<td>A2.C</td>
<td>determiner-adjective-noun agreement in case</td>
</tr>
<tr>
<td>agreement</td>
<td>A2.D</td>
<td>determiner-adjective-noun agreement in definiteness</td>
</tr>
<tr>
<td>agreement</td>
<td>A4.A</td>
<td>number agreement for subject-predicate with copula</td>
</tr>
<tr>
<td>agreement</td>
<td>A6</td>
<td>subject-reflexive agreement</td>
</tr>
<tr>
<td>relative</td>
<td>A3</td>
<td>relative pronoun-antecedent agreement, number of relative pronouns in a relative clause</td>
</tr>
<tr>
<td>order</td>
<td>O1, O2</td>
<td>Verbal complex order</td>
</tr>
<tr>
<td>order</td>
<td>O1, O4</td>
<td>Mittelfeld non-pronouns</td>
</tr>
<tr>
<td>order</td>
<td>O1, O5</td>
<td>Vorfeld order</td>
</tr>
<tr>
<td>order</td>
<td>O2</td>
<td>Nachfeld (also non-finite verb)</td>
</tr>
<tr>
<td>order</td>
<td>O1, O4</td>
<td>Mittelfeld pronouns</td>
</tr>
<tr>
<td>order</td>
<td>O6, O7</td>
<td>prepositional phrase and noun phrase order</td>
</tr>
<tr>
<td>order</td>
<td>O</td>
<td>incorrect order in coordinate phrase</td>
</tr>
</tbody>
</table>

Table 10.1: Mapping Between Principles and EAGLE Error Types
types, which are simply cases with one missing and one extra argument. However, information about the tokens involved can be used to determine whether an error involves an extra or missing word. In the case of missing words, only a single token is involved in the detailed error type can be determined by examining the lexical entry of the word missing an argument and the token features returned by QUICKXPLAIN-XDG.

An example with a missing determiner is shown in (33) and the QUICKXPLAIN-XDG output is shown in Figure 10.9. The output shows that only a single token (#3) is involved and the token features show that it either requires a determiner or relative pronoun determiner (dessen or deren). Since Freundin is a noun, this indicates to the error type S3.A.i.

(33) * Petra und Freundin gehen gern joggen.
       Petra and friend go like to jogging
       ‘Petra und friend like to go jogging.’

When there is an extra word, the output constraints reported in the conflict set apply to every word in the sentence as the constraint solver tries to find a place to attach the extra word in the parse. In these cases, it is difficult to pinpoint which word is the extra word. “Incorrect complement type” errors are simply a combination of a single missing word and a single extra word error.

**Missing Main Finite Verb**

A special case for missing word errors is the sentence selection error type (S5b.1), which corresponds to a missing finite verb in the main clause. As described in section 7.3.2, the EAGLE annotation guidelines require annotators to check for the presence of a finite verb in the main clause to insure that the student’s response is a complete sentence. A phrase such
Conflict #1:
Constraint: 4556 constraint set_card(fs1134, fd289);
Principle: id principle.out idout
Tokens: 3 id model daughtersL det

Constraint: 4557 constraint set_in(fd289, fs893);
Principle: id principle.out idout
Tokens: 3 id entry out

Constraint: 4588 constraint set_card(fs1150, fd305);
Principle: id principle.out idout Out
Tokens: 3 id model daughtersL rdet

Constraint: 4589 constraint set_in(fd305, fs909);
Principle: id principle.out idout Out
Agreement: 3 id entry out rdet

Figure 10.9: QUICKXPLAIN-XDG Output for (33)

Figure 10.10: Parse after Constraint Relaxation for (33)
as schlechtes Wetter ‘bad weather’ can be parsed as a noun phrase by the EAGLE grammar and thus QUICKXPLAIN-XDG does not find any conflicts, but it does not conform to the expected full sentence response. To detect and diagnosis this error type, QUICKXPLAIN-XDG adds an extra conflict containing the error S5b.1 if a main clause finite verb is not found in the parse. The finite verb in the main clause can be identified easily by looking for a token assigned the v12 label in the lp dimension output.

