KU Leuven
Faculty of Arts
Blijde inkomstraat 21, Box 3301
3000 Leuven, Belgium

Pictograph-to-text translation with *Depicto 0.1*

Prototyping a rule-based translation pipeline for use in assistive writing systems

Adriaan Lemmens

Presented in fulfilment of the requirements for
the degree of Master of Arts in Linguistics

Supervisor: dr. Vincent Vandeghinste

Academic year: 2015-2016

Characters (approx.) :	161,688
Words (approx.) :	31,400

Acknowledgements

First and foremost, I would like to thank my supervisor, *Vincent Vandeghinste*, for giving me the freedom to determine the scope of my research while also being there when I needed help dialing it in. Needless to say, I am very grateful to him for sacrificing so many of his Friday afternoons these past few months. Our meetings, which, I swear, I really did go into with all the intentions of brevity, were a consistent source of encouragement and did much in the way of answering those questions which I could not articulate for myself. Also present at most of these meetings was *Leen Sevens*, whom I would like to thank from the bottom of my heart for her insight and, perhaps most of all, enthusiasm. In return, I have taken care to give as adequate a representation of the *Picto2Text* system as my own ignorance will permit. However, should I have misrepresented any part of the system, I apologise in advance. I would also like to thank the other partners at the Centre for Computational Linguistics: *Ineke Schuurman*, who, throwing an observation over her computer monitor from time to time, sometimes pushed the conversation in exciting new directions, and always made me feel welcome; and *Prof. Frank Van Eynde*, for helping me find my way around all the subtly different flavors of HPSG, and for his lectures on HPSG (attended last year), without which the work presented here would most likely have taken a very different, and most likely bumpy, course. On a similar note, I would also like to thank *Prof. Piet Mertens*, whose master's course on parsing (also attended last year) provided me with enough frame of reference to get to grips with the DELPH-IN formalism quickly and independently.

Next, I would like to express my gratitude to all contributors to the DELPH-IN project, with special thanks to *Woodley Packard* for developing (and open-sourcing) the ACE processor, as well as the maintainers of the DELPH-IN wiki, where, for the past months, I have practically lived. Thank you, maintainers, for creating such a sense of community.

Last but certainly not least, I would like to thank those dear to me, starting with my parents, to whom I am beyond grateful, not only for putting up with me and supporting my ambitions, but for providing me with the opportunity to spend so many months focusing solely on my research. Further, I would like to thank my brother and sister, *Reinert* and *Evelien*, as well as *Eva Vanermen*, *Sue Goossens*, *Xavier G.A.*, *Fred Giannetti*, and *Charlotte* 'discosis' *Tesselaar*, for their truly one-of-a-kind company and for reminding me from time to time not to take myself too seriously. Finally, I would like to thank my girlfriend, *Elisa Maes*. More than anyone else these past few months, she has kept me grounded when I wanted to float away, fascinating and frustrating me every day in a way that I cherish, but cannot explain, being *her* and no-one else, for which I thank her profoundly.

Contents

1 Introduction 5

1.1 Motivation & goals . . . 5

1.2 Related work: Approaches . . . 8

1.2.1 Semantic frames . . . 9

1.2.2 Shallow analysis . . . 12

1.3 The *Depicto* system . . . 14

1.4 Thesis overview . . . 17

2 Resources 19

2.1 The *Sclera* symbol set . . . 20

2.1.1 Bottom-up development . . . 20

2.1.2 Taking inventory . . . 22

2.1.3 Choosing *Sclera* . . . 24

2.2 Grammar engineering: The DELPH-IN toolkit . . . 26

2.2.1 Computationally implemented HPSG . . . 28

2.2.2 Minimal Recursion Semantics (MRS) . . . 37

2.2.3 The LinGO Grammar Matrix starter kit . . . 40

2.2.4 The ACE processor . . . 44

2.3 Transfer-based MT: the LOGON approach . . . 47

2.3.1 MRS rewriting . . . 49

2.3.2 The Transfer Matrix . . . 52

3 *Depicto* I: Deep parsing *Sclera* 54

3.1 Parser input: A note on picto identifiers . . . 55

3.2 A first crack at a grammar of *Sclera* 58
3.2.1 *Sclera* as a (natural) language 59
3.2.2 Setting up with the LinGO Matrix 64
3.2.3 'Bringing back' determiners 71
3.3 Understanding the MRS output 79
3.4 Extending the *Sclera* grammar 81
3.4.1 Temporal disambiguation 81
3.4.2 Triggering the interrogative mood 85
3.4.3 Semantically complex pictos 90

4 *Depicto* II: Transfer & generation 101
4.1 Transferring MRSs to the target language 102
4.1.1 A *Sclera*-Dutch transfer grammar 103
4.1.2 Output of the transfer stage 109
4.2 MRS-based target language generation 110
4.2.1 Setting up a toy grammar of Dutch 111
4.2.2 Exhaustive generation of well-formed surface strings . . 119

5 Winding down 125
5.1 Progress so far 125
5.2 Evaluation 126
5.2.1 Precision, coverage & performance 127
5.2.2 Comparing the *Depicto* and *Picto2Text* systems 132
5.3 Conclusions 137
5.4 Future work 140

A Demoing *Depicto* 144

B *Sclera*-grammar lexicon 148

C Example test sentences 149

Summary 156

Chapter 1

Introduction

1.1 Motivation & goals

There can be little doubt that, with its ever increasing reach, the digital (i.e., online) world has redefined what it means to be a participating member of society. At the same time, while it continues to make good on many of its promises of connectedness, social inclusion and equality, its expansion has yet to significantly benefit those members of society for whom its primary mode of information sharing, viz., written text, is inaccessible, resulting in a form of social exclusion. One group where much progress still needs to be made consists of individuals who, due to some *Intellectual Disability* (ID), are unable to comprehend or produce written (or spoken) natural language in a form that is sufficiently well-formed to be intelligible to other members of the language community (a language impairment which we will generally refer to as *aphasia* or *dysphasia*, depending on the severity of the condition) and, in many cases, are unable to use ICT independently altogether as a result. Prone to social isolation, as well its effects, including loneliness and depression, this group stands to benefit tremendously from improved access to remote communication facilities such as instant messaging, social media, and email, which could enable its members to communicate with (non-impaired) family members, friends, caregivers, even employers, as well as to seek out contact with those who perhaps understand them best, i.e., individuals in similar situations. Indeed, opening up online communication services to persons with ID not only represents a necessary step in the realization of a socially non-exclusive internet, but has the potential of significantly increasing the quality of life experienced by these individuals. To this end, there is a need for alternative interfaces that make the tasks of comprehending and producing text online more accessible. Here, we *focus on the task of text production.*

Previous work on *Alternative and Augmentative Communication* (AAC), a well-established field of research concerning the development of assistive tools and resources (both no/low-tech and high-tech) for a *wide* range of communication disabilities, shows *pictographs* (or 'pictures', 'symbols', 'pictograms', 'pictos', 'icons') to be particularly well-suited as an alternative to spoken and written communication (Steele et al. 1989). For individuals with a language impairment (i.e., as opposed to a motoric and/or speech impairment), the advantages of pictograph-based communication are (a) that symbols, in contrast to natural language words, provide a direct visual cue as to their meaning; (b) that they belong to a relatively small and easy-to-learn symbol set; and (c) that they tend to be structurally simple, expressing *essential* 'lexical' meaning only (i.e., without any grammatical adornment) and, by default, imposing few to no syntagmatic constraints when used to express a complex of meaning.

Pictograph-enabled writing tools allow users to 'write' (and verbalize) natural language text using only pictographs. Essentially, they are an application of *Machine Translation* (MT) and *Natural Language Generation* (NLG), providing an interface between pictographs and text. Over the past two decades, a number of approaches to mediating this interface have been proposed. With the exception of a number of recent newcomers, such as Sevens et al. (2015), none of the these approaches *specifically* target persons with ID (the needs associated with which are generally backgrounded by those of users with speech and/or motion impairments), but they are each unique in the way that they deal with the *challenges* posed by the pictographic input to regular machine translation methods, as the mixed selection of approaches discussed in Section 1.2 makes clear. Within the European Union alone, ID-sensitive assistive writing tools could benefit an estimated 2 to 5 million people (Keskinen et al. 2012).

The main *challenges for pictograph-to-text translation* stem from precisely those features that make pictographic communication so attractive in the first place. In the first place, in order to remain simple, pictographs tend to be *underspecified* both semantically and grammatically. That is, they individually encode for *less* information than do corresponding words in natural language. The precise degree of underspecification varies among different sets of pictographic symbols. In the second place, pictographs can in principle be used in any order, with the result that, in the worst case, the input to translation is unpredictable and potentially ambiguous. Further, since there are no 'corpora' of pictographic language use, traditional statistical translation methods are unsuitable.

In the context of this thesis, we have set out to design and implement a working prototype for an open-source machine translation pipeline that, when combined with a hypothetical user interface, enables *users with ID in particular* to compose grammatically precise (written) natural language utterances based on a sequence of freely selected pictographic symbols. Presented under its working name, the result is the *DePicTo* system: *De-Pictofication Tool*, which is re-written as *Depicto* for aesthetic purposes. The core of this system, the first version of which has been built for use by members of a Dutch-speaking language community, differs from that found in alternatives in that it is an implementation of a *rule-based* (or more precisely, *transfer-based*) approach to machine translation. The motivation for this approach, along with a more detailed description of the system, is given in Section 1.3. For now, the goals of the work presented in this thesis can be summarised as follows:

- Design and implement an *actual working* pictograph-to-text translation system that can deal with the challenges posed by the pictographic input (supra) – without restricting the method of user input.

- Get a sense of the costs of developing and extending the system (rule-based approaches to translation are notorious for its reliance on expensive language resources), and minimize these costs by adopting a modular design that ensures the re-usability of shared components. (We will elaborate on this in Section 1.3.)

1.2 Related work: Approaches

Since the early 1990s, a number of systems have been proposed that provide automatic mediation in the interface between AAC symbols on the one hand and natural languages on the other. Focusing on the subset that takes pictographic input to yield natural language output, this section briefly (and non-exhaustively) discusses three systems that have been designed to aid people with communication disabilities in the production of syntactically well-formed textual content. Although each largely unique in their approach to the challenges of translating from pictographs, they reflect a fundamental commitment to the needs of their users, including the reality of their socio-economic situation, with accessibility and affordability forming important design factors throughout. In that regard, they are the closest 'allies' of the *Depicto* system, although the approaches they take contrast with *Depicto*'s old-school, rule-based simplicity. For one thing, all three systems include at least one statistical component; for another, they treat their immediate pictographic input in very different ways. To make this second point clear, they are classified here as employing either *semantic frames* or *shallow* methods to analyze their input. These classes are not mutually exclusive, as the first system in the *semantic frame* class demonstrates, but they are sufficient to highlight the fundamental difference between the first and second systems' orientation to semantics and the third system's more token-oriented approach. Note, however, that many other classification systems would have been possible, depending on which parts of the individual approaches we wish to compare.

1.2.1 Semantic frames

1.2.1.1 Prothèse Vocale Intelligente (1998)

An older example, the Prothèse Vocale Intelligente (PVI) system for French (Vaillant 1998) stems from hands-on research involving people with cerebral palsy. This disorder causes neuromotor problems and cognitive deficits, which are manifested, linguistically, in difficulty with syntax – so-called 'agrammatism' – and a preference for short, telegraphic-style communication.

The PVI system's user interface is compatible with a number of input devices, suitable for various levels of motoric ability. Users can select pictographs from a grid using a computer mouse, a switch (for limited motoric ability), or a specialized keyboard. Once composed, input sequences are passed to an analysis module, where they are parsed semantically. This process, which ultimately yields a conceptual graph of the sequence's meaning, falls into three steps. First, the identifiers associated with the individual pictographs are retrieved, and predicative items, that is, those items with one or more selectional restrictions, e.g., verbs, are singled out. In the same step, the system moves on to the remaining pictographs and picks out the best candidates for each of the predicate's argument slots, choosing between different possible readings on the basis of a 'semantic harmonization' metric. Second, a lexical choice component which maps icons into French words prepares the resulting conceptual graph for the generation phase, adding uninstantiated morphosyntactic variables where appropriate. Third, and finally, the conceptual graph, now complete, is passed to a generator based on the Tree-Adjoining Grammar (TAG) formalism (Joshi & Schabes 1997). Using lexicalized subtrees consisting of anchors and slot-fillers, this step builds up syntactic trees corresponding to one more French sentences, which are given as output.

To its credit, the PVI system makes no assumptions about the internal order of pictographic sequences, leaving a great deal of expressive power up to the user. However, in order for this to work, the selectional restrictions used to disambiguate between all possible readings, are made rather strict. As a result, elliptical sequences, i.e., consisting of single pictographs or an incomplete predicative frame, are not accepted. On a more practical level, the PVI system does not seem to have been extended to other languages. This may have been due to the fact that the system relies on a visual language mapped specially to French words, requires a language-specific case grammar knowledge database for analysis, as well as a (again, language-specific) tree-adjoining grammar for generation. Nevertheless, for the *Depicto* system, in its current form 100% rule-based, the PVI system forms both a tale of caution and, with its commitment to free ordering on the input, a source of inspiration.

1.2.1.2 Sanyog (2009)

India is a large country that is home to a proportionally large number of individuals with speech and motion impairments. Stressing the need for computer-mediated AAC systems that, unlike other options available at the time, are tailored to Indian languages and, perhaps most important, are affordable, Bhattacharya & Basu (2009) introduce Sanyog, a picture-based communication aid that coverts pictographic input into grammatically correct sentences in Bengali or Hindi. Developing for these languages did not prove to be easy, and the developers' account of how they were eventually forced to revise their system, designing in the process a novel way of eliciting input at the user interface, is very instructive.

The original design of the Sanyog system made use of the 'compansion' approach. In this approach, pictographs selected through interaction with a grid-like interface undergo a preliminary phase of word order parsing. Words are grouped into sentence-sized chunks and part-of-speech information is gathered. The resulting augmented chunks are passed to the analysis module, where they are parsed semantically to form the basis of an instantiated semantic frame. Finally, in the generation step, this semantic frame is filled out through combination of syntactic and semantic constraints as well as the application of a series of preference rules, adding, among other, prepositions, determiners and morphological modification to the output.

However, the linguistic knowledge databases (e.g., WordNet and Framenet) and preference rules required for compansion to work do not exist in a sufficient form for Hindi and Bengali. Therefore, the developers adopt a new strategy for frame instantiation: ‘cogeneration’ (Copestake 1997), which can best be described as an extension of compansion in that it still makes essential use of semantic frames, but uses statistical information about the input, as well as n-gram word collocation information about the output, to assign frame roles to the most likely words.

There is one problem, though. While the cogeneration approach is effective once semantic parsing has completed, getting there unambiguously requires a reliable language model of the input, for which no available text corpus of Hindi or Bengali is entirely suitable. As an alternative solution, Bhattacharya & Basu (2009) develop the QR (Query-Response) model, a novel user-computer interface. Rather than giving the user full reign over the internal composition of pictographic sequences, the QR model expects users to select a verb first and then answer a series of (pictographically conveyed) queries relating to the roles associated with the semantic frame of the verb. The QR interface also includes a small set of optional markers with which users can specify different modalities (e.g., assertive, interrogative, as well as ‘wishing’ and ‘requesting’), negation, and tense and aspect.

The developers report that the system is already deployed at several institutions across India, and that the results of user testing are positive. The QR approach is experienced as intuitive and user friendly, and the quality of translation is satisfactory.

1.2.2 Shallow analysis

1.2.2.1 Picto2Text (2015)

The third and final system to be discussed in this section is Picto2Text, currently being being developed at the University of Leuven by Sevens et al. (2015). The system is part of a larger family of homegrown picto-based AAC technologies targeting social inclusion in the digital world. In fact, in terms of functionality, it is the complement of the previously developed Text2Picto system, which, as the name suggests, translates text into pictographic sequences (Vandeghinste et al. 2015). Like the latter, which was designed with the aim of facilitating online reading comprehension for people with ID, Picto2Text is designed in close co-operation with the WAI-NOT online communication platform[1] (for speakers of Dutch), where it will eventually be used, among other, as an assistive tool for email and instant message composition. Situated within the EU Able-to-Include framework, the project emphasises extensibility to as many European languages as possible. Currently, the system supports generation in Dutch, English, and – more recently – Spanish. Finally, of the three systems discussed in this section, Picto2Text relies the most on statistical models for its translation strategy.