10.5 Constraint Ordering and Learner Modelling

The XDG Development Toolkit ranks the XDG principles so that they are posted in an efficient order for the constraint solver. The key constraints for error diagnosis from the previous section are posted in the following order: out, order, relative, agreement / government. This ordering makes intuitive sense: it is first relevant which arguments each word expects and which order they are expected to be found; once the possible types and orders of arguments are established, agreement and government constraints narrow down the role of each word in the sentence. The constraint solver becomes very inefficient if this ordering is modified because constraint propagation cannot make any progress when relevant information is not present. For example, if constraints on subject-verb agreement are posted -2009 this constraint might apply to and is unable to narrow down the search space.

As explained in section 6.2.2, the order of the constraints in the CSP determines which conflict the QUICKXPLAIN algorithm returns. QUICKXPLAIN searches for the earliest constraint that leads to failure and returns the conflict set containing that constraint. The diagram in Figure 10.11 depicts the constraints as a series of boxes ordered from left to
Figure 10.11: Constraint Ordering and QuickXplain

right. Two minimal conflict sets are shown in the boxes numbered “1” and “2”. QuickXplain searches for the earliest point at which the constraint solver fails to find a solution, which is immediately after the final constraint in conflict set #1 has been posted in this example. Because a constraint in conflict set #1 led to the failure, QuickXplain returns the three constraints in conflict #1. After the constraints in conflict #1 are relaxed, running QuickXplain again would return conflict #2.

For different activities or different learner models, the agreement and government constraints, which are posted at the same stage by the constraint solver, can be reordered to prefer certain error types over others. The process of converting QuickXplain-XDG output into learner feedback would also naturally involve re-ranking or ignoring certain error messages (cf. Heift, 2003b). In the EAGLE grammar described in the previous chapter, the agreement and government constraints have been ordered by descending error frequency in EAGLE so that more frequent errors in EAGLE are ranked higher in QuickXplain-XDG’s output for sentences with multiple errors.
10.6 Summary

The QUICKXPLAIN algorithm for detecting conflicts in overconstrained CSPs is used to detect and diagnose errors in XDG CSPs. When conflicting constraints are found, those constraints are relaxed until one or more parses can be found. The conflicting constraints, which are each associated with XDG principles, are mapped directly to error types in the EAGLE error typology in order to provide an error diagnosis. The QUICKXPLAIN-XDG algorithm divides the constraints in an XDG CSP into hard and soft constraints, finds conflicting constraints, and returns error diagnoses from the EAGLE error typology along with the parses found after conflicting constraints have been relaxed.
Chapter 11: Evaluation

The Fledgling grammar checker is presented in its entirety and evaluated using the EAGLE corpus. The evaluation considers whether Fledgling can correctly distinguish grammatical and ungrammatical sentences and how accurately Fledgling identifies error types at different levels of the EAGLE error typology.

11.1 Overview

Figure 11.1 summarizes the processing steps for the Fledgling grammar checker. First, the learner sentence is tokenized using the Dipper tokenizer (Dipper, 2008) and spelling errors are semi-automatically corrected (see section 7.3.1). The Extensible Dependency Grammar Development Toolkit (XDK, Debusmann et al., 2004) reads in the EAGLE grammar and generates a constraint satisfaction problem for the parse of the learner sentence. QUICKXPLAIN-XDG searches for constraint conflicts in this CSP and diagnoses errors by mapping the conflicting constraints to error types in the EAGLE error typology. QUICKXPLAIN-XDG outputs a set of error diagnoses and a set of parses that were found after conflicting constraints were relaxed.

The evaluation presented below focuses on the error diagnoses returned by QUICKXPLAIN-XDG. It presumes that the tokenization contains no errors and that the spell checker has
Figure 11.1: Overview of the Fledgling Grammar Checker
either automatically or through interaction with the learner produced perfect corrections
for any non-word spelling errors. In an online workbook, the error diagnoses and parses
from QUICKXPLAIN-XDG would be provided to a feedback module that generates appro-
priate feedback for the learner. The feedback module, which is outside the scope of this
thesis, could use information from the activity model and learner model to rank, filter, and
reformulate feedback for presentation to the learner.

The evaluation will consider all error types from the EAGLE error typology aside from
punctuation errors since the EAGLE grammar currently ignores punctuation and the con-
ventions of punctuation are separate from modeling grammar. QUICKXPLAIN-XDG can
currently only diagnose those error types specified in the principle–error type mapping in
section 10.4. There are many additional error types annotated in the EAGLE corpus, in
particular word form errors, that Fledgling cannot diagnose. The error types covered by
the principle–error type mapping account for 75% of non-punctuation errors annotated in
EAGLE.

11.2 Test Data

The evaluation of the Fledgling grammar checker focuses on the workbook subcorpus,
since the workbook activities have a more limited vocabulary than the essays. This is in-
tended to limit the amount of effort required to extend the EAGLE grammar and lexicon to
cover a large number of test sentences. The test data is divided into two sections, Test10
and Test90, containing 10% and 90% of the test sentences respectively. Full lexical cover-
age for Test10 is guaranteed in the EAGLE grammar while the Test90 evaluation includes
only those sentences with lexical coverage from existing resources.
For 10% of the workbook test corpus (Test10), which contains 318 sentences, a sorted list of tokens was inspected in order to make sure the EAGLE grammar provides coverage for all tokens. The sorted token list does not reveal which sentences or which types of errors from the corpus appear in the test data, but it is important to note that the test data is not strictly unseen. The test data was sequestered shortly after the annotation was complete (section 7.5) and before the EAGLE grammar development started, but the entire EAGLE corpus was examined by the author during the error annotation process and for the error analysis in chapter 8.

For the remaining 90% (Test90), 9% of the 2854 sentences were excluded from the evaluation because of a lack of lexical coverage in the EAGLE grammar, which is limited by the entries that could be generated automatically using Morphy and the Buchholz verb database (see section 9.2).

For the Test10 results, three nonsense responses, which are not currently automatically detected by Fledgling, (asdfsds, j, k) are ignored in the evaluation. When disregarding punctuation and sentence-initial capitalization errors,\(^{32}\) 244 of the 315 sentences (77%) are grammatical according to the EAGLE error annotation.

### 11.2.1 Time Limits and Processing Times

The XDG CSPs are of a non-trivial size for a conflict detection method such as QUICK-XPLAIN. The plain-text FlatZinc representation of an EAGLE XDG CSP is on average

---

\(^{32}\)Sentence-initial punctuation errors are those where the learner failed to capitalize the first letter in a sentence. These errors would be trivial to detect automatically. Otherwise, non-initial capitalization errors, which in German can change the part-of-speech of a word, pose significant challenges for Fledgling or any German grammar checker.
17MB, which is time-consuming to read in, and the Test10 sentences contain on average 170,000 variables and 230,000 constraints per sentence. The Fledgling grammar checker was evaluated using a desktop computer with an Intel Core2 Duo 3GHz processor with 8 GB RAM. QUICKXPLAIN-XDG was restricted to a maximum of ten minutes per sentence. For 9% of the Test10 sentences, QUICKXPLAIN-XDG returns a timeout, which is analyzed as an ungrammatical sentence with an empty error diagnosis. QUICKXPLAIN-XDG typically runs into problems with extra words, as mentioned in section 10.2.2, with coordination, with longer sentences, and with numerous, overlapping errors. For the remaining 91% of the Test10 sentences, for which QUICKXPLAIN-XDG does return an error analysis, QUICKXPLAIN-XDG required an average of 35 seconds per sentence. The shortest processing time for a single sentence was 0.25 seconds, the longest was 595 seconds, and the median processing time was 9 seconds.

11.3 Evaluation

The evaluation compares the EAGLE annotation and the Fledgling grammar checker output on the sentence level, as in the interannotator agreement evaluation (see section 7.4). For each sentence, the EAGLE annotation and the Fledgling output provide a set of errors found in the sentence. Four evaluation metrics will be used: grammaticality, accuracy, precision, and recall by top-level error category, intersection of error types, and the MASI distance metric (Passonneau et al., 2006, see section 7.4).
11.3.1 Grammaticality

The grammaticality metric determines whether the EAGLE annotators and Fledgling agree on the grammaticality of a sentence. Punctuation errors and sentence-initial capitalization errors are ignored, but all other errors, including word form errors that Fledgling does not currently target are taken into account. If they agree the score is 1. Otherwise, the score is 0.