In terms of both its user interface and its translation engine, the Picto2Text system is still under active development. In the version of the system described in Sevens et al. (2015), users can choose between two input methods. In the first, related pictographs are grouped into stacks based on topic similarity and frequency, and user navigates through these stacks to find the desired pictograph. In the second, a dynamic pictograph prediction tool allows users to quickly compose frequent utterances by selecting from a list of likely next pictograph candidates. At the moment, the choice between these two input methods is not exclusive, and users have complete freedom in the selection and ordering of pictographs. Sequences of selected pictographs are translated

[1] http://www.wai-not.be/

into the target language in real time, i.e., as they are composed. At the heart of the translation engine is a database of WordNet (Miller 1995) links and an n-gram language model. This works as follows. Pictograph identifiers are linked to language-specific WordNet synsets (the result of previous work on the Text2Picto system (Vandeghinste & Schuurman 2014)), such that when a pictograph is selected the synonyms from its associated synset are retrieved. These synonyms undergo reverse lemmatization. Each of the resulting surface forms represents a hypothesis for the language model. Since closed-class lexemes do not appear in the WordNet database, a series of additional rules ensure that pronouns (which do exist in the pictographic language used) are instantiated on these hypotheses. Similarly, nouns undergo a process of rule application that adds articles based on part-of-speech information. Next, hypotheses are passed to a language model where Viterbi-decoding based on a trigram model determines the most likely candidate permutation for output. In the Dutch version, this model has been trained on both written and spoken corpora.

In several ways the Picto2Text system is an interesting newcomer in the field of icon-aided communication. At the interface level, its dynamic pictograph prediction tool is the first of its kind and may, once refined, open the way to greater ease of composition as well as increased accuracy, without sacrificing freedom of expression (as in, e.g., the Sanyog system). The translation engine, further, requires a minimal set of resources (i.e., a WordNet database linked to a pictographic language, and a number of corpora), and has a result scaled nicely to include other European languages. At the same time, in its current form, the system does suffer from a number of important limitations. The first relates to accuracy of translation, which is currently too low for practical usage. The second is that the pictographic input is expected to resemble the structure of the target language (a consequence of the fact that no corpus of pictographic data exists with which to tune the language model). The third limitation is that the WordNet database with which the best scoring version of the system is currently linked (The Cornetto database of Dutch (Vossen et al. 2007)) has recently been made proprietary. This is arguably not a limitation of the system itself, especially since the its developers are already working on a possible migration to the open-source Open Dutch WordNet (Postma & Vossen 2014), but it may form a problem for sustained development in the future, which is why it is mentioned here. Nevertheless, as the online demo[2] shows, Picto2Text is a promising system, whose developers are already considering hybridization options involving analysing the input sequence semantically by means of rule-based parsing, and using semantic transfer for a more generation-heavy approach.

And this is where the *Depicto* system comes in.

1.3 The ***Depicto*** system

The *Depicto* system is a *proof-of-concept* implementation of a semantics-oriented, rule-based approach to pictograph-to-text translation that treats the source language utterance as a structured whole that can be analysed by 'deep' parsing methods (i.e., syntactically and semantically), rather than as a 'shallow' string of tokens. In the context of this thesis, the system aims (implicitly) to explore the feasibility of developing a rule-based pipeline that could, hypothetically, serve as a complement to the probabilistic core of the *Picto2Text* system (above) in future hybrids. As a standalone system, *Depicto* additionally aims to show that, while purely rule-based approaches to translation are out of vogue nowadays (Chan 2014), they provide a powerful means for overcoming both the sparsity of data of pictographic usage (above) and the semantically underspecified character of pictographic 'languages' in general, such that, by incorporating linguistic intuition (captured as rules), an approach of this kind is able to translate a pictographic source utterance in a way that is not only 100% consistent, but derives *as much meaning as possible* from the pictographic input. A working demo of the system can be obtained by following the instructions in Appendix A.

[2] http://picto.ccl.kuleuven.be/DemoP2T.html

The general architecture of the *Depicto* system, presented in Figure 1.1, consists of three consecutive *modules.* The first module, which is described in Chapter 3, employs a constraint-based grammar of the pictographic source language to perform a *deep analysis* on an input sequence of symbol identifiers. The output of this stage is a (potentially enriched) representation of the semantic structure of the source sequence. The second module, introduced in the first half of Chapter 4, uses a series of rewrite rules stored in a so-called *transfer grammar* to adapt this semantic representation so as to make it compatible with the target language. The result of this step is passed to the the third module, where, as described in the second half of Chapter 4, it is used to generate surface strings based on a grammar model of the target language that is written in the same formalism and framework as the grammar used by the first module. Currently, the output of the pipeline is an exhaustive, i.e., unfiltered, *list* of grammatically possible surface realizations. The analysis, transfer and generation modules are each operationalised by a different mode of the *Answer Constraint Engine* (ACE) processor. The visible input and output of the pipeline are strings. The intermediate data structures, visualised here as parallelograms, constitute semantic representations, which are formalized within the *Minimal Recursion Semantics* (MRS) framework. More information is provided in the next chapter.

The design of the *Depicto* system has been guided by two main criteria: *user-sensitivity* and *extensibility.*

The first, *user-sensitivity*, requires that the system stay true to the needs of its intended users. Like other assistive communication tools, the system's success will ultimately be measured by its usability, which, in this case, amounts to whether it can handle the communicative 'behaviour' of dysphasic users, e.g., unpredictability with regard to word order. As we see further on, the current version of the system has difficulty meeting this criterion completely. Nonetheless, it forms an important backdrop to the process of modelling the grammar used by the analysis module.

The criterion of *extensibility* requires that it be possible (if not easy) to

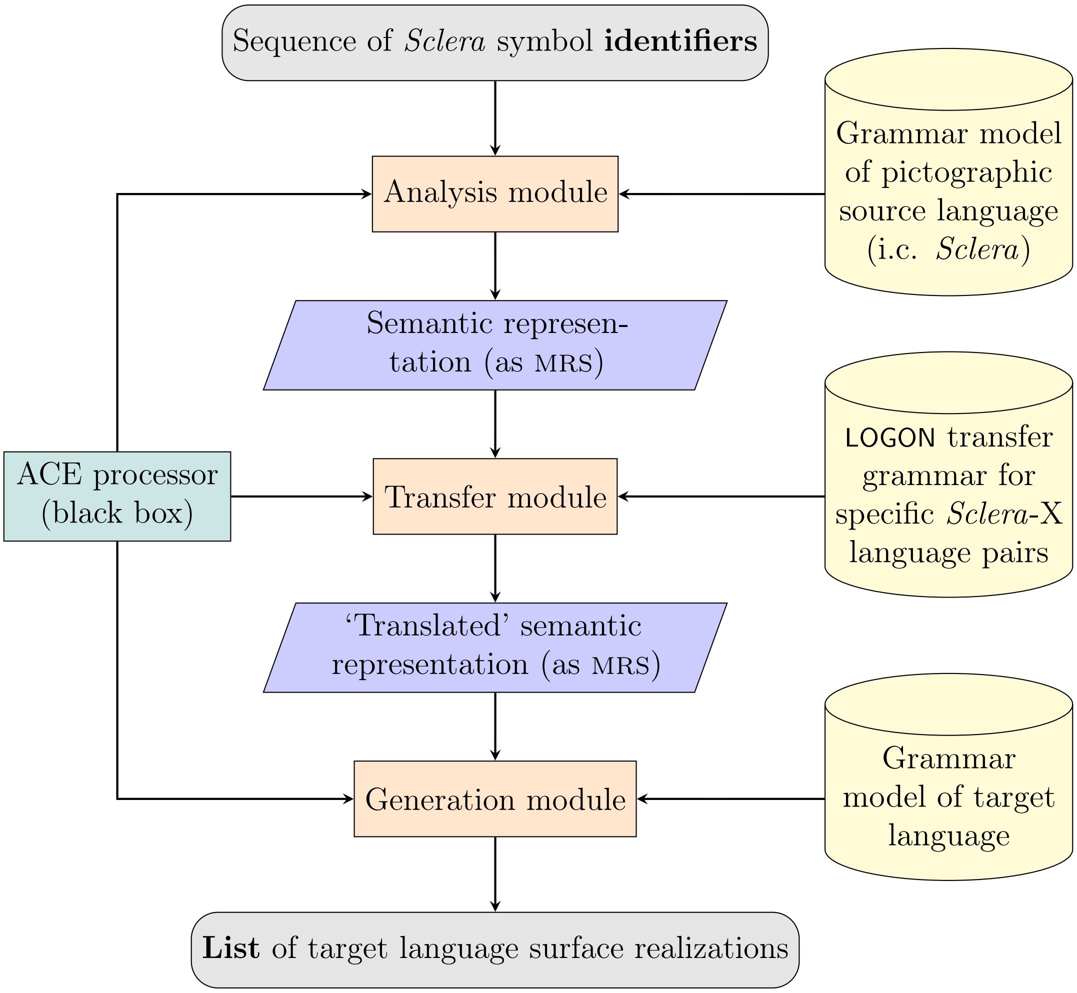

Sequence of *Sclera* symbol **identifiers**
Analysis module
Grammar model of pictographic source language (i.c. *Sclera*)
Semantic representation (as MRS)
ACE processor (black box)
Transfer module
LOGON transfer grammar for specific *Sclera*-X language pairs
'Translated' semantic representation (as MRS)
Generation module
Grammar model of target language
List of target language surface realizations

Figure 1.1: Architecture of the *Depicto* pipeline

extend the system to other target languages than Dutch. This forms the basis not only for *Depicto*'s modular design, which, in any case, is very similar to that used by other transfer-based systems (Chan 2014), but also for a second layer of modularity concerning the language resources required by the system's modules themselves, which we refer to as the *Principle of modularity*. According to this principle, the grammars of the (pictographic) source language and target language must be self-contained, i.e., independent of the other parts of the system. In other words, the successful operation of the system should not depend on any overlap between these two grammars. If adhered to, this principle makes it possible to incorporate, or 'plug in', other pre-existing target language grammars without having to make any changes to them. Meanwhile, the source language grammar is designed to be language independent (although semantic representations use an English-based metalanguage). Thus, in the ideal case, incorporating a new target grammar requires only the development of a new transfer grammar. This is easy enough to do, provided developers are somewhat familiar with the lexicon used by the target language grammar. Under the same criterion of extensibility, the design of the *Depicto* system uses only open-source resources and hopes to attract collaboration.

1.4 Thesis overview

Chapter 2 introduces and contextualizes all third-party *resources* used by the *Depicto* system. These include the pictographic symbol set, *Sclera*, toward which *Depicto*'s analysis module is geared; the formalisms (DELPH-IN-style *Typed Feature Structures*, *Minimal Recursion Semantics*), frameworks (*Head-driven Phrase Structure Grammar*), and grammar resources (the *LinGO Grammar Marix*) upon which *Depicto*'s source and target language grammars depend, as well as the parsing and generation tool responsible for processing these grammars (i.e., the *Answer Constraint Engine*); and, finally, the (LOGON-style) rewrite formalism used by *Depicto*'s transfer grammar.

Chapter 3 describes the development of *Depicto*'s analysis module. After a brief characterization of the input, the focus shifts to modelling the pictographic source language within a constraint-based framework. To start off, we show how *Sclera* can be analyzed as a natural language. Next, a simple grammar model is set up. We show how this model can be extended in a way *Sclera*-idiomatic way so as to increase its coverage as well as its precision with regard to the detection of illocutionary force and tense. Finally, we walk through an example of the output of the analysis module.

Chapter 4 describes the development of the other two modules in the pipeline.

In Section 4.1, we show how we set up a (very) basic transfer grammar for a *pictograph-to-Dutch* language pair. In Section 4.2, we design a toy grammar of Dutch that can be used as a target language in the *Depicto* system and show how this grammar is used by the third module to produce an exhaustive list of grammatical surface strings.

Finally, **Chapter 5** summarizes *progress* made on the *Depicto* system; presents a brief *evaluation* of the system, the second half of which involves a comparison with the *Picto2Text* system; draws a number of *conclusions* relating to the main goals and design criteria of the system; and, finally, looks ahead in terms of three areas of potential future work.

Chapter 2

Resources

This chapter provides background to the third-party resources upon which the *Depicto* system draws. First up is the extensive, open-source pictographic symbol set *Sclera*. Here, the discussion centers on a semantic inventory of the symbol set which later forms the basis for the *Depicto* system's approach to *Sclera* as a (simple) language whose grammar can be *modelled* formally. The theoretical framework within which this modelling process is carried out, viz., Head-driven Phrase Structure Grammar (HPSG, Pollard & Sag (1994)), is introduced next, along with the precise formalisms, frameworks, resources, and tools used in the development and operationalisation of the computationally implemented grammars that form the *Depicto* system's analysis and generation modules. Barring the HPSG framework itself, these resources have all been developed within the DELPH-IN community[1] (an informal international umbrella organisation for grammar engineering research that takes the HPSG framework as its starting point), and they are freely available on the DELPH-IN public repository. In a separate third section, the LOGON system for semantic transfer between monolingual DELPH-IN grammars is introduced. Similarly developed within DELPH-IN, this system, which allows grammars to 'communicate' with one another (albeit it in a one-directional

[1] http://www.delph-in.net/wiki/index.php/Home

way), forms the keystone to the *Depicto* pipeline and provides a mechanism that ensures the modularity of the system as a whole.

2.1 The *Sclera* symbol set

In the *Depicto* system, input sequences are composed using pictographic symbols taken from a subset of the extensive and freely available *Sclera* symbot set. In its entirety, *Sclera* comprises upwards of 13,000 pictographs that have been specifically designed to meet the communicative needs of people with ID. Visually, these symbols are characterized by a large degree of uniformity, with most adhering to a strict black-and-white color scheme. While several alternatives to *Sclera* exist (some of which are mentioned further down), none lend themselves quite so well to the desire for expressivity, as well as the open-source mentality, that has guided the development of the system presented in this thesis. Moreover, *Sclera* is the pictographic language of choice in a number of homegrown AAC technologies, including the Text2Picto system and the Picto2Text system (the latter already discussed in section 1.2.2.1). The remainder of this section briefly discusses the background and aims of the *Sclera* project, gives a brief overview of the kinds of pictographs found in the symbol set, discusses the Sclera2Cornetto resource for pictograph-WordNet linking, and lists a number of alternative options, with particular attention to the *Beta* language.

2.1.1 Bottom-up development

The vast size of the *Sclera* set as it stands today is by all accounts a testament to the success of one of the main aims of the *Sclera* NPO. As set out in its mission statement (Sclera NPO 2012), this is to provide various social target groups, particularly people with cognitive disabilities, with accessible and affordable (in this case: free) visual aids that are tailored to the needs of individual users. In addition to designing and distributing pictographs, the *Sclera* group also provides users and caretakers with free advice on how to use them. Through these interactions, as well as via the *Sclera* website [2], the developers have received an increasing number of requests for new and specific pictographs to which they have been happy to respond. Thus, since the release of the first registered batch of pictographs in 2007, the *Sclera* set has grown from 200 to just over 13,000 pictographs (Vandeghinste & Schuurman 2014).

However, size is not the only aspect that the *Sclera* group's bottom-up approach to pictograph development has influenced. In fact, the very function of the pictographs themselves has changed: while originally they were intended as a way of communicating instructions *to*, they now also address the communicative needs of the users. In terms of design aesthetics, the bottom-up approach was influential in the adoption of the black-and-white color scheme. Its uniformity was found to lend the images a sense of 'calmness' and proved to be very popular among people with autism, and the higher contrast appealed to people with partial visual impairments. Finally, user feedback suggested that certain groups vary in their preference for a particular depiction of a concept, finding one pictograph more obvious than others. In the *Sclera* set, therefore, concepts can stand in a one-to-many relationship with pictographs, as Fig. 2.1 shows for the concept 'dog'.