11.3.2 Accuracy, Precision, and Recall for Top-Level Error Categories

The grammar checker focuses on three top-level error categories: selection, agreement, and word order. The evaluation by error category calculates the accuracy, precision, and recall on the sentence level for the presence or absence of these three categories in turn.

11.3.3 Intersection

The intersection metric checks whether Fledgling’s output and the errors annotated in EAGLE intersect. As in the interannotator agreement calculations, the hierarchical nature of the error typology makes it possible to compare error descriptions at different depths. The first level is the main category type (e.g., S), the second level is one level deeper (e.g., S1), the third level is another level deeper (S1.A), and the fourth level includes the complete error description. If the sentence contains errors and Fledgling finds at least one of these, the score is 1. If there is no overlap in the errors found by Fledgling and the annotated errors, the score is 0. If the sentence contains no errors and Fledgling finds no errors, the score is 1.
11.3.4 MASI

The MASI (measuring agreement on set-valued items) distance metrics takes the size and intersection of the two sets into account:

\[ MASI_{Dist}(S_1, S_2) = \frac{|S_1 \cap S_2|}{\max(|S_1|, |S_2|)} \]

Unlike the intersection metric, MASI penalizes the grammar checker for returning error types that the annotators did not find. The MASI metric is also reported for the full range of depths of the error annotation.

11.4 Results and Analysis

The following sections present and analyze Fledgling’s performance on the Test10 and Test90 test sets.

11.4.1 Grammaticality

The grammaticality scores are shown in Table 11.1. Fledgling’s grammaticality judgements are correct for 80% of the Test10 sentences and a similar percentage of Test90 sentences.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Test10 Score</th>
<th>Test90 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grammaticality</td>
<td>0.804</td>
<td>0.770</td>
</tr>
</tbody>
</table>

Table 11.1: Grammaticality Scores
The sources of the errors in grammaticality judgement are summarized in Table 11.2. In 65% of the errors in grammaticality judgement (38 sentences), Fledgling reported errors in sentences that were grammatical according to the EAGLE annotation. Of the 58 errors in grammaticality judgement, 44 can be traced directly to problems in the EAGLE annotation or in Fledgling, which can be addressed in the future:

![Table 11.2: Sources of Errors in Grammaticality Judgement for Test10](image)

**Annotation errors**  For 7 sentences (2% of Test10), the EAGLE annotation was in fact incorrect and Fledgling returned the correct judgement. This indicates that Fledgling will be a useful tool for verifying EAGLE error annotation in the future.

**Grammar coverage**  In 25 sentences, a lack of coverage in the EAGLE grammar led to the discrepancy. Several frequent sources of problems were support for coordination, missing verb subcategorization frames, and a missing adjunct preposition (*seit*). The EAGLE grammar development is ongoing and it is simple to add additional subcategorization
frames to the grammar. Analysis of coordination in dependency grammars poses well-known difficulties that require more complicated extensions to the grammar (cf. Nivre, 2005; Osborne, 2008; Dickinson and Ragheb, 2011).

**Capitalization normalization** Three errors are due to Fledgling’s capitalization normalization (see section 9.3.4) which did not normalize Könnten, Morgen, and Wegen at the beginning of the sentence. An improved POS-based capitalization normalization method (or a grammar that does not require this preprocessing step) should eliminate these problems in the future.

**Main clause detection** Nine errors are due to Fledgling’s main clause detection algorithm, which checks for a verb-first or verb-second verb in the first parse returned by QUICKXPLAIN-XDG. In some cases, the EAGLE grammar provides a correct verb-final parse for clauses that could appear in subordinate phrases such as Käthe reitet ‘Käthe rides (horses)’, which could be a main clause or appear in a subordinate clause such as dass Käthe reitet ‘that Käthe rides (horses)’. In other cases, the main clause detector rejects well-formed verb-final subjunctive clauses such as Wenn Häuser nur weniger kosteten! ‘If only houses cost less!’ In all cases, rather than only examining the first parse, Fledgling should examine all parses returned by QUICKXPLAIN-XDG rather than relying on the first parse.