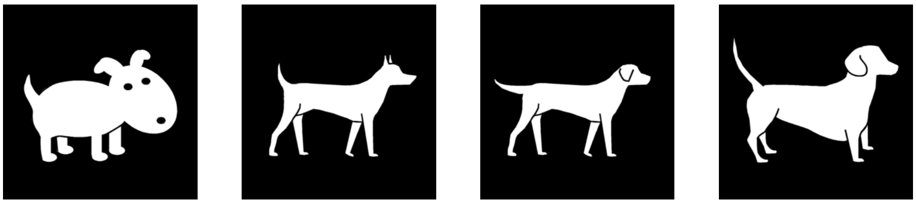


Figure 2.1: Different *Sclera* depictions of the concept 'dog'

[2] http://www.sclera.be/en/vzw/home

2.1.2 Taking inventory

The *Sclera* set is primarily composed of pictographs depicting entities or processes. In terms of function, these correspond to nouns and verbs. Within the ‘nominal’ category, pictographs generally do not distinguish between singular and plural readings, nor do they encode quantification or reference, since quantifiers and determiners do not exist in the *Sclera* language. (The exception concerns a small set of pictographs that convey ‘pronominal’ meaning, including pictographic equivalents of personal, demonstrative, possessive, and interrogative pronouns.) As a result of the customization-ready approach discussed above, there is room for hyponymy, with specific realizations of a particular concept receiving their own pictographic depiction, as Fig. 2.2 shows for a sample of different realizations of the hypernym ‘bicycle’. Next, within the verbal category, pictographs are similarly underspecified for number, as well as tense and, unsurprisingly, inflection. Auxiliary verbs are absent, although there does exist a pictographic representation for the copular verb ‘be’. A small set of adjectives is also included. Most of these relate to emotion.

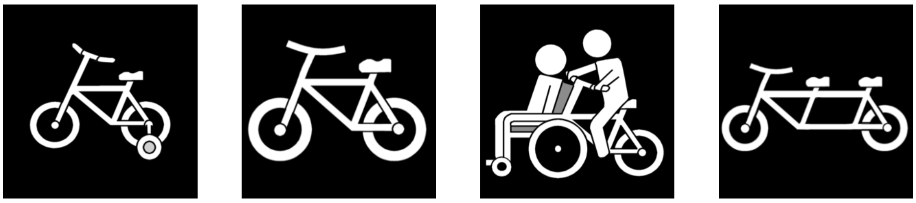
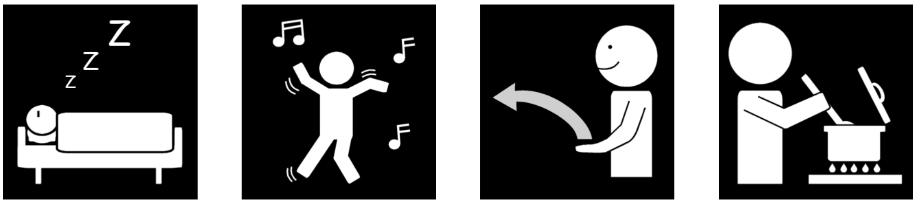


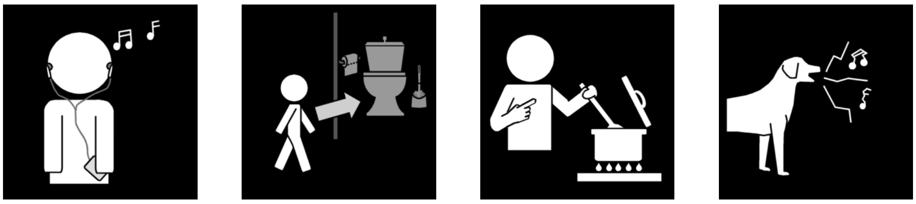
Figure 2.2: Different kinds of bikes

In addition to pictographs that depict simple concepts, most of these corresponding to a single word in some natural language (depending on the language, of course), a sizeable portion of the *Sclera* inventory consists of pictographs that represent conceptual complexes. For pictographs depicting entities this amounts to a form of nominal compounding. Within the verbal category, it is not uncommon for a verbal head and one or more of its complements to be rolled into a single pictograph. These complex pictographs provide a mechanism for expressing relations which, in English and Dutch, are generally conveyed by prepositions (another category missing from the content word-oriented symbol set). Adverbial relations, which also do not appear in the system set, can similarly be expressed.



(a) 'sleep' (b) 'dance' (c) 'give' (d) 'cook'

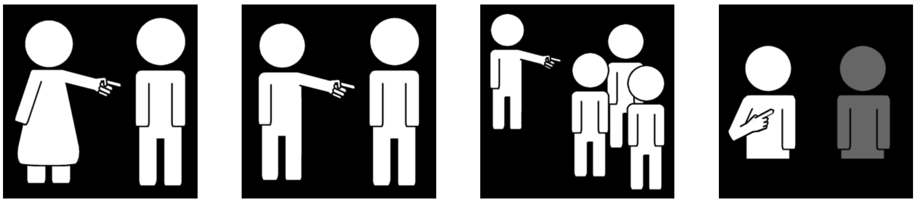
Figure 2.3: Simplex situations ('verbs' pictos)



(a) 'listen' + 'music' (b) 'go (to)' + 'bathroom' (c) 'I' + 'cook' (d) 'dog' + 'bark'

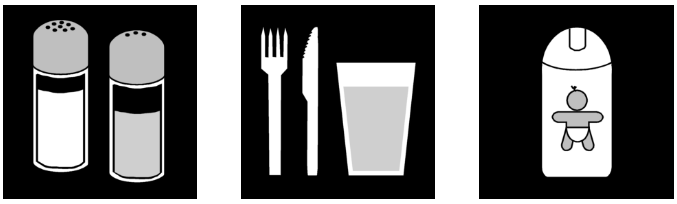
Figure 2.4: Complex situations

Finally, the *Sclera* set includes a number of pictographs which are slightly more marked. A first subset comprises opposition pairs. These often relate to behaviour. In contrast to the rest of symbol set (with a few exceptions), these pictographs make use of color: their (black) background is swapped out for either a green or red one to indicate permission or prohibition, respectively. Other pictographs, most of them 'verbal', are marked for 'positive' or 'negative' connotation on top of the concept that they depict, allowing the user to express his attitude toward, for example, a situation of 'saying goodbye'. On a thematic level, the *Sclera* set contains a wide variety of pictographs



(a) ‘he/him’ (b) ‘you (singular)’ (c) ‘you (plural)’ (d) ‘mine’

Figure 2.5: ‘Pronominal’ pictographs



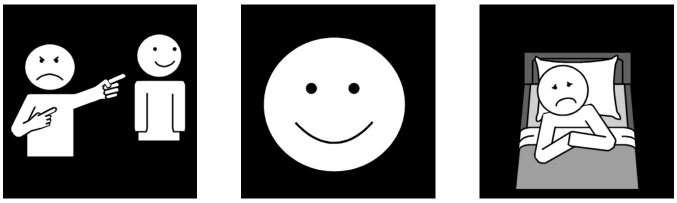
(a) ‘salt and pepper’ (b) ‘food and drink’ (c) ‘baby soap’

Figure 2.6: Complex entities (nominal compounds)

relating to independent living (bank cards, ATMs, taking the train, etc.), culture, contemporary lifestyle and technology (electronic communication, drugs, music), as well a number of ‘taboo’ subjects such as sexuality, religion, hygiene, and death.

2.1.3 Choosing *Sclera*

Sclera is already used by the Picto2Text system (Sevens et al. 2015), as well as by its forerunner, the Text2Picto system (Vandeghinste et al. 2015), both introduced in section 1.2.2.1. As is discussed there, both translation systems rely on previously established links between pictograph identifiers on the one hand and WordNet synsets on the other (Vandeghinste & Schuurman 2014). (These synsets are in turn linked to the EuroWordNet grid and SUMO ontology.) In order to keep open the option of integration with the Picto2Text system further down the road, as well as to enjoy the possible future benefits of the rich lexical-semantic resource upon which its operation relies, the

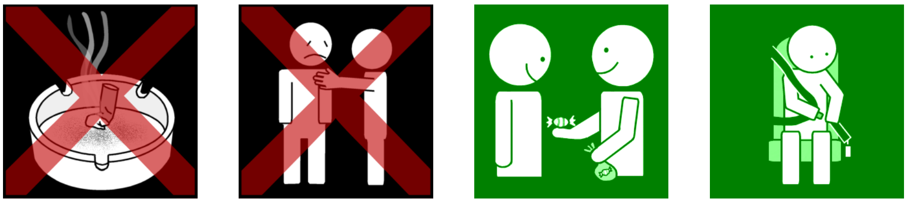


(a) ‘jealous’

(b) ‘happy’

(c) ‘bed uncomfortable’

Figure 2.7: Emotion/attitude



(a) ‘no smoking’ (red cross)

(b) ‘no touching others’ (red cross)

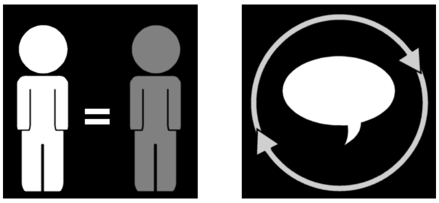
(c) ‘share candy’ (green)

(d) ‘buckle seatbelt’ (green)

Figure 2.8: ‘Imperative’ pictographs (directives)

Depicto system follows the Picto2Text example and adopts the *Sclera* set.

There are alternatives, however. Filtering these on availability (the operative criterion being ‘free’), expressiveness, and success in user testing, the next best option is the *Beta* symbol set. The pictographs in this set make much more use of color than those in *Sclera*, which, despite the resulting lack of uniformity, makes them visually more appealing for some people. Aesthetics aside, there are some important differences with the *Sclera* set. For one thing, *Beta* symbols depict only simplex concepts, as opposed to allowing compounding and complexing. For another, prepositional relations are depicted. While the first divergence makes little difference for compatibility with a *Sclera*-oriented system, as *Beta* can simply be treated as a subset of the expressive scope of *Sclera*, the second difference makes things a little trickier, since prepositional pictographs are not supported by the grammar of *Sclera* which the *Depicto* system uses. For the time being, therefore, *Beta* is left to the side, although it is important to realize that it is there, if only because it *is* supported by the Picto2Text and Text2Picto systems.



(a) 'equal opportunities'

(b) 'repeat'

Figure 2.9: Abstract concepts

2.2 Grammar engineering: The DELPH-IN toolkit

There are numerous grammar formalisms for which parsing and (slightly less often) generation implementations exist. These include – to name a few – Context-Free Grammars (CFGs; Chomsky (1959), Unification Grammars (UGs; i.a. Shieber et al. (1983)), (Lexicalized) Tree-Adjoining Grammars ((L)TAGS; Joshi & Schabes (1997)), and Dependency Grammars (DGs; Vater (1975)). With the exception perhaps of traditional CFGs, each of these formalisms has its merits. Additionally, their implementations vary in the algorithms that they use, the degree to which these algorithms are optimized, and the extent to which they have been designed with a particular theoretical framework for syntactic analysis in mind. In short, choosing a formalism is no easy task.

To keep things simple, therefore, both the analysis and generation modules found in the *Depicto* system make use of the same unification-based formalism, are written within the same framework, viz., HPSG (Pollard & Sag 1994), and rely on the same software for processing. These resources, each representing multiple person-years of work, are all developed and distributed by the same community: the *Deep Linguistic Processing with HPSG Initiative* (DELPH-IN[3]), an international collaborative grammar engineering effort aimed at developing open-source NLP tools for 'deep' linguistic processing of human language. These tools can be augmented with statistical processing methods,

[3] http://moin.delph-in.net/

but the basis is thoroughly symbolic. Through collaboration, with founding members coming from Stanford's CSLI (USA), from Germany's DKFI and University of Saarland, and from Norway's University of Science & Technology and University of Oslo, the DELPH-IN project has made it significantly more cost-effective to develop efficient large-scale implemented grammars. It has also made the development process a great deal more rewarding by providing a variety of specialized tools to process grammars with, the most important relating to parsing, generation and, as we shall see in the next section, the transfer of semantic representations for translation and paraphrasing purposes. Some of these tools are intended for production, and DELPH-IN grammars and tools are already in place in a number of commercial applications.

The intercompatibility of DELPH-IN tools and DELPH-IN grammars, even though these are often developed independently of another, is due to the convergence on a small set of common descriptive formalisms. The first such reference formalism relates to the specific variant of typed feature structure logic which all DELPH-IN grammars are expected to follow. (Typed feature structures form the primary data structure in HPSG as well as the LFG and CG frameworks, both of which, incidentally, are also compatible with the DELPH-IN grammar processors.) A second reference formalism is the Minimal Recursion Semantics format for semantic representation, which generally forms the main output of a DELPH-IN parser, and from which DELPH-IN grammars are able to generate. Both these formalisms, as well as the declarative language used to implement them, will be discussed in more detail in the course of the current section.

The following provides a brief introduction to the HPSG framework, discusses the DELPH-IN-style typed feature structure (TFS) formalism in more detail, and takes a look at the declarative specification language in which grammar files are written ($\mathcal{TDL}$) as well as the way in which individual files combine to form a complete HPSG grammar. After introducing the MRS formalism for semantic representation next, we turn to the LinGO Grammar Matrix starter-kit, which provides the 'core' for the grammars used by the *Depicto* system's analysis and generation modules. Finally, the relatively recent ACE system is introduced, an all-in-one DELPH-IN grammar processor designed for speed and efficiency, with support for transfer-based machine translation.

2.2.1 Computationally implemented HPSG

This subsection presents a lightning introduction to the framework of Head-driven Phrase Structure Grammar (HPSG) (Pollard & Sag 1994) and describes the specific typed feature structure formalism used in DELPH-IN implementations. To round off, it looks at the expected source file structure of a typical DELPH-IN and details a selection of common differences between theoretical and computational variants of HPSG.

2.2.1.1 The basic theoretical framework

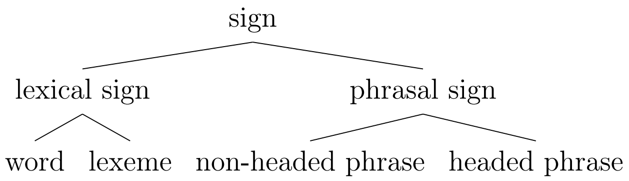
Head-driven Phrase Structure Grammar (HPSG) is a strongly lexicalist, modular, unification-based linguistic theory originally developed by Ivan Sag and Carl Pollard in the mid-1980s (Pollard & Sag 1994). Unlike transformational grammars, it is entirely surface-oriented, and treats linguistic objects not as trees, but as structured sets of constraints that can be modelled by means of *feature structures* (FSs) (informally: feature-value pairs; formally, directed acyclic graphs). Within this model-theoretic approach, each linguistic object, and thus each feature structure, is further seen as instantiating a particular *type.* The type to which a linguistic object belongs is situated within a greater hierarchy of types. These are ordered from most general to most specific according to the generality of the information, i.e., constraints, specified by the associated feature structure. This *type hierarchy* is used to capture both lexical and syntactic generalizations. The constraining relation between types and their features is expressed by *typed feature structure descriptions*[4], which

[4] The literature tends to emphasize that there is a difference between these typed feature structure descriptions and the typed feature structures (TFSs) which they describe (Copestake 2002). However, the difference is very subtle, essentially boiling down to the

are represented as attribute-value matrices (AVMs) (Sag et al. 1999).

To re-iterate the last paragraph slightly – within the HPSG framework all linguistic objects are modelled as TFSs. On the linguistic level, the most general type is that of the *sign* (a tuple of phonological and syntactic-semantic information) (Fokkens 2014), which has lexical and phrasal signs as its subtypes. Lexical signs include lexemes and/or words. Phrasal signs amount to the grammar rules that combine lexical or other saturated phrasal TFSs (via the operation of unification) into larger well-formed TFSs corresponding to immediate constituents. The principles that determine both the syntactic and semantic well-formedness of these constituents form an additional set of phrasal TFSs. As larger TFSs are built up (from a parsing perspective), the phonological/orthographic information borne by constituent TFS is appended, such that the root TFS, which is equivalent to the root of a syntactic tree, contains the entire parsed string.

(2.1) Top of a typical HPSG type hierarchy



True to its name, the HPSG framework generally treats syntactic constituents as consisting of a head structure and one or more sister structures which are selected for by the head or, in some cases, select the head, but without assuming its status. In terms of the valential relationship between the head and non-head daughters of the mother node, the head-subject, head-complement, head-specifier, and head-modifier rules are the most common, and most phrasal rule types inherit from them. (Their names are sufficiently

fact that TFS descriptions place partial constraints on possible models, whereas typed feature structures are understood as maximally specific models that correspond to linguistic reality. Because of this subtlety, particularly in an implementation-oriented context, where, prior to compilation, grammars consist entirely of TFS *descriptions*, I follow Copestake (2002)) in ignoring it except when it is absolutely relevant.

self-explanatory for this purposefully brief introduction.) In order that the head of a constituent is accessible to subsequent phrasal rules, its HEAD feature, which encodes syntactical features such as its part of speech, is passed up by virtue of the Head Feature Principle (Sag et al. 2003), which. The TFS through which this principle enters the grammar makes use of an identity statement (notationally indicated with co-indexed variable tags) that requires the value of the HEAD feature of the mother TFS and the value of the HEAD feature of the head daughter TFS to be token-identical. This mechanism of structure sharing is characteristic of HPSG and, combined with unification, accounts for much of its flexibility and elegance.

As a lexical theory, finally, HPSG treats the lexicon as a rich and structured object (Müller et al. 2013). In order to avoid redundancy, the type hierarchy includes a separate set of mostly unary lexical rules whose ‘application’ is constrained to TFSs of the type *lexeme* and/or *word* (depending on the precise flavor of HPSG one is working in). These can be subdivided into inflecting/morphological/affixing rules and non-inflecting/constant rules. While intuitively evident in theoretical versions of HPSG, the former, when implemented, requires additional machinery which usually amounts to some form of regex substitution. Common examples of usage cases for non-inflecting lexical rules include modification of the valential requirements of a matrix verb in the passive voice and of a ditransitive verb that can undergo ‘dative shift’. Unlike the rules discussed above, lexical rules are less free as regards the order of their application, particularly, as we shall see, when implemented, with inflecting rules always coming before their inflecting counterparts (again, from the perspective of parsing, that is).

2.2.1.2 The DELPH-IN formalism of TFSs

DELPH-IN grammars consist of a specification of a type system and of various typed feature structures (TFSs) which are to be inferred from this type system. These TFSs, which function (among other) as grammar rules, lexical rules, and lexical entries (Copestake 2000), are considered well-formed on the basis of a relatively conservative subset of the typed feature structure logic described by Carpenter (1992). This logic also forms the basis of the machine-readable specification language in which DELPH-IN grammars are encoded: type-description language ($\mathcal{TDL}$) (Krieger & Schäfer 1994). We will return to $\mathcal{TDL}$ shortly.

The selection of the DELPH-IN formalism was guided by concerns about linguistic adequacy grounded in HPSG and about requirements for efficient processing (Oepen 2010). Informally, the formalism is characterized as relating to a closed-world, multiple-inheritance type system that enforces strong typing (i.e., every structure belongs to a type) and strict appropriateness (i.e., all features a type can be defined for must be introduced at some point in the inheritance hierarchy). At the same time, the formalism allows types to be associated with arbitrarily complex constraints (such as TFS embedding via structure sharing), which are inherited either upon compilation or at runtime. The inheritance of constraints, moreover, is monotonic, and features are introduced maximally, such that in instances of multiple inheritance (which, to be clear, form the majority) from several partial TFS descriptions, the result is always a unique most general structure, i.e., a greatest lower bound. Compared to other variants of typed feature structure logic, the DELPH-IN formalism is very restricted. There is no room for formal devices such as disjunction (which can lead to expensive backtracking), negation, implication, inequality, default inheritance (this is actually permitted, but rarely used), set-valued features and relational constraints (e.g. 'true' lists, which are implemented instead using the mechanism of embedded feature-structures), and extensionality (anything that is not in the grammar does not belong to the models which it can produce) (Copestake 2000). As a matter of fact, the only formal device/operator which the DELPH-IN TFS formalism uses is conjunction.

Type Description Language ($\mathcal{TDL}$) Set out in Copestake (2002) (to date, still the best introduction to DELPH-IN-style grammar development),

$\mathcal{TDL}$ is the specification language used to define types, their constraints, and their place in the type hierarchy, i.e., in relation to their supertypes. $\mathcal{TDL}$ descriptions always have an AVM equivalent, as (2.2) shows for a non-linguistic example. Feature structures are demarcated by opening and closing square brackets. Feature names are conventionally written in uppercase, while their values, which can be other types, are written in lowercase. If the value of a feature is both a type *and* and a more specific feature structure appropriate to that type, then the two are separated by an ampersand (‘&’) to indicate conjunction/unification. Feature-value pairs are separated by commas, and all $\mathcal{TDL}$ descriptions must terminate in a period (‘.’). Instead of using boxed numbers, $\mathcal{TDL}$ descriptions indicate structure sharing by means variable names prefixed with a hashtag (or ‘octothorp’, ‘pound sign’, ‘#’).

(2.2) Equivalent AVM and $\mathcal{TDL}$ descriptions for a (**non-linguistic**) typed feature structure

a. AVM description

$$\begin{bmatrix} \textit{dog} & \\ \textsc{breed} & \text{‘Dalmation’} \\ \textsc{coat} & \begin{bmatrix} \textsc{background} & \boxed{1}\ \textit{white} \\ \textsc{foreground} & \begin{bmatrix} \textit{spotted} & \\ \text{SPOT-COLOR} & \textit{black} \end{bmatrix} \end{bmatrix} \\ \textsc{fear} & \text{‘Cruela de Vil’} \\ \textsc{litter-size} & \textit{more-than-100} \\ \textsc{health-problems} & \left\langle \textit{deafness} \right\rangle \\ \textsc{other} & \begin{bmatrix} \textsc{favorite-camouflage-environment} & \begin{bmatrix} \textit{snow} & \\ \textsc{color} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

b. $\mathcal{TDL}$ description

```
dalmation := dog &
    [ BREED "dalmation",
      COAT [ BACKGROUND #camo-color & white,
             FOREGROUND spotted & [ SPOT-COLOR black ] ],
      FEAR  "Cruela de Vil",
      LITTER-SIZE more-than-100,
      HEALTH-PROBLEMS < deafness >,
      OTHER.FAVORITE-CAMOUFLAGE-ENVIRONMENT snow & [ COLOR #camo-color ] ].
```

2.2.1.3 Components of a DELPH-IN grammar

At the core of any DELPH-IN grammar is its type hierarchy. This hierarchy can be distributed over more than one $\mathcal{TDL}$ file (e.g., for purposes of design, debugging, clarity), but is interpreted by the grammar processor as building up one large ontology of TFS descriptions.

The remaining components of the grammar are referred to as *entries*. When compiled or invoked by the processor, these give rise to maximally specific typed feature structures. They comprise both rules and lexical entries. Once again, these can be distributed over several $\mathcal{TDL}$ files, but the processor will interpret them accordingly. Phrasal rules are expanded upon compilation, as are lexical rules, although these are treated separated so as to account for the unique order-specific properties of the lexical rules, as well as the intended behaviour of the phrasal rules, which, though specified as TFSs, are interpreted as rewrite rules over other TFSs. Entries in the lexicon, finally, are expanded at runtime rather than during compilation so as to keep down the size of the compiled grammar. The configuration file associated with the processor tells it how to interpret which components.

An small class of additional grammar entries is contributed by two files that conventionally bear the names `roots.tdl` and `labels.tdl`. `roots.tdl` contributes those TFSs that can function as start symbols in the grammar, that is, as the root of a parse tree. `labels.tdl` contributes the labels used to name the nodes in parse trees.

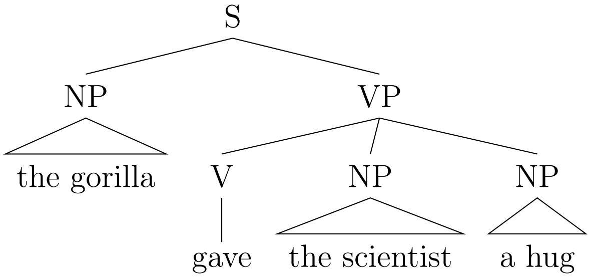
2.2.1.4 Deviations from theoretical HPSG

While canonical theoretical HPSG (which, for the sake of the discussion here, I assume to be the flavor introduced in Pollard & Sag (1994)) and implemented HPSG have, as one would expect, much in common, there are also some differences in the analyses which they produce. These differences are relatively minor, but it helps to be aware of them nonetheless. The most salient, therefore, are briefly sketched out here.

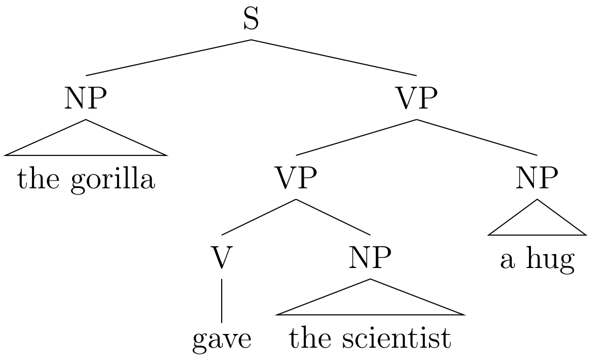
Preference for unary or binary branching rules While the general phrasal rules, or 'immediate dominance schemata' (Pollard & Sag 1994), of theoretical HPSG generally license *N-ary* (viz., 1,2,3,4...n-ary) branching phrase structures example (2.3a), such rules are generally constrained to being either binary or unary in implemented HPSG. This is not to say that ternary, quaternary, etc. branching is ruled out; it is simply dispreferred, as it leads to a significant increase in the size of the search space that the parsing and generation algorithms need to consider. The traditional analysis of the ditransitive valence schema of a verb such as *give* is instead analysed as two successive 'applications' of a binary head-complement rule, as in example (2.3b).

(2.3) Ternary vs. binary branching analyses of ditransitive verb phrase (VP)

a. Ternary branching VP analysis



b. Binary branching VP analysis



Lists as feature structures In theoretical HPSG, lists are expressed as comma-separated items between large angle brackets. While $\mathcal{TDL}$ contains syntactic sugar that mimics this notation, the underlying implementation of the list, however, is that of a recursive typed feature structure consisting of a *head* (FIRST) and a *tail* (LAST), which can have as its value a terminating *nil*, any single type, or another list with the same constraints.

(2.4) a. Traditional AVM notation used for lists:

$$\begin{bmatrix} \textsc{fruits} & \left\langle \text{‘oranges’, ‘bananas’, ‘apples’} \right\rangle \end{bmatrix}$$

b. Underlying structure of list, as made explicit in implementations

$$\begin{bmatrix} \textsc{fruits} & \begin{bmatrix} \textsc{first} & \text{‘oranges’} \\ \textsc{rest} & \begin{bmatrix} \textsc{first} & \text{‘bananas’} \\ \textsc{rest} & \begin{bmatrix} \textsc{first} & \text{‘apples’} \\ \textsc{rest} & \textit{*end*} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

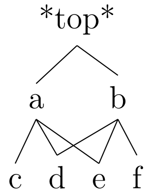
Logics sets, which theoretical HPSG indicates using a special curly brackets notation, are treated within the DELPH-IN formalism as simple lists. That is, the items of the set are treated as having a particular order.

Finally, lists which are appended during the construction of larger TFSs (e.g., lists of semantic predications, the value of the feature bearing orthographic/phonological information about the parsed/generated string, etc.) are implemented as special kind of list, namely, a *difference list.* We do not need to get into the details of how this separate type of list is used within the grammar, but it is useful to know that $\mathcal{TDL}$ includes special syntactic sugar that allows lists of this kind to be written as `<!  [ LIST-ITEM ] !>`.

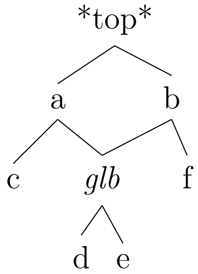
No Lexeme-word distinction Some flavors of HPSG posit a distinction between lexemes on the one hand and the words which these lexemes instantiate on the other. As a result, one or more constant lexical rules are required to *pump* non-inflecting lexemes to the word level. This distinction is perfectly compatible with the DELPH-IN framework, yet it is rarely adopted. In DELPH-IN-grammars lexemes and words tend to be synonymous. This is certainly the case for the grammars developed in this thesis.

***Greatest lower bound* requirement** Multiple type inheritance is as powerful a mechanism in theoretical HPSG as it is in implemented HPSG. However, while theoretical variants, generally for illustrative purposes, show multiple same-level types inheriting from the *same* supertypes, this is not allowed in the more strictly formalized type system found in implemented HPSG: in cases of multiple inheritance, types must share a unique most general type, or *greatest lower bound (glb).* In large-scale implemented grammars, this generally leads grammar writers to specify intermediate types that, in a purely theoretical framework, may be seen as redundant. While this well-formedness condition on the type hierarchy is clearly not trivial, failing to take it into account does not immediately undermine the grammar either, as most grammar processors possess additional machinery to add missing *glbs* to the type hierarchy.

(2.5) a. Hypothetical example of multiple inheritance in *theoretical* HPSG



b. Multiple inheritance in DELPH-IN, with additional *greatest lower bound* (*glb*) type



2.2.2 Minimal Recursion Semantics (MRS)

DELPH-IN grammars are designed to map bi-directionally between surface strings and semantic representations. These representations are encoded as TFS and correspond to descriptions used within the formalism of Minimal Recursion Semantics (MRS) (Copestake et al. 2005). This formalism, which aims to provide a framework for computational semantics that is suitable for both parsing and generation, serves as theory-agnostic meta-language for more traditional semantic formalisms such as predicate calculus and generalized quantifiers (which are common in DELPH-IN grammars). MRS abstracts away from surface syntactic structure and undecidable scope ambiguities by means of characteristically flat (i.e., non-recursive) representations that use explicit pointers to encode argument structure and scope effects (Copestake & Flickinger 2000). The resulting potential for underspecification recommends MRS to computational applications that, like the one developed here, require robust generation. Despite its close ties to the DELPH-IN project, MRS has also become popular among theoretical HPSG researchers as well as among researchers working within non-HPSG frameworks such as TAG, CxG, and LFG. In the remainder of this subsection, we take a closer look at the structure of the representations themselves.

2.2.2.1 Make-up of an MRS structure

An MRS structure (sometimes abbreviated, somewhat confusingly, to 'MRS') essentially consists of three parts: an index/indices, a bag of elementary predications, and a set of handle constraints. Each is explained briefly here with reference to the example MRS representation for the sentence *Every brown dog barks* shown in (2.6).

(2.6) MRS representation of *Every dog barks*

$$\begin{bmatrix} \textit{mrs} & \\ \textsc{hook} & \begin{bmatrix} \textit{hook} & \\ \textsc{ltop} & \boxed{h0} \\ \textsc{index} & \boxed{e2} \end{bmatrix} \\ \textsc{rels} & \left\langle \begin{matrix} \begin{bmatrix} \textit{ep} & \\ \textsc{lbl}\ \boxed{h4} & \\ \textsc{pred} & \text{`every\_q\_rel'} \\ \textsc{arg0} & \boxed{x3} \\ \textsc{rstr} & \boxed{h5} \\ \textsc{body} & \text{H6} \end{bmatrix}, \begin{bmatrix} \textit{ep} & \\ \textsc{lbl} & \boxed{h7} \\ \textsc{pred} & \text{`dog\_n\_rel'} \\ \textsc{arg0} & \boxed{x3} \end{bmatrix}, \\ \begin{bmatrix} \textit{ep} & \\ \textsc{lbl} & \boxed{h1} \\ \textsc{pred} & \text{`bark\_v\_rel'} \\ \textsc{arg0} & \boxed{e2} \\ \textsc{arg1} & \boxed{x3} \end{bmatrix} \end{matrix} \right\rangle \\ \textsc{hcons} & \left\langle \begin{bmatrix} \textit{qeq} & \\ \textsc{harg} & \boxed{h0} \\ \textsc{larg} & \boxed{h1} \end{bmatrix}, \begin{bmatrix} \textit{qeq} & \\ \textsc{harg} & \boxed{h5} \\ \textsc{larg} & \boxed{h7} \end{bmatrix} \right\rangle \end{bmatrix}$$

Elementary predications The primary units of information in an MRS representation are its *elementary predications (EPs).* Each *EP* consists of a named relation (or predicate) and any number of associated arguments, conventionally labelled ARG*N*. The values of these arguments are logical variables and belong either to the type *event* (e) or *entity/individual/referential index* (i). These logical variables establish links across EPs. They can also encode for additional *variable properties* relating, among other, to tense, aspect, illocutionary force, person, number, or gender, depending on whether the variable relates to an event or to an entity. Additionally, all EPs have a unique distinguished argument ARG0 which, informally, function as a sort of index to the predicate itself. Generally, the ARG0 of the main predicate forms the top index of the MRS representation (found under HOOK|INDEX) Within the DELPH-IN TFS geometry, EPs are generally found under the feature RELS. Although they formally form an unordered bag (i.e., do not have fixed order or require uniqueness), they are implemented as ordered lists.