33 One of the coordination errors in Test10 is due to unlike coordination: *Meine Lieblingsjeans sind blau und bei American Eagle* ‘My favorite jeans are blue and from American Eagle’ and there are several similar instances of unlike coordination in the EAGLE training data.
<table>
<thead>
<tr>
<th></th>
<th>Error Type</th>
<th>Test10</th>
<th>Test90</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy</strong></td>
<td>Selection</td>
<td>0.819</td>
<td>0.816</td>
</tr>
<tr>
<td></td>
<td>Agreement</td>
<td>0.883</td>
<td>0.898</td>
</tr>
<tr>
<td></td>
<td>Word Order</td>
<td>0.902</td>
<td>0.877</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>Selection</td>
<td>0.409</td>
<td>0.395</td>
</tr>
<tr>
<td></td>
<td>Agreement</td>
<td>0.400</td>
<td>0.494</td>
</tr>
<tr>
<td></td>
<td>Word Order</td>
<td>0.100</td>
<td>0.099</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>Selection</td>
<td>0.878</td>
<td>0.788</td>
</tr>
<tr>
<td></td>
<td>Agreement</td>
<td>0.741</td>
<td>0.841</td>
</tr>
<tr>
<td></td>
<td>Word Order</td>
<td>0.429</td>
<td>0.358</td>
</tr>
</tbody>
</table>

Table 11.3: Evaluation by Error Type

The remaining errors are more difficult to analyze due to timeouts in the error diagnosis step, so the source of the errors is not clear in all cases. The errors due to timeouts are typically longer sentences (>10 tokens) with coordination, relative clauses, or extra words that lead to long processing times for QUICKXPLAIN-XDG.

11.4.2 Accuracy, Precision, and Recall by Top-Level Error Category

Table 11.3 shows accuracy, precision, and recall by top-level error category for Test10 and Test90. Fledgling accurately identifies the presence of selection, agreement, and word order errors in 82–91% of the Test10 sentences. Selection errors cause the most difficulty with only 82% accuracy in Test10 while the presence of agreement and word order errors is identified with approximately 90% accuracy. The recall for Fledgling for selection errors is quite good at 88% for Test10. Recall for agreement errors is also good at 74% while recall for word order errors is much lower at 43%.
The precision, approximately 40% for selection and agreement and 10% for word order errors, shows that Fledgling often proposes many more errors than are present in the EAGLE annotation. Fledgling often reports a bundle of errors that represents a single error from the perspective of the learners and the EAGLE annotators (cf. Menzel and Schröder, 1998). To improve these results, the list of errors needs to be ranked and filtered before being displayed to the learner. In particular, Fledgling often proposes word order errors in combination with certain types of noun phrase agreement errors, since rearranging determiners or prepositions is sometime sometimes one possible, if strongly dispreferred, solution to gender or case agreement errors.

Example (34) shows one sentence where Fledgling diagnoses a related bundle of errors while an EAGLE annotator would annotate only one.

(34) * Wenn die Mieten in dieser Stadt nur nicht zu hoch wäre!
   if the rents in this city only not too high were
   ‘If only the rents in this city were not too high!’

According to the EAGLE annotation guidelines, this sentence contains a single error, a subject-verb agreement error between die Mieten$_{pl}$ and wäre$_{sg}$. Fledgling, however, reports the following set of errors:

- subject-verb agreement error in person and/or number
- determiner-adjective-noun agreement error in person and/or number
- word order error in a noun phrase or prepositional phrase

Fledgling recognizes that the subject-verb agreement error could also be analyzed as a determiner-noun agreement error within the subject, since die could also be singular and
making *Mieten* singular would fix the error. Likewise, reordering the preposition could allow *dieser Stadt* (with further corrections) to be the subject of the sentence instead of *Mieten*. From the learner’s perspective, the word order error is quite far-fetched, but *QUICKXPLAIN-XDG* does not currently have a mechanism to prefer one error type over another. In particular, both the subject-verb agreement error or the determiner-adjective-noun agreement error could be plausible depending on the strengths of the learner or the design of the activity.