Handle constraints In order to capture scopal information, *labels* are introduced (on the EP feature LBL) with a so-called *handles (hN)* as their values. These handles form the values of scopal arguments (e.g. RSTR and BODY), in which case they are sometimes referred to as *holes*, the highest scoping EP shares its handle with the *hook* feature LTOP (*local top*), and EPs that have the same handle are interpreted as conjuncts. In a well-formed MRS, handles can be identified (in one or more ways) such that the dependencies between EPs form a tree. Constraints on possible interpretations of scopal relations are introduced under HCONS. Here, the type *qeq (equality modulo quantification)* is used to indicate that either the value of LARG l fills the hole h in HARG, or that l is indirectly linked to h via one or more *floating quantifiers* (for which the reader is referred to Copestake et al. (2005)). In both cases the effect is to convey that one handle outscopes another.

Hooks Already mentioned above, the features INDEX and LTOP belong to the type *hook*. The function of this type is to 'publish' those parts of the

EP list that the grammar's rules need access to when building up semantic representations. Generally, these parts include the index of the head of the current phrase and the identity of the highest-scoping handle. There are several other features which *hook* can be specified for, but these are not discussed here. As far as semantic composition in the *Depicto* system is concerned, knowledge of INDEX and LTOP is sufficient.

2.2.3 The LinGO Grammar Matrix starter kit

Designing large-scale TFS grammars from scratch is hard work, regardless of the precise formalism, implementation or framework used. In fact, as Bender et al. (2002) point out, the most extensive DELPH-IN grammars to date, viz., the English Resource Grammar (ERG; (Copestake & Flickinger 2000)), the Jacy grammar of Japanese (Siegel & Bender 2002), and the German Grammar (GG) (Müller & Kasper 2000), each required between 5 and 15 years of development by experts before being ready for use in real-life applications; and some, such as the ERG, are still undergoing refinement. This lengthy and costly development process is – or, rather, *was* – largely due to the fact that, while these grammars use the same formalism and framework, they were developed more or less independently of each other, resulting in time being spent on design problems (e.g., the structure of the type hierarchy, the HPSG feature geometry, the implementation of semantics as well as technical devices such as lists and 'appends' – to name a few), for which compatible solutions had already been devised. As an alternative to the cycle of wheel re-inventing, Bender et al. (2002) present the first version of the LinGO[5] Grammar Matrix starter kit (it has evolved considerably in the years since, undergoing several refactoring stages), a language-independent grammar resource designed to facilitate both the initial and long-term development of grammars written in the DELPH-IN formalism. At its core, the starter kit comprises a set of typologically agnostic generalizations across the largest DELPH-IN grammars,

[5] The CSLI (Stanford University) Linguistic Grammars Online (LinGO) laboratory is one of the main founders of the DELPH-IN project

including definitions for basic linguistic objects, common best practices (e.g., type naming conventions), and analyses that adequately generalize to other languages. The idea is for new DELPH-IN grammars[6] to be able to build on top of this core, i.e., by inheriting from it and adding more specific constraints until (a subset of) the grammatical requirements of a given language are met. The remainder of this section takes a slightly closer look at the Matrix, its core in particular, and introduces the script-based customization system that its developers have since devised in order to make prototyping with it even faster.

2.2.3.1 Inside the Matrix

As a starter kit, the Grammar Matrix includes all the components needed to construct a DELPH-IN grammar (see section 2.2.1.3). The most important component is a file called `matrix.tdl`, which defines the top of the grammar's TFS hierarchy. Aspects of the grammar that have been defined here since the first version of the Matrix (Bender et al. 2002) include the general feature geometry of signs, technical devices (e.g., list manipulation), basic lexical and phrasal rule types (both unary and binary), headed and non-headed rule types, phrase-internal order rule types (e.g., head-left and head-right), constructions corresponding to the HPSG schemata (i.e., Head-Subject, Head-Complement, Head-Specifier, and Head-Modifier rules[7]), all of them underspecified for word order. The `matrix.tdl` file, finally, includes a semantic component, where a hierarchy of MRS relation types is defined, as well as types and constraints for the propagation of semantic information throughout the phrase structure tree, a representation of illocutionary force, and rule types allowing grammar rules to make semantic contributions (Flickinger & Bender 2003).

For the most part, the definitions in `matrix.tdl` were originally based on

[6] A slight anachronism, as the DELPH-IN project did not take off until later

[7] Note that the flavor of HPSG used in the LinGO Matrix does not follow proposals to roll the Head-Modifier and Head-Specifier schemata into a single *head-functor* rule, as in Van Eynde (1998)

those found in the English Resource Grammar (ERG). In order to make the Matrix as language-independent as possible, however, the Japanese grammar Jacy, a smaller grammar for Spanish, and general knowledge about typological variation were used to guide decisions about which definitions to keep, discard or underspecify (Bender et al. 2002; Fokkens 2011). An example of the latter is the separation of rules concerning immediate dominance (i.e., constituent structure) on the one hand and linear precedence (i.e., constituent-internal ordering) on the other. Several large-scale grammars have been constructed using the original Grammar Matrix. These include grammars for Norwegian, Spanish, Modern Greek, Portuguese, Korean, and Huasa (Fokkens 2014). Based on feedback from developers working with increasingly diverse languages, the Matrix has already undergone several update cycles. In order to bring new improvements to existing Matrix-based grammars, several script-based tools have been developed that modify all the components of a given grammar to reflect changes in the updated Matrix core, such as – in recent years – the introduction of *disjunctive head types*. The problems which these scripts have to overcome are by no means trivial.

Also included with the Matrix starter kit, finally, are configuration and parameter files for easy start-up with a number of DELPH-IN processors. Originally, these were just for the LKB grammar engineering environment (Copestake 2002), but recent versions of the Matrix also support the ACE system (Packard 2015) and the PET parser. The LKB and ACE systems are introduced in section 2.2.4.

2.2.3.2 The customization system

By providing a solid, language-independent foundation, the Grammar Matrix significantly reduces the initial cost of and level of expertise needed for developing new processable grammars, which, as explained above, essentially extend the core grammar to cover the specific phenomena and properties of a given language. Drawing on experience teaching students [8] to use the Matrix, its developers notice that parts of this process of extension can be automated (Bender 2007). Language-specific analyses of crosslinguistically variable phenomena are not implemented manually, but parameterized and generated by a script, or rather, *several* scripts, each representing a so-called *module* (today: *libary*). First proposed in Bender & Flickinger (2005), this approach formed the basis for the Grammar Matrix customization system[9] (Bender et al. 2010). The customization system uses a web-based questionnaire to dynamically (i.e., adapting the HTML code based on other answers) elicit information from users about the language they want to model. This information corresponds to the modules for basic word order, yes-no questions, sentential negation and lexical items, including nouns, transitive and intransitive verbs, auxiliaries, determiners, negation markers and case marking adpositions. Input from the questionnaire is stored in a *choices* file. Once validated, i.e., when no inconsistencies or missing information are identified, the choice file is passed to the customization script, which creates an initial custom grammar. The user can test this grammar remotely by generating sentences with it. If these are all or mostly grammatical, the user can choose to download the complete grammar. From that point on the grammar behaves like any other DELPH-IN grammar. (For a clear and relatively up-to-date overview of the system in its entirety, please see Fokkens (2014), whose research on 'Metagrammar engineering' is largely based on the idea of script-based grammar generation.) As chapter 3 explains in more detail, both the *Depicto* system's grammars stem from a simple SVO-language grammar that was initially set up with the aid of the customization system.

[8] Since 2004, the Grammar Matrix has been used in a grammar engineering course taught at the University of Washington (http://courses.washington.edu/ling567/)

[9] http://www.delph-in.net/matrix/customize/matrix.cgi

2.2.4 The ACE processor

The DELPH-IN repository includes a number of ready-made made tools for parsing[10], two of which, viz., the LKB (Copestake 2002) and the ACE (Packard 2015) system, additionally support generation as well as semantic transfer (the latter introduced in the next section (2.3). The LKB was originally designed as a tool for grammar development, serving as a so-called *GDE* (a play on *IDE*[11]). The ACE system, by contrast, was developed for speed. It uses the same processing algorithms as the LKB (barring a handful of optimization techniques), but performs about 15 times faster (Packard 2015). Since users tend to like their translation tools as responsive as possible, the ACE processor is the obvious choice for deploying such systems in production. As far as the grammar development process is concerned, here, too, the ACE processor has proven itself to be a suitable tool. The remainder of this subsection introduces the ACE system in slightly more detail and explains how it can be used for effective grammar development as well. Before we proceed, however, it ought to be said that, while the following does include details about parsing and generation algorithms, the ACE system is to all intents and purposes treated as a 'black box' within the *Depicto* system. It does what it promises, and it does it well; but its codebase and API, both written in C, are best left to the more initiated.

The Answer Constraint Engine[12] (ACE) (Packard 2015) is a *relatively* recent addition to the open-source DELPH-IN ecosystem. Initial development, which began in 2004, was for proprietary uses, and the software was not openly released until 2011 (under the open-source MIT license). Although essentially a one-man project, the ACE processor is still actively being developed, as its release notes suggest[13], as does the DELPH-IN wiki, which hosts an up-to-date wish list of improvements that other developers would like to see incorporated. ACE runs on GNU/Linux systems and stands out as being the *only* DELPH-IN

[10] for a visual comparison see http://sweaglesw.org/linguistics/delphin-engines.html

[11] (Integrated Development Environment)

[12] http://sweaglesw.org/linguistics/ace/

[13] http://sweaglesw.org/linguistics/ace/download/RELEASE-NOTES-0.9.20

processor that also supports Mac OS X (my main development environment). Interacting with the system, which is available as a ready-made binary, is done via the command line. Both the inputs and outputs are plain text. In parsing mode, ACE takes as input a stretch of text and outputs, if possible, one or more MRSs. In generation mode, it takes as input an MRS and outputs one or more instances of generated text. In *transfer mode* (i.e., the 'bridge' in transfer-style rule-based translation systems, this concept introduced in the next section), both the input and the output are MRSs. One of the many practical advantages of the ACE system is that the output of given mode can be passed, if appropriate, to the input of an arbitrary mode: multiple calls to the ACE binary can be chained easily by means of Unix 'pipes' (i.e., |'s).

For parsing, the ACE system uses a classic, agenda-driven chart parser, based on Kay 1986 (Crysmann & Packard 2012). To be precise, this is a variation on the bottom-up Cocke–Younger–Kasami algorithm, which is also used by other DELPH-IN processors. ACE also comes with REPP support (*Regular Expression Pre-Processing*) and built-in part-of-speech tagging. (These features are of less use to the more predictable input expected by the *Depicto* system.) For generation, ACE uses a chart generation algorithm described in Carroll et al. (1999). The input to the generation system is a grammar and the semantics of the utterance to be generated (expressed in MRS). The generator's output is the list of *all* strings which are related to the input MRS by the grammar. The generation process happens in three stages: lexical-lookup (entries with corresponding predicate name values are retrieved from the lexicon and instantiated on the chart generator); chart generation; and modification insertion (whereby morphological lexical rules are finally applied) (Fokkens 2014). Two efficiency measures are taken to "combat the exponential worst-case complexity of the chart generation algorithm" (i.e., if the input MRS is too underspecified) (Crysmann & Packard 2012). Described in Carroll & Oepen (2005), these are *Ambiguity packing under subsumption* and *Index accessibility filtering.* My knowledge of parsing and generation techniques is too limited to provide an adequate explanation of these. Therefore, please see the related work instead. For transfer, the ACE system uses the LOGON

MRS-rewriting machinery. This is explained in the next section.

2.2.4.1 Why not the LKB, and how to use ACE for development

On paper, the LKB jumps out as the most suitable tool for designing and debugging DELPH-IN-style grammars. Not only does it include a graphical user interface, it also provides a rich set of tools for visualizing various components of the grammar, such as the type hierarchy, instantiated TFSs, parse trees, generated trees, and parse/generation charts – to name the most useful. Particularly useful in the context of debugging is ability to interactively step through the parse chart. However, in developing the *Depicto* system, I found the LKB lacking in a number of respects. In the first place, its GUI is relatively outdated by modern design standards, does not render well on HiDPI screens, and its reliance on the CLIM (Common-Lisp Interface Manager) library restricts its portability to Linux, 64-bit versions of which have additionally to deal with the problem of missing 32-bit dependencies. The alternative is to interact with the LKB via its Lisp console. This console runs in, but behaves very differently to, a standard Unix shell/terminal, and is only really usable through mediation of the *Emacs* text editor. In other words, in order to circumvent the LKB's GUI one must have some knowledge of Lisp (which I do not to a sufficient degree) and one must be comfortable working with Emacs (something of a presumption, given the wide range of capable text editors available nowadays). And yet, unless one is able to circumvent the GUI, simple tasks such as reloading (recompiling) a grammar, much less multiple grammars at the same time, become highly repetitive and time-consuming. Setting up a machine-translation system of the kind developed here *is* possible[14], but it requires three separate *Emacs* windows running a separate LKB session each and a fair amount of additional Lisp code, leaving a fair amount of room for error. The process is so convoluted, in fact, that I was able to get it working *once*.

[14] http://moin.delph-in.net/MtSetup

Fortunately, the ACE system can be set up so as to get much of the same visualisation functionality that makes the LKB so useful. For this, the LUI[15] (Linguistic User Interface) application is required. Once downloaded and installed on the PATH, it allows ACE to start up in so-called *LUI mode* in combination with any of its three other modes [16]. In this mode, ACE can instantiate visual browsers for constituent trees, feature structures, MRSs, parse charts, and local supertypes and subtypes. Unlike the LKB, ACE-in-LUI-mode is not able to visualize the entire type hierarchy. This is not really a limitation, however: when dealing with large-scale grammars, there inevitably comes a point when the type hierarchy simply becomes too big to be visualized informatively. In comparison to the LKB, LUI mode also adds some new features, the most useful of which is drag-and-drop interactive unification. Combined with the LUI application, as well as a smattering of simple ad hoc bash script here and there, the ACE system becomes every little bit as useful for grammar development as the LKB, if not more.

2.3 Transfer-based MT: the LOGON approach

As explained in Section 2.2, DELPH-IN grammars are designed for both parsing and generation, bi-directionally mapping between surface strings and semantic representations (in the MRS format; see section 2.2.2). This bidirectionality has many uses. For example, it is possible to check whether a given grammar is prone to overgeneration (i.e., is not sufficiently constrained) simply by letting it generate from an MRS obtained by parsing with the same grammar. (The ACE processor, with its support for 'pipes', makes this particularly easy to set up (see section 2.2.4).) A more advanced application, which is developed here, similarly involves chaining the parsing and generation steps, but uses *different* grammars for each, which has the effect of translating the input surface string into a semantically equivalent string licensed by a

[15] http://moin.delph-in.net/LkbLui

[16] http://moin.delph-in.net/AceLui

different grammar. This comes fairly close to the essential function of a rule-based machine translation (RB**MT**) system. However, this approach is somewhat simplistic, since the output of an RBMT system's *source-language* grammar is generally not compatible with its *target-language* grammar, i.e., the grammar used for generation. In the present case, the challenge consists in the fact that the MRS framework is not intended to be used as an *interlingua* (Copestake et al. 2005): MRS representations are structurally codified with regard to a number of conventions, but their semantic content (viz., the (lexicalized) predicate names found on their list of *Elementary Predications*; see section 2.2.2) is largely language-specific. Since interlinguas are, in fact, rare, most RBMT systems make use of an intermediate *transfer* component, which mediates between the output of the source grammar and the input to the target grammar, transforming as required the (semantic and/or syntactic) data structures passed between them (Vandeghinste 2008).