The fact that *QUICKXPLAIN-XDG* reports such sets of errors leads in part to the low precision. To improve in this area, either Fledgling needs to incorporate more ranking and filtering or the feedback module needs to recognize that it will need to take over this task. Precision is particularly important for ICALL systems so that learners are not misled into thinking a grammatical sentence is ill-formed (cf. Tschichold, 1999, 2003).

A closer analysis of the Test10 output shows that ranking and filtering will lead to improved results. If *QUICKXPLAIN-XDG*’s output for Test10 is filtered by hand using the principles established in the EAGLE error annotation guidelines (see section 7.3.4), cases such as the one in (34) are eliminated and precision rises noticeably. The filtered results are shown alongside the original results in Table 11.4. Selection precision improves slightly from 0.41 to 0.47, agreement precision rises noticeably from 0.40 to 0.54, and word order precision improves greatly from 0.10 to 0.75.

### 11.4.3 Intersection and MASI Scores

The intersection scores are shown for Test10 and Test90 in Table 11.5. The intersection scores, which are around 70% for all depths in the error typology show that Fledgling
usually correctly identifies a sentence as grammatical or includes at least one of the correct error types in its diagnosis. The MASI scores are shown in Table 11.6. The MASI scores, which penalize Fledgling for reporting errors that are not present, are closer to 0.60, which identify the same problem noticed in the precision scores in the previous section, namely that Fledgling currently overflags errors.
<table>
<thead>
<tr>
<th>Score by Error Typology Depth</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test10 MASI</td>
<td>0.675</td>
<td>0.630</td>
<td>0.615</td>
<td>0.608</td>
</tr>
<tr>
<td>Test90 MASI</td>
<td>0.652</td>
<td>0.613</td>
<td>0.594</td>
<td>0.583</td>
</tr>
</tbody>
</table>

Table 11.6: MASI Scores by Error Typology Depth

11.5 Summary

Fledgling correctly classifies 80% of sentences in the Test10 test set in terms of grammaticality for input with gold-standard non-word spelling corrections. Errors in grammaticality classification are primarily due to the coverage of the EAGLE grammar, which is still under development. In approximately 70% of the Test10 sentences, Fledgling’s error diagnosis includes at least one of the specific error types annotated in the sentence. It makes an accurate classification about the presence of a selection error for 82% of Test10 sentences, an agreement error for 86%, and a word order error for 91%. The evaluation shows that Fledgling overflags all error types, in particular word order errors, because it does not currently include a mechanism to rank and filter errors. If Fledgling’s reported errors are filtered by hand using the annotation principles from the EAGLE annotation scheme, the precision for identifying the presence of error types is 75% for word order errors and approximately 50% for selection and agreement errors. Recall is 43% for word order errors, 74% for agreement errors, and 88% for selection errors.
Chapter 12: Conclusion and Outlook

This thesis has presented EAGLE, an error-annotated corpus of beginning learner German, and Fledgling, a grammar checker for beginning learners of German that diagnoses errors using constraint relaxation. The error diagnosis algorithm developed for constraint relaxation with Extensible Dependency Grammar, QUICKPLAIN-XDG, is entirely language-independent. The only language-specific components are the grammar, which is naturally language-specific, and a mapping from XDG principles to error types used to diagnose errors.

12.1 EAGLE Corpus

The EAGLE Corpus (first presented in Boyd, 2010a) contains 9200 sentences with morphological and syntactic error annotation in two subcorpora, online workbook responses and exam essays. The annotation guidelines developed for EAGLE (Boyd, 2010b) use a linguistically-grounded error typology with a multi-layer, standoff annotation format. In order to make it possible to attain high interannotator agreement, the EAGLE annotation manual includes explicit, incremental steps that guide the annotator in the selection of error spans, target corrections, and error types. Specific guidelines describe how ambiguous errors and other difficult cases are to be annotated.
An interannotator agreement evaluation, which to the best of our knowledge is the first published study on interannotator agreement for an error annotation scheme, takes the sentence as the basic unit of annotation and considers the set of errors annotated on each sentence. The evaluation shows that annotators using the EAGLE annotation scheme show good agreement on grammaticality and the presence of selection, agreement, word order, and punctuation errors. Looking more closely at specific error types in each top-level category, the annotators show good agreement in the first and second levels of the agreement, word order, and punctuation error typologies. Selection and word form errors do not yet show acceptable agreement at any level of the annotation scheme, which points to the difficulties in annotating ambiguous errors and indicates that the EAGLE annotation guidelines should be revised and extended to better handle these cases.

12.1.1 Future Work

An error-annotated learner corpus offers a wealth of possibilities for further analysis and research. First, the EAGLE annotation should be reviewed and revised, using Fledgling where appropriate to verify annotators’ decisions. The evaluation showed that examining sentences where Fledgling and EAGLE disagree in terms of sentence-level grammaticality identifies errors in a small percentage of EAGLE sentences. The workbook subcorpus would also benefit from being annotated by a second annotator, preferably by a German native speaker, to improve the quality of the annotation. Additionally, an adjudicated version of the essay subcorpus should be created.

There are also several facets of the corpus that have yet to be explored. The workbook subcorpus keeps track of students’ progress through each activity and throughout the course.
The initial responses to each activity could be distinguished to examine the types of errors students make before receiving feedback from Quia and later responses could reveal students’ correction strategies. A profile of errors made throughout the course could make it possible to develop learner models that could be used in improving Fledgling’s performance or creating feedback modules.

To make annotation decisions in EAGLE reliable, the target hypotheses proceed bottom-up in a syntax-focused manner, ignoring semantics and the surrounding text or activity context as much as possible. Annotating context-dependent target hypotheses in addition to the existing syntax-dependent target hypotheses would make EAGLE more useful for research in foreign language teaching, for example in determining how to construct a feedback model for particular exercise types, where information about the activity context makes it easier to determine what the learner intended to write.

12.2 Fledgling

The QUICKXPLAIN algorithm (Junker, 2001) has been extended for use with Extensible Dependency Grammar CSPs and implemented using the constraint solver Gecode (Gecode Team, 2006). Fledgling shows high accuracy (82%–91%) in identifying selection, agreement, and word order errors, but low precision, in particular for word order errors. Further improvements to Fledgling relate to the grammar and the error diagnosis approach.

12.2.1 Grammar Development

A hand-written grammar is always a work in progress and the grammar developed for the EAGLE learner data is no exception. It needs to include coverage for more vocabulary, for
more subcategorization frames for verbs, nouns, and adjectives, more support for collocations, and especially for more complex constructions such as coordination. The EAGLE training data or native German text can be used to support future grammar development.

12.2.2 Error Detection and Diagnosis

The evaluation revealed a number of areas where Fledgling can be improved, many of which need to take into account its intended use with learner, activity, and feedback modules in an online workbook.

Missing and Extra Words

The handling of extra words is a particular problem for many parsers developed for ICALL systems (cf. Heift, 2003a; Reuer, 2005). Heift (2003a)’s E-Tutor uses restricted activities so that the parser does not need to handle missing or extra words. Each activity has a target answer with a specific number of words, and the system uses an extra module to check for missing and extra words that appears before the full-fledged parser in the processing pipeline. If there are missing or extra words, the input is not passed on to the parser until the missing and extra words are corrected by the learner. This greatly restricts the types of activities that can be added to an online workbook, but can also greatly improve the efficiency of the parser. Fledgling could be modified to work in both contexts—an activity setting could specify whether a preprocessing step has checked for the presence of the required words (e.g., the words provided in the activity); if so, QUICKXPLAIN-XDG does not need to check for missing or extra words.
**Ranking Partial Analyses**

QUICKXPLAIN-XDG currently outputs only the first parse found after constraints have been relaxed. Ideally, it should return all parses and use information about German, the activity, or the learner to find the best parse(s), which can be passed on to the feedback module. The parses can also be used in narrowing down QUICKXPLAIN-XDG’s constraint bundles to single errors that reflect the approach in the EAGLE annotation scheme. In the same way EAGLE annotators are instructed to prefer a minimal number of errors and errors that leave existing lexical items and phrases intact, Fledgling could be modified to prefer particular diagnoses.