For DELPH-IN-based machine translation systems, an open-source MRS-oriented transfer mechanism is already available. Originally developed within the context of the LOGON Norwegian-English MT project (Lønning et al. 2004), this system rewrites MRSs step by step until one or more MRSs are derived from which a target-language grammar can generate. The rewrite process itself is determined by a set of bilingual rules defined in a *transfer grammar* (Oepen 2008) for a given language pair. Implementations of the transfer engine are found in both the LKB and (since recently) the ACE processor (introduced in section 2.2.4).

The rest of this section takes a closer look at the LOGON[17] transfer system. First, it provides an informal introduction to the general formalism for MRS rewriting, focusing on those characteristics of the formalism that are best kept in mind when developing a transfer grammar, and gives a schematic description of the rewrite rules that constitute this type of grammar. Next,

[17] LOGON may be understood as a branch of the DELPH-IN consortium. For just over a decade, it has worked on developing a hybrid machine translation infrastructure suitable for MRS-oriented grammars. The project started off as a collaboration among three Norwegian universities, although it has since attracted developers from other areas.

we see how these rules are implemented descriptively, namely, as typed feature structures, and how this benefits the internal organisation of the transfer grammar. The last subsection shows how this *object-oriented* style of organisation makes it possible for developers to provide resources that can serve as starter kit for the development of new transfer grammars, in much the same vein as the LinGO Grammar Matrix (see section 2.2.3), albeit in a more rudimentary form. For the sake of clarity, finally, it should be observed that the LOGON transfer system forms merely one part of the larger LOGON MT infrastructure[18]. The complete system combines a symbolic basis with stochastic extensions, which provide, inter alia, probabilistic rankings in the parse, transfer, and generation components, as well as fallbacks for when the transfer system fails (Bond et al. (2011), Oepen et al. (2002)).

2.3.1 MRS rewriting

The LOGON transfer formalism provides a special-purpose graph rewriting system[19] in which individual unification-based rewrite rules relate parts of a source-language MRS to the corresponding parts of a target-language MRS (Oepen 2008). This is a stepwise process, with rewrite rules applying in the order in which they are specified in the transfer grammar, as well as 'consuming' their input, i.e., replacing it with the corresponding target-language MRS. As a result, the application of each rule "changes the 'state of the universe' visible to subsequent rule applications" (Oepen 2008). (This contrasts with the phrasal and lexical rules encountered in the previous section, which 'combine' rather than 'consume' their daughters, and, in fact, imposes upon the transfer grammar the requirement that rules be ordered in increasing specificity, so that more general rules are applied first.) In fact, the structures manipulated throughout most of the transfer process consist of a mix of the source-language MRS and the target-language MRS. The input and output

[18] http://moin.delph-in.net/LogonTop

[19] Formally, MRS structures constitute *directed acyclic graphs* (for more detail, see section 2.2.2)

specifications of individual rules can access arbitrary structural properties of the MRS graph, as well as establish bindings via structure sharing, by means of which properties can be 'carried over' from the input to the output. Because an MRS can contain multiple parts for which the same rule is relevant, rule application in the LOGON system is not only ordered, but breadth-first (i.e., all possible applications of a given rule are considered before moving on to the next rule). Finally, to allow for the inverse situation, that is, where multiple (possibly optional[20]) rules apply to the same MRS fragment, the system supports backtracking (Oepen 2008).

Turning to the rewrite rules themselves, which are referred to to as MRS *Transfer Rules* (or MTRs), these constitute four-tuples of (partial) MRS descriptions which correspond, as schematized in (2.7), to the rule's *Input*, *Output*, and two optional components that condition the applicability of the rule either by constraining it with additional *Context*ual information, or by blocking it entirely by means of a *Filter* (Oepen 2008).

(2.7) $[Context]\ Input\ [Filter]\ \longrightarrow\ Output$

The general idea is that, for an MTR to be applied, the MRS description provided by its *Input* component must be compatible with the MRS (structure) currently being processed by the system. (This can be either the original MRS input to the transfer system or an intermediate structure obtained from earlier MTR applications.) If the partial MTR's *Input* component can be unified with this MRS (that is, if the the two MRS graphs can be structurally aligned), then the transfer rule is invoked, or 'fired'. The unified (aligned) substructures are consumed, and the MRS description on the *Output* component specifies what to insert in their place (Oepen 2008). Those parts of the transferred MRS which are not aligned with the rule's *Input* are not affected by it.

Originally built as an extension of the LKB system (introduced in section

[20] Rewrite rules can be marked for optionality. This is useful when dealing with an MRS fragment that has several possible translations. Optional rules, which causes the rewrite process to 'fork' into parallel branches, are generally provided as a series. The last rule in this series is marked as non-optional so as to terminate it (Oepen 2008).

2.2.4), the LOGON transfer system describes MTRs as typed feature structures (TFSs, see 2.2.1.2). Accordingly, transfer grammars, which may be understood as ordered sequences of MTR descriptions, are written in the $\mathcal{TDL}$ specification language[21] (see 2.2.1.2). Figure 2.8 shows the TFS description of a hypothetical MTR that translates English *Dog* to Dutch *Hond.* The attribute-value matrix version and the $\mathcal{TDL}$ version are given side by side.

(2.8) A simple transfer rule (MTR)

a. AVM representation of MTR

$$\begin{bmatrix} \textit{mtr} & \\ \text{INPUT} & \begin{bmatrix} \textit{mrs} & \\ \text{RELS} & \left\langle \begin{bmatrix} \textit{ep} & \\ \text{LBL} & \boxed{1} \\ \text{PRED} & \textit{dog\_n\_rel} \\ \text{ARG0} & \boxed{2} \end{bmatrix} \right\rangle \end{bmatrix} \\ \text{OUTPUT} & \begin{bmatrix} \textit{mrs} & \\ \text{RELS} & \left\langle \begin{bmatrix} \textit{ep} & \\ \text{LBL} & \boxed{1} \\ \text{PRED} & \textit{hond\_n\_rel} \\ \text{ARG0} & \boxed{2} \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix}$$

b. $\mathcal{TDL}$

```
hond_mtr := noun_mtr &
  [ INPUT.RELS < [ LBL #lbl,
                   PRED "_dog_n_rel",
                   ARG0 #arg0 ] >,
    OUTPUT.RELS < [ LBL #lbl,
```

[21] Upon which the LOGON system, in fact, builds, adding syntax for regular expression matching and introducing a 'copy' operator. Like, unfortunately, much of the LOGON system, these features are scarcely documented; however, some information can be found at http://moin.delph-in.net/AceTransfer

```
                              PRED "_hond_n_rel",
                              ARG0 #arg0 ] > ].
```

As typed feature structures, transfer rules form a multiple-inheritance hierarchy with strong typing as well as appropriate feature constraints for MRSs and MTRs. (Incidentally, this is what sets the LOGON system apart from work on semantic transfer in the VerbMobil MT project (Wahlster 2013), whose main ideas the LOGON project largely follows (Lønning et al. 2004)). In close analogy to the methodology used by DELPH-IN-style grammars (in Section 2.2), such typing makes it possible to state generalizations over transfer rules and to enforce global well-formedness conditions, making it easier to write large-scale transfer grammars that, additionally, allow for underspecification.

Similarly to DELPH-IN grammars, the LOGON system makes a distinction between, on the one hand, rule *descriptions* (comprised by the type hierarchy) and, on the other, rule *instances*, i.e., those rules which are instantiated by one or more typed descriptions. Within the context of the LOGON system, this corresponds to the difference between general *patterns of translational correspondence* and actual *transfer rules*, the latter to be converted from expanded TFSs to processable rules at compile-time (Oepen 2008). It is these rules whose order determines the rule application process. Unsurprisingly, therefore, they are identified separately by the associated configuration file.

2.3.2 The Transfer Matrix

Bond et al. (2005) describe preliminary work on a Japanese-to-English hybrid MT system (JaEN) that incorporates the LOGON transfer engine. Although the transfer rules used for this component are for the most part automatically acquired from aligned corpora and bilingual dictionaries, the type hierarchy from which they inherit has a highly re-usable, language-independent core that is derived from the Norwegian-to-English transfer grammar (Lønning et al. 2004) (for which the LOGON system was originally developed). This core transfer grammar functions similarly (*mutatis mutandis*) to the LinGO Grammar Matrix starter-kit (section 2.2.3), as its unofficial name, viz., the *Transfer Matrix* (Bond et al. 2011), reflects. This collection of generic transfer types, which also includes specifications for basic MRS and MTR types, forms the foundation of the (highly experimental) transfer grammar used in the *Depicto* system, developed here. Other parts of the JaEn system's transfer grammar have proven valuable as well. To understand this, one must consider two problems. In the first place, documentation relating to the *practical* implementation of LOGON-style grammars is relatively scarce (as the developers of the LOGON system admit themselves here and there [22]). In the second place, LOGON-based MT systems are rarely both large *and* completely open-source. Thus, although one is largely forced to learn by example (that is, by studying other grammars), there are not many grammars that can serve as a sufficiently representative example. The JaEn system (Bond et al. 2011) is the exception. Its transfer rules have served as the main source of inspiration for experimentation in the transfer component of the *Depicto* system.

[22] http://moin.delph-in.net/LogonReports

Chapter 3

Depicto I: Deep parsing *Sclera*

This is the first of two chapters that introduce the *Depicto* pipeline by looking at each of its consecutive grammar modules. The focus here is the module responsible for parsing a string of *Sclera* symbols and extracting as much semantic information from it as possible. First, Section 3.1 shows how *Sclera* symbols are passed to the parser as strings of identifiers. Next, Section 3.2 details the initial stage of designing a grammar that can be used to process such strings. A (hypothetical) grammar of *Sclera* is sketched out, the LinGO Grammar Matrix customization system is used to quickly obtain a working grammar prototype, and an initial set of modifications is made so as to re-introduce missing relations of quantification. As a brief intermezzo, Section 3.3 gives a walkthrough of the MRS output yielded by the *Sclera* analysis module. Finally, Section 3.4 describes three case studies which demonstrate how the initial model of *Sclera* can be extended with a view to (a) increasing its semantic accuracy and (b) incorporating complex, or 'compound', pictographs. Note that, all in all, this chapter concerns the most innovative contribution of this thesis. As a result, in contrast with the chapter that comes after, it may at times be observed that brevity is foregone in favor of extra clarity.

3.1 Parser input: A note on picto identifiers

The *Depicto* pipeline analyses sequences of selected pictographs not as actual images, but as strings of identifiers with which individual pictographs are associated. These identifier *tokens* are matched against lexical entries defined on the parsing grammar's lexicon. (Unlike natural language tokens, pictograph identifiers are not subject to inflectional rules, spelling mistakes or other sources of variation, so matching is non-partial.) In Matrix-based grammars, the specific part of the lexical entry against which identifiers are matched is a string value specified on the feature STEM[1]. To illustrate this, (3.1) shows the $\mathcal{TDL}$ description of a typical lexical entry, of which the AVM equivalent is given directly below, in (3.2).

(3.1)
```
dog := common-noun-lex &
  [ STEM < "dog" >,
    SYNSEM.LKEYS.KEYREL.PRED "_dog_n_rel" ].
```

(3.2)

$$\left\langle \text{dog}, \begin{bmatrix} \textit{common-noun-lex} & \\ \text{STEM} & \langle \text{`dog'} \rangle \\ \text{SYNSEM} \mid \text{LKEYS} \mid \text{KEYREL} \mid \text{PRED} & \text{`\_dog\_n\_rel'} \end{bmatrix} \right\rangle$$

Pictograph identifiers are passed to the parser as simple character strings (i.e., no complex data structures are involved), separated by any number of horizontal white spaces.

The naming convention to which these pictograph identifiers adhere warrants slightly more detailed discussion, not least because there are, in fact, several naming systems to choose from.

[1] Non-Matrix grammars use a functionally identical feature, but use different names for it. For instance, the English Resource Grammar (Flickinger et al. 2014) uses PHON, which is in accordance with (Pollard & Sag 1994) as well as the off-shoot HPSG 'variant' SBCG (Boas & Sag 2012)

The first, and easiest, option (at least, initially) involves identifying pictographs with the names of the files in which they are individually stored. These filenames, whose extension is stripped away, either convey some part of the pictograph's sense or describe the appearance of the pictograph itself. An example of a hypothetical lexical entry where the identifier value of STEM is thus formatted is given in (3.3).

```
(3.3)  thunderstorm := common-noun-lex &
         [ STEM < "thunder", "and", "lightning" >, ; from: "thunder and lightning.png"
           SYNSEM.LKEYS.KEYREL.PRED "_thunderstorm_n_rel" ].
```

Ignoring the fact that not all filenames correspond to the image's meaning, this naming strategy has two major drawbacks In the first place, it restricts the parser's grammar model entirely to the *Sclera* symbol set, whereas the aim of *Depicto* is to be applicable to at least a subset of other symbol sets, too. In the second place, filename-based identifiers make it, if not impossible, needlessly complicated to generalize over pictographs that are synonymous as far as the system is concerned: multiple lexical entries bearing the same semantic meaning could be provided for each picto 'synonym', but this would be a costly procedure inevitably requiring manual intervention, since synonymy is not always suggested by the filename alone.

A different approach to naming pictograph identifiers piggy-backs on work by Vandeghinste & Schuurman (2014) on linking pictographs to the English WordNet database (Miller 1995) and international variants thereof, such as the Cornetto database for Dutch (discussed in Section 1.2.2.1; (Vossen et al. 2007)). Just as in the Text2Picto and Picto2Text systems (again, see Section 1.2.2.1), pictographs are identified with the ID tag of a linked *synonym set*, as the value of STEM in (3.4) shows. Synset IDs vary across different WordNet databases, but they are easily made compatible with the right database query. Moreover, the pictograph–WordNet dictionaries developed by Vandeghinste & Schuurman (2014) are additionally linked to the *Beta* symbol set (discussed in Section 2.1.3), so that a grammar model of *Sclera* in which pictographs are identified by linked synset IDs has the potential, in theory, to be applied to *Beta* as well.

(3.4)
```
door := common-noun-lex &
  [ STEM < "d_n-27669" >,
    SYNSEM.LKEYS.KEYREL.PRED "_door_n_rel" ].
```

Still in an early stage of prototyping, the *Depicto* system does not make use of synset IDs, yet. This has to do, in part, with the fact that non-content words are omitted from WordNet. As the inventory in Section 2.1.2 suggests, this does not present such a big problem for *Sclera*, with its general lack of prepositions, determiners, auxiliaries, etc., although personal pronouns do exist in *Sclera*, as well as a small group of operators (conveying, e.g., conjunction, negation, lack of permission). For these, an additional set of identifiers must be devised. Granted, this does not present too much of a hurdle. More relevant to the motivation for not using synset IDs in the current version of *Depicto*, yet similarly practical in nature, is a problem that comes up in the early stages of grammar development, when testing a grammar is most easily done *not* by some automatic process, but by manually typing out test strings that include the phenomenon being tested for. Synset IDs, as (3.4) might suggest, are not particularly easy to remember and, much like an overly complicated password, comprise enough unpredictable characters to be prone to typographic error. The result is that manual methods of grammar testing quickly become tiresome.

To aid the process of prototyping, therefore, pictographs are currently passed to the analysis module as simple English lemmas, as shown in (3.5) and (3.6). (In the case of complex pictographs, the constituent semantic parts are additionally separated by *underscores.*) This approach represents a comfortable compromise between the comparative readability of filename-based identifiers and the (currently unexplored) potential of being linked to the WordNet database that comes with synset IDs (from which the current approach gets its lemmas). This compromise is only temporary, however.

Although the lexicon of the grammar model is small enough not to cause any problems at the moment, further growth will necessitate a transition to a synset ID-based naming approach.