Statistical extensions to XDG could be used to rank the parses and to improve the running time (see the following section). Dienes et al. (2003) presents an approach that uses an A* heuristic to guide the search for an optimal parse. Their probability model focuses on bilexical dependency probabilities for labeled relations. For future work in this area, a larger corpus of grammatically correct learner data or learner-targeted sentences such as material from textbooks would be needed to generate appropriate probability models. Dienes et al. (2003) report that they had problems with data sparsity training on dependency conversions of the Penn Treebank (Bies et al., 1995) for English and the NEGRA Treebank (Brants et al., 1997) for German, both of which are much larger than the EAGLE corpus.

**Running Time**

One major drawback of QUICKXPLAIN-XDG is the running time, on average over 30 seconds per sentence. One major factor is the size of XDG CSPs, which contain hundreds of
thousands of variables and constraints for even relatively short sentences. Many of these constraints are superfluous and could be eliminated in a preprocessing step. For example, a subject-verb agreement principle adds five low-level constraints for each pair of words in the sentence. Many pairs of words, such as a pair of determiners, are never going to stand in a subject relationship, so there is no need to include such constraints. The lexical entries or part-of-speech tags could be used to filter impossible or improbable relations, which could reduce the size of the CSPs. This would reduce the time needed to read in the CSP in Gecode and the number of key constraints that QUICKXPLAIN-XDG needs to examine. One such approach for HPSG uses specifications from a grammar to determine which nodes and constraints need to be examined by the parser (Götz and Meurers, 1995). An alternative, which focuses the dependency relations between words in an annotated dependency treebank rather than information from a hand-written grammar, is to use machine-learned classifiers such as the one developed in Bergsma and Cherry (2010) for filtering unlikely dependency arcs for statistical dependency parsing. For the Penn2Malt conversion of the Penn Treebank, they are able to prune 78% of potential dependency arcs while preserving 99.5% of the correct dependencies. A statistical parser could also be used to determine a set of probable dependency arcs, which could be passed to QUICKXPLAIN-XDG.

The XDG Development Toolkit, which is used for grammar development and to generate the XDG CSPs, is also limited by its implementation in Mozart/Oz. Mozart/Oz development has largely been abandoned, the lack of documentation for Mozart/Oz makes XDK difficult to modify and maintain, and Mozart/Oz is limited to use on 32-bit architectures, which makes it more cumbersome to install on many modern servers. Ideally, the XDK
metagrammar compiler and FlatZinc output would be rewritten in a faster, more common language, but this would be a non-trivial undertaking.

As part of a project on machine translation for under-resourced languages, Gasser (2011) has reimplemented parts of XDK with a small subset of principles in python. It is not advanced enough to support all of the constraints needed for German or the EAGLE grammar, but it shows a wider interest in maintaining and extending work on XDG.

12.3 Outlook for ICALL

After further development, it would be worthwhile to integrate the Fledgling grammar checker into an online workbook for learners of German. Fledgling’s analysis of learner’s responses can be used within the workbook to provide feedback to learners and also to extend the EAGLE corpus with automatically annotated responses. In a context with restricted activity types, i.e., activities that take into account the types of constructions supported well by Fledgling and the EAGLE grammar, Fledgling is already able to identify a wide range of errors. An extended grammar and a feedback module that ranks and prioritizes error types would make it possible to use Fledgling for a wider range of activities.


Dipper, Stefanie (2008). Tokenizer for German.


Grosz, Barbara, Karen Sparck Jones and Bonnie Lynn Webber (eds.) (1986). *Readings in Natural Language Processing*. Los Altos, California: Morgan Kaufmann.


262