(3.5)
```
dog := common-noun-lex &
  [ STEM < "dog" >,
    SYNSEM.LKEYS.KEYREL.PRED "_dog_n_rel" ].
```

(3.6)
```
walk_v := common-verb-lex &
  [ STEM < "walk" >,
    SYNSEM.LKEYS.KEYREL.PRED "_walk_v_rel" ].
```

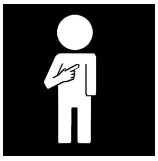
3.2 A first crack at a grammar of *Sclera*

This section details the construction of a constraint-based grammar model that can be used to 'deep' parse strings of *Sclera* symbol identifiers. Drawing entirely on the DELPH-IN resources introduced in Section 2.2, this grammar is used by the analysis module to produce semantic representations of the *Sclera* input. Before looking at the modelling process itself, however, we briefly examine the inherent assumption that *Sclera can be modelled* in the first place. The first next section presents an illustration of a number of 'typical' *Sclera* strings and shows how these can be analysed in natural language terms. The phenomena which a simple grammar should aim to cover are outlined here as well, with the necessary caveats here and there. With these preliminaries in place, we see how an initial starter grammar is obtained through the Matrix customization system (see Section 2.2.3.2) and how this is extended to account for the most salient structural properties of the *Sclera* input as well as basic word order patterns. Next, we look at one of the first major milestones in the construction of the *Sclera* grammar, namely, the re-introduction of (non-existent) determiners onto the semantic representations of parsed *Sclera* utterances (recall the inventory of *Sclera* provided in Section 2.1.2). This process of *rule-based semantic enrichment* (a notion borrowed from Crysmann & Packard (2012)), of which two more examples are given in Section 3.4, goes a long way in establishing the viability of an approach that, as we explain, aims to treat *Sclera*, or indeed any pictographic language, not simply as an underspecified analogue of some natural language, but as a language in its own right, with its own rules and idiosyncrasies. This part of the modelling process is therefore discussed in a separate subsection (3.2.3).

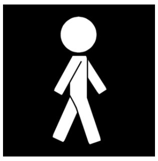
3.2.1 *Sclera* as a (natural) language

Building on the inventory of *Sclera* given in Section 2.1.2, we now look at how individual pictographs can be combined to form meaningful sequences, or strings, much like the words of a given language can be combined into arbitrarily complex utterances. In fact, with much of the *Sclera* set consisting of pictographs depicting concepts that are generally the domain of verbs and nouns, it is possible to project large parts of our knowledge of natural language syntax onto the surface structure of *Sclera* strings, as (3.7) illustrates with a simple example.

(3.7)



I



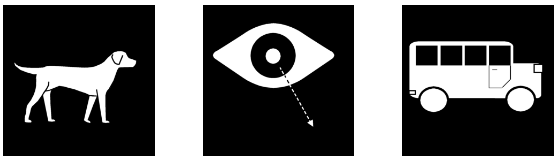
walk

‘I walk.’

The picto sequence in (3.7) can be analysed structurally as comprising an intransitive ‘verb’ picto *walk* and a ‘pronoun’ picto *I*, which serves as the subject of the verb.

Note that, despite appearances, the pictographs themselves do not encode mutual grammatical agreement. This can be seen more clearly in the example in (3.8), which shows a similar single ‘picto clause’, except here the ‘verb’ picto, *see*, depicts a process that is conceptually transitive, where the ‘noun’ picto *bus* serves as object.

(3.8)



dog *see* *bus*

‘The dog sees the bus.’

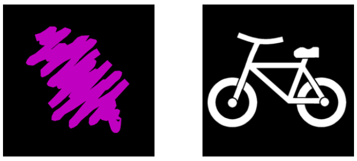
Ignoring for the time being the missing determiners in (3.8), it is, so far, possible to conclude that ‘verb’ pictos can be analysed in much the same way as ordinary verbs with regard to their valency, despite the fact that, by default, they do not encode for number, person, tense, aspect or mood, nor are their arguments marked for number, person or gender (leaving quantification aside for now). Note, too, that both examples are internally structured according to a *subject-verb-object* (henceforth *SVO*) element order. Since *Sclera* is virgin territory as far as its hypothetical syntax is concerned, there is no evidence to suggest that this is the only word order that a grammar of *Sclera* should expect, much less that it is the one which ID users would prefer; however, in the examples that follow, as well as in the grammar model whose design these examples ultimately inform, *SVO is the word order that is assumed*[2].

Structurally, ‘adjectival’ pictographs can also be hypothesized to have combinatory potential, as examples (3.9) and (3.10) suggest. Here, *purple* and *happy* can be seen to modify the ‘noun’ pictos *bike* and *dog*, respectively.

[2] As discussed later, in Section 5.2.1, this is a serious limitation of the current version of the system. Nevertheless, it is currently maintained as the only permitted word order because it prevents structural ambiguity (which arises quickly in the absence of inflection and other syntactic markings) and allows the analysis of valency and word order to be implemented relatively easily. Moreover, the combination of multiple permitted word order patterns in a single grammar represents a complex undertaking, although, ultimately, it is a goal of the *Sclera* grammar.

Of course, this is not the only function that such adjectival pictographs can fill: they can also serve as 'nouns' and predicative complements, the latter function demonstrated in (3.11). (Note that, while interesting, 'adjective' pictos form a minority within the *Sclera* set. Further, the predicative use of adjectives is not subject to modelling.)

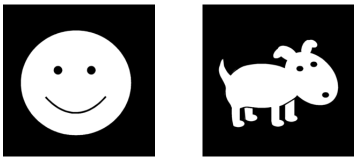
(3.9)



purple *bike*

'The purple bike.'

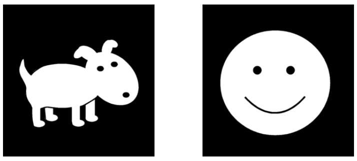
(3.10)



happy *dog*

'The happy dog.'

(3.11) *predicative use of adjective picto



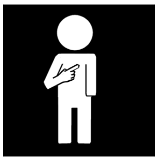
dog *happy*

'The dog is happy.'

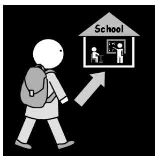
The foregoing examples all have in common that they involve pictographs depicting a conceptual simplex: each pictograph corresponds to a single concept and (in English at least) to single word. However, recall from the inventory of *Sclera* given in Section 2.1.2 that *Sclera* additionally comprises a large set of pictographs which depict conceptual *complexes*. These 'complex' pictographs generally center on a conceptual process (verb) and ‘bundle’, as it were, one or more concrete arguments associated with this process.

There are several kinds of complex pictogaphs. The least problematic for a traditional constituent-based approach is illustrated in (3.12).

(3.12)



I

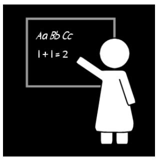


go+school

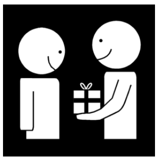
‘I $\left\{\begin{array}{l}\text{go}\\ \text{am going}\end{array}\right\}$ to school.’

The complex picto in (3.7), *go+school*, depicts two concepts: a process of ‘going’, and the target of movement, i.e., ‘school’. In English (and, incidentally, in Dutch also) the latter is expressed as an obligatory locative complement, set off by a preposition. Syntactically, complex pictographs of this kind can be analysed as partially ‘saturated’ verbal constituents that are still ‘seeking’ an element to fill their subject slot. Other kinds of complex pictographs, however, may combine with optional complements, as illustrated by *give+present* in example (3.13). Such complex pictographs are slightly less evident, but are generally compatible with SVO phrase structure analyses.

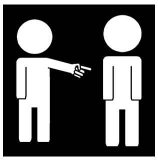
(3.13)



teacher



give+present

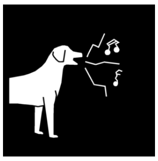


you

‘The teacher gives a present to you.’

Other pictographs depict conceptual complexes that correspond to entire clauses, such as *dog+barks* in (3.14).

(3.14)



dog+barks

'The dog barks.'

Like the absence of determiners/quantifiers, complex pictographs are a feature of *Sclera* that sets it apart from most 'true' natural languages. Unsurprisingly, therefore, including these pictographs in a grammar model of *Sclera* requires some slightly different machinery, particularly to make 'constituent' concepts accessible to other parts of the grammar, so that sequences like in (3.15) are grammatical. We return to the integration of complex pictographs in Section 3.4.3. Easier to analyse, simplex pictographs will form the basis of the initial grammar.

(3.15)

happy

dog+barks

'The happy dog barks.'

To the best of my knowledge, there currently exist no 'corpora' of pictographic text. The reasons for this are not hard to intuit. Popular and accessible pictograph-to-text systems are few and far between (hence, the contribution of this thesis!). If data about how these systems are used collected, it is not shared. Yet even if it were, pictographic symbol sets vary, making comparison difficult. As a result of all this, modelling *Sclera* is less a descriptive than a creative process. Indeed, all the example strings presented in this sketch are hypothetical, albeit based on linguistic knowledge. As a preface to the next section, the assumptions which guide this 'armchair' approach, as well as the aims of the forthcoming grammar model, are summarised below.

Grammar aims

- Coverage of simplex picto '(pro)nouns', 'verbs', and 'adjectives', bearing in mind their underspecified status.
- Theory of picto valency/subcategorization patterns
- Analysis of missing determiners (Section 3.2.3).
- Integration of complex pictos.
- Ability to parse simple main/matrix clauses and noun phrases

Assumptions

- *Sclera has* a grammar that can be modelled
- The input to the *Sclera* parser consists of lemmas that are morpholgically atomic.
- The elements on the *Sclera* input adhere to subject-verb-object order.
- The input can be resolved to a complete main clause or to a noun phrase.

3.2.2 Setting up with the LinGO Matrix

The starting point for the construction of an implemented grammar of *Sclera* is the LinGO Grammar Matrix customization system (introduced in Section 2.2.3.2). This system, to rephrase its basic function, generates language-specific extensions of a core type hierarchy, viz., the LinGO Matrix, based on a number of (interactively provided) statements about properties of the language being modelled.

For ease of prototyping, however, the language that is *actually* modelled in this step is not *Sclera*, but the other language with which this thesis deals, i.e., Dutch (to which we return in the next chapter). This may seem somewhat confusing at first, particularly at the current level of abstraction, but the idea is simple and bears mentioning sooner rather than later. As the observations in the previous section suggest, *Sclera* can (though not necessarily) be analysed as an SVO language and, as such, shows some amenability to the mechanisms used in analyses of such languages. Dutch is an SVO language, too (at least, at certain levels, which I will say more about later). Thus, one can expect some overlap between *Sclera* and Dutch, especially on the level of their type hierarchies. Since these grammars are developed in parallel anyway, from an engineering standpoint, it makes sense to capitalize on such overlap so that those parts which both grammars have in common need not be defined twice. Therefore, we opt to use the grammar customization step to obtain a grammar of Dutch, from which the grammar of *Sclera* inherits *part* of its type hierarchy. Crudely put, the *Sclera* grammar is an underspecified variant of a model of Dutch (though English would have worked just as well) with a few *Sclera*-specific additions, which are introduced in the following sections. However, this does not mean that the *Sclera* grammar will be indefinitely tethered to the grammar of Dutch: it is a marriage of convenience, not necessity. Eventually, the two will be untangled into two discrete grammars. For reference purposes, (3.16) shows how the *Sclera* grammar's components are related with respect to the names of the associated files.

(3.16)

Exclusive to *Sclera*	**Inherited from Dutch grammar**
`mini-sclera.tdl` (*Sclera*-specific type hierarchy; Developed in sections 3.2.3 and 3.4)	`matrix.tdl` (LinGO Matrix core type hierarchy (± 3800 lines))
`lexicon.tdl` (contains lexical entries for pictograph identifiers)	`mini-dutch.tdl` (type hierarchy of Dutch)
`rules.tdl` (contains phrase rule entries)	`head-types.tdl` (recent addition to Matrix (not discussed); defines disjunctive head types)
`lrules.tdl` (contains lexical rule entries)	`roots.tdl` (list of structures which can serve as start symbol)
`config.tdl` (Initial configuration file used by ACE during compilation; concerns settings)	`labels.tdl` (specifies labels for nodes in syntax trees (forwhen using visual modes))
`mini-sclera-top.tdl` (Second ACE configuration file; points compiler to appropriate files)	`semi.vpm` (Virtual Property Mapping interface (Not introduced in Section 2.2.1.3 on account of its rather technical nature; mediates between 'internal' and 'external' MRS representations; see the DELPH-IN wiki entry at `moin.delph-in.net/RmrsVpm`))

To recapitulate, the result of the customization step is *actually* an initial grammar of Dutch, from which an initial grammar of *Sclera* is derived. Although the two grammars bifurcate slightly as they are extended later on (large parts of either are, in fact, later re-written or replaced with patches from other DELPH-IN grammars), at this stage, the grammar of *Sclera* can be thought of simply as a less specific, i.e., more 'permissive', variant of the grammar of Dutch. The grammar inherits all of the other's type constraints, but leaves these underspecified where necessary, for instance, when a given feature is not relevant.

The primary locus of underspecification is the lexicon. To illustrate this, compare the lexical entry for *hond* ('dog') in the grammar of Dutch (3.17)) to the lexical entry for the picto *dog* in the grammar of sclera (3.18). (Note that the pictograph identifier, specified on the list value of the STEM attribute, is provided as a simple lemma, as discussed in Section 3.1.)

(3.17) Lexical entry for Dutch *hond* ('dog')

```
hond := femmasc-count-sg-noun-lex &
  [ STEM < "hond" >,
    SYNSEM.LKEYS.KEYREL.PRED "_hond_n_rel" ].
```

(3.18) Lexical entry for picto depicting 'dog'

```
dog := common-noun-lex &
  [ STEM < "dog" >,
    SYNSEM.LKEYS.KEYREL.PRED "_dog_n_rel" ].
```

The types with which these lexical entries are identified, viz., *femmasc-count-sg-noun-lex* and *common-noun-lex*, are situated on different levels of the type hierarchy. As the local hierarchy in (3.20) shows, *common-noun-lex* is a supertype of *femmasc-count-sg-noun-lex*. It inherits from *obl-spr-noun-lex* itself, which in turn multiply inherits from several other general types. (The dashed edges between tree nodes indicate where the hierarchy has been purposefully simplified.) Lexical entries of the type *common-noun-lex*, an AVM example of which is given in (3.19), have a HEAD attribute that takes values of the type *noun* (which can be specified for the feature CASE), require one *obligatory* argument[3], which is identified with the value of the SPecifieR

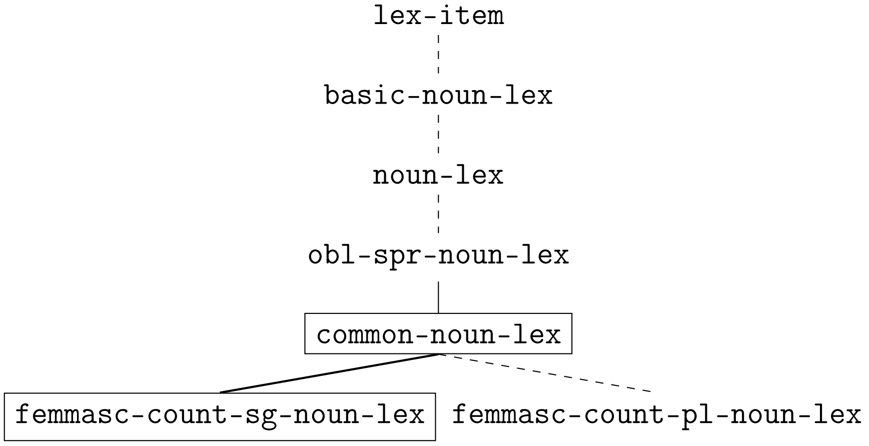
[3] See Section 3.2.3

(SPR) attribute (used to build determiner/quantifier-noun phrases), and, on a semantic level, have additional appropriate features *person*, *number*, and *gender*, which correspond to MRS variables, but *common-noun-lex* does not stipulate any constraints on the values of these MRS variables. As a result, the lexical entry for the pictograph ‘dog’ is appropriately identified with the part-of-speech category ‘noun’, but is left entirely underspecified with respect to the features number, person and (since this varies from language to language) gender. By contrast, the types used in the lexicon of the grammar of Dutch are more specific. For instance, by inheriting from *femmasc-count-sg-noun-lex*, the lexical entry in (3.17) is defined explicitly as being singular, non-neuter (feminine or masculine), and count (requiring an explicit determiner). This is appropriate because these features are relevant to the syntax of Dutch.

(3.19)

$$\begin{bmatrix} \textit{common-noun-lex} & \\ \text{STEM} & \left\langle \text{‘dog’} \right\rangle \\ \text{SYNSEM}\,|\,\text{LOCAL} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{noun} & \\ \text{CASE} & \textit{case} \\ \text{MOD} & \langle\ \rangle \end{bmatrix} \\ \text{VAL} & \begin{bmatrix} \text{SUBJ} & \langle\ \rangle \\ \text{SPR} & \left\langle \boxed{1} \begin{bmatrix} \text{LOCAL}\,|\,\text{CAT}\,|\,\text{HEAD} & \textit{det} \\ \text{OPT} & \text{-} \end{bmatrix} \right\rangle \\ \text{COMPS} & \langle\ \rangle \end{bmatrix} \end{bmatrix} \\ \text{CONT} & \begin{bmatrix} \text{HOOK}\,|\,\text{INDEX}\,|\,\text{PNG} & \begin{bmatrix} \textit{ref-ind} & \\ \text{PER} & \textit{person} \\ \text{NUM} & \textit{number} \\ \text{GEND} & \textit{gender} \end{bmatrix} \\ \text{RELS} & \left\langle \begin{bmatrix} \textit{ep} & \\ \text{PRED} & \text{‘\_dog\_n\_rel’} \end{bmatrix} \right\rangle \\ \text{HCONS} & \langle\ \rangle \end{bmatrix} \end{bmatrix} \\ \text{ARG-ST} & \langle \boxed{1} \rangle \end{bmatrix}$$

(3.20)



‘Verb’ pictographs in the lexicon of the *Sclera* grammar are similarly identified with comparatively general lexical types. As may be intuited partially from the associated type names, the constraints for which entries in the Dutch grammar are specified (including *person*, *number*, and *tense*) are left underspecified in equivalent entries in the grammar of *Sclera*. Once again, an illustration of the (local) type hierarchy, given in (3.23), does much in the way of clarity.

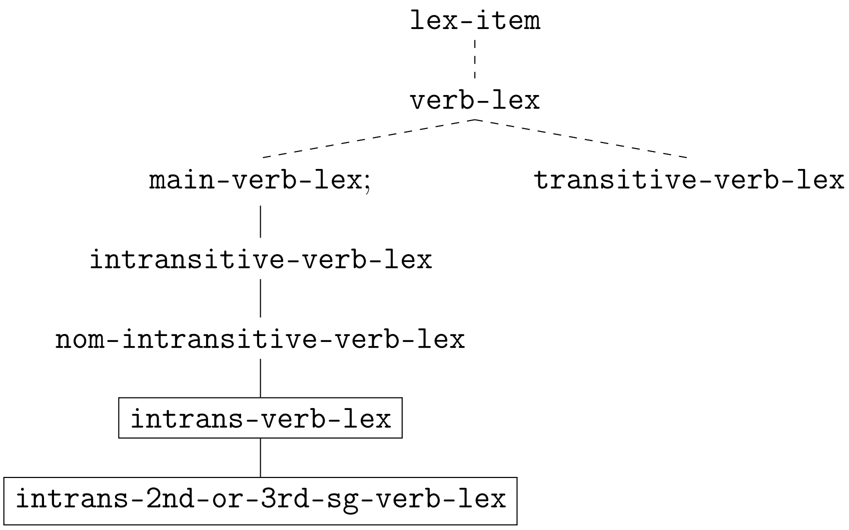
(3.21)

```
slaapt := intrans-2nd-or-3rd-sg-verb-lex &
  [ STEM < "slaapt" >,
    SYNSEM.LKEYS.KEYREL.PRED "_slapen_v_rel" ].
```

(3.22)

```
sleep := intrans-verb-lex &
  [ STEM < "sleep" >,
    SYNSEM.LKEYS.KEYREL.PRED "_sleep_v_rel" ].
```

(3.23)



Turning now to phrase structure rules – at this stage, the grammar of *Sclera* uses the same basic rule types as defined by the grammar of Dutch. The three most important correspond to the 'schemata' of canonical HPSG(Pollard & Sag 1994). Provided as entries in `rules.tdl`, as shown in (3.24), these types instantiate the backbone for the SVO order assumed of *Sclera* strings in the previous section.

(3.24) Top of `rules.tdl`

```
head-comp := head-comp-phrase. ;; (1)
subj-head := subj-head-phrase. ;; (2)
head-spec := head-spec-phrase. ;; (3)
```

Word order is accounted for by combining rule types that impart unique 'headedness' to the daughters of a phrase with rules that constrain the linear position of the head daughter in relation to her non-head sisters. For instance, the *head-complement phrase* rule, in (3.25), combines the constraints of *basic-head-1st-comp-phrase* (a subtype of 'dominance' rule types involving complements) with the constraints belonging to the type *head-initial*, which, as its name suggests, requires that the head daughter come first in the phrase.

Variations on this logic are used in the type definitions of the other two rules, except here the head daughter is required, by *head-final*, to come last, or – as these rules are consistently binary (see Section 2.2.1.4) – second.

```
(3.25)  head-comp-phrase := basic-head-1st-comp-phrase & head-initial.
(3.26)  subj-head-phrase := decl-head-subj-phrase & head-final &
              [ HEAD-DTR.SYNSEM.LOCAL.CAT.VAL.COMPS < > ].
(3.27)  head-spec-phrase := basic-head-spec-phrase & head-final.
```

The result of the customization stage (followed by a fair amount of manual intervention) is a partial model of *Sclera* comprising, among other, type descriptions for simplex 'noun' and 'verb' pictographs and basic grammar rules for phrase construction (which, it ought to be mentioned, also implement the semantic composition principles of HPSG (Sag et al. 1999)) as well as the internal structure of phrases, with respect to both linear precedence and immediate dominance (Pollard & Sag 1994). However, the current model is not entirely ready to be used for parsing yet. The next section aims to fix this.

3.2.3 'Bringing back' determiners

Recall that the function of the grammar of *Sclera* within the *Depicto* system is to 'extract' as much semantic information as possible from the pictographic input, which, by natural language standards, is vastly underspecified. In order to do this, then, the grammar must be able to make hypotheses about semantic structure based on what little clues the input affords. The grammar developed so far does not do this, however; only the immediate surface structure of the input is taken into account. As a result, the absence of determiners in the *Sclera* language causes the grammar to run into trouble. This is because of two reasons. First, on a theoretical level, the LinGO Matrix incorporates the assumption that the referential indices, i.e., 'noun' predications, of a well-formed MRS must always be outscoped by some quantifier (Copestake et al. 2005). Second, in the grammar of Dutch, from which the grammar of *Sclera* derives its types, this assumption is worked out by an interplay of constraints which have the effect of requiring that common nouns select for an obligatory 'specifier' (quantifier/determiner). This is what is specified by the negative value of the OPT attribute in the AVM description of the type *common-noun-lex* in (3.19), the relevant portion of which is repeated here in (3.28). Since determiners do not occur on the input, the parser fails when passed a *Sclera* string containing pictographs of the 'common noun' type.

(3.28)
$$\begin{bmatrix} \textit{common-noun-lex} & \\ \mathrm{SYNSEM}\,|\,\mathrm{LOCAL}\,|\,\mathrm{CAT}\,|\,\mathrm{CAL}\,|\,\mathrm{SPR} & \left\langle \begin{bmatrix} \mathrm{LOCAL}\,|\,\mathrm{CAT}\,|\,\mathrm{HEAD} & \textit{det} \\ \mathrm{OPT} & \text{-} \end{bmatrix} \right\rangle \end{bmatrix}$$

However, not all types of nouns select an explicit determiner, as (3.29) and (3.30) show for mass nouns and pronouns, respectively. These types take either no specifier or take one but mark it as optional.

(3.29) **Water** has three forms.

(3.30) **You** should tell **me** about **them**.

When occurring without a specifier, as in the examples above, these noun types form so-called *bare noun phrases*. However, semantically, their referential indices are still scoped by an 'exist' quantifier, as represented semi-formally in (3.31).

(3.31) $exists(x, noun\_relation(x))$

In order to achieve this effect, the LinGO Matrix provides a definition of a unary phrase rule type that takes a single nominal head daughter with an optional specifier constraint and *adds* a quantification relation to the list of *elementary predications* provided by the head daughter. The semantic contribution of the rule is specified on the attribute C-CONT (*Construction CONTent*). As the identity statements (prefixed by a '#') in the *TDL* description in (3.32) show, the C-CONT part of the rule identifies the daughter's *index* with the index argument (ARG0) of *quant-relation.* At the same time time, via identity with an attribute of the handle constraint relation in HCONS, the top handle (LTOP) of the daughter is placed within the quantifier's scope.

(3.32)

```
basic-bare-np-phrase := head-only &
  [ SYNSEM.LOCAL.CAT.VAL [ SPR < >,
               SUBJ < >,
               COMPS < >,
               SPEC < > ],
    HEAD-DTR.SYNSEM.LOCAL [ CAT [ HEAD noun,
                                  VAL [ SPR < [ LOCAL.CAT.HEAD det,
                                                OPT + ] >,
                                        SUBJ < >,
                                        COMPS < > ] ],
                            CONT.HOOK [ INDEX #index,
                                        LTOP #larg ] ],
    C-CONT [ RELS <! quant-relation &
                      [ ARG0 #index,
                        RSTR #harg ] !>,
             HCONS <! qeq &
                     [ HARG #harg,
                       LARG #larg ] !>,
             ICONS <! !>,
             HOOK [ INDEX #index ] ] ].
```

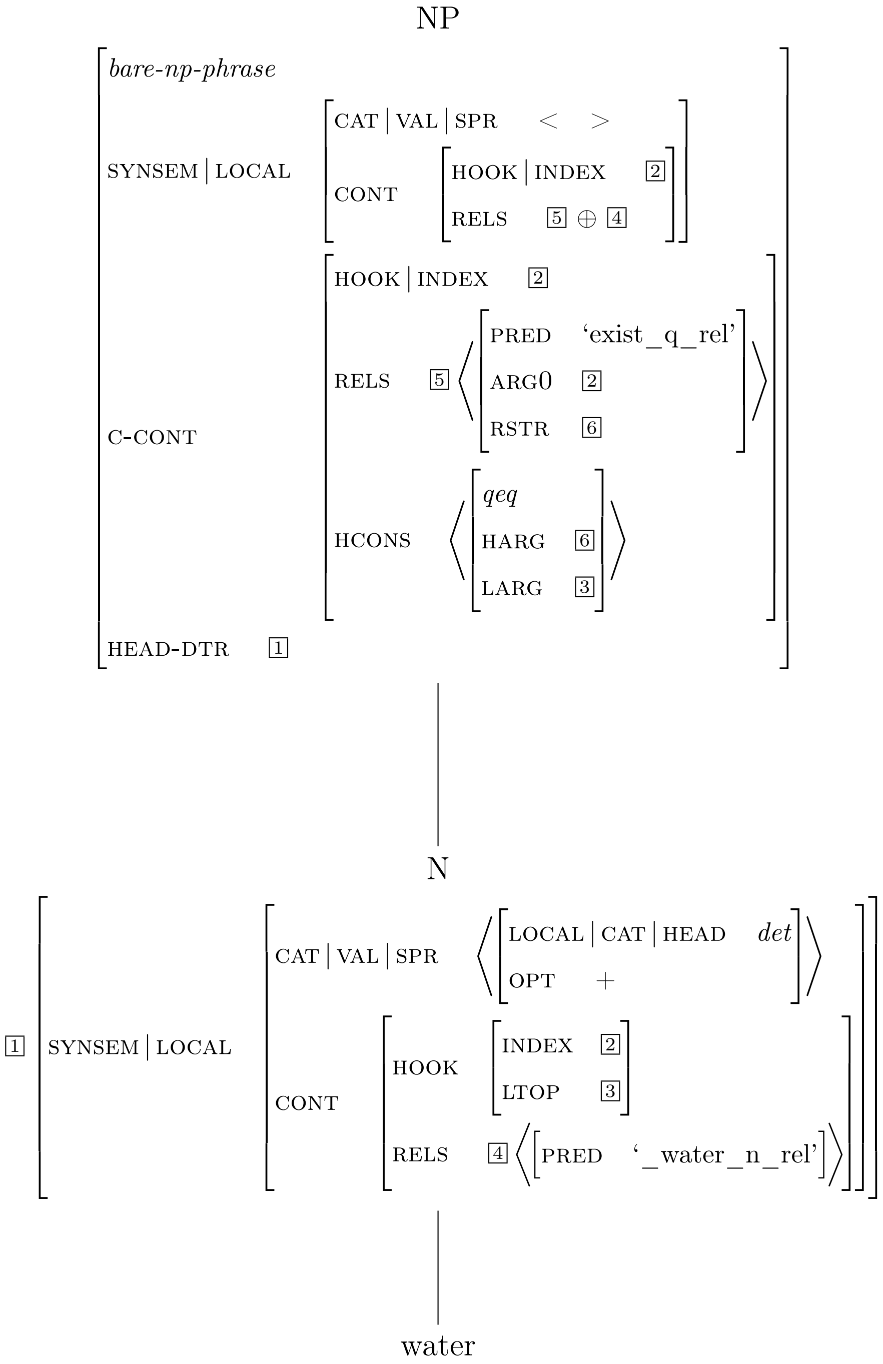
The Matrix definition of the rule leaves the specification of the quantifier's PRED value (i.e., relation name) to the precise grammar in which it is used. In the grammar of *Sclera* this becomes *'exist_ q_ rel'*, as shown in (3.33).

(3.33)

```
bare-np-phrase := basic-bare-np-phrase &
  [ C-CONT.RELS <! [ PRED 'exist_q_rel' ]  !> ].
```

Example (3.34) shows the *basic-np-phrase* rule being used to parse the mass noun *water* (as it is used (3.29)). Notice that the mother node, i.e., the value of the top-level feature path SYNSEM|LOCAL, is (a) a saturated noun phrase (NP), which makes the resulting structure compatible with the head-complement and head-subject rules; and (b) that its relations list (under CONT) is the append of those of C-CONT and HEAD-DTR. In short, the semantic content of the mother node, or, as it were, the 'left side' of the phrase rule, contains *more* than what is found on the actual input.

(3.34)

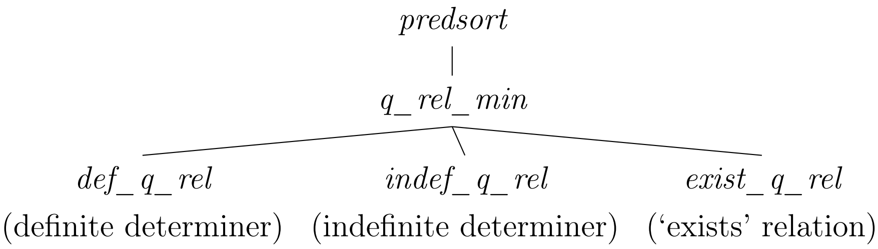


This notion that a phrase rule can itself *contribute* semantic information (or simply modify it) forms the basis for many of the analyses which enable the *Sclera* grammar to make the best of its semantically impoverished input, including the grammar's treatment of missing determiners. (In fact, if the foregoing explanation is at all clear, the reader may by now already have some idea as to how this might work.) All that is needed to 'bring back' determiners is a slightly modified version of the *bare-np-phrase* rule which unifies with common nouns, i.e., noun entries with a *non-optional* specifier constraint, and places these nouns in a similar scoping relation. The *bare-np-phrase* rule, therefore, is copied and modified accordingly to obtain the *TDL* description in (3.35). Notice that, instead of 'exist_q_rel', this rule contributes a new quantifier relation, 'q_rel_min'. Though specified as a string in the *Sclera* grammar, this quantifier relation is later converted to a type, which, as the next chapter will explain in more detail, is situated at the top of a local hierarchy of different determiner and quantifier relations. Slightly ahead of time (the hierarchy is the domain of the target grammar), this hierarchy of determiner types is shown in (3.36)). By virtue of the *sclera-basic-insert-det* rule, the *Sclera* grammar can now parse strings containing common nouns without requiring that these be accompanied by (non-existent) determiners.

(3.35)
```
sclera-basic-insert-det-rule := head-only &
  [ SYNSEM.LOCAL.CAT.VAL [ SPR < >,
                           SUBJ < >,
                           COMPS < >,
                           SPEC < > ],
    HEAD-DTR.SYNSEM.LOCAL [ CAT [ HEAD noun,
                                  VAL [ SPR < [ LOCAL.CAT.HEAD det,
                                                OPT - ] >,
                                        SUBJ < >,
                                        COMPS < > ] ],
                            CONT.HOOK [ INDEX #index,
                                        LTOP #larg ] ],
    C-CONT [ RELS <! quant-relation &
                     [ PRED "q_rel_min",
                       ARG0 #index,
                       RSTR #harg ] !>,
             HCONS <! qeq &
```

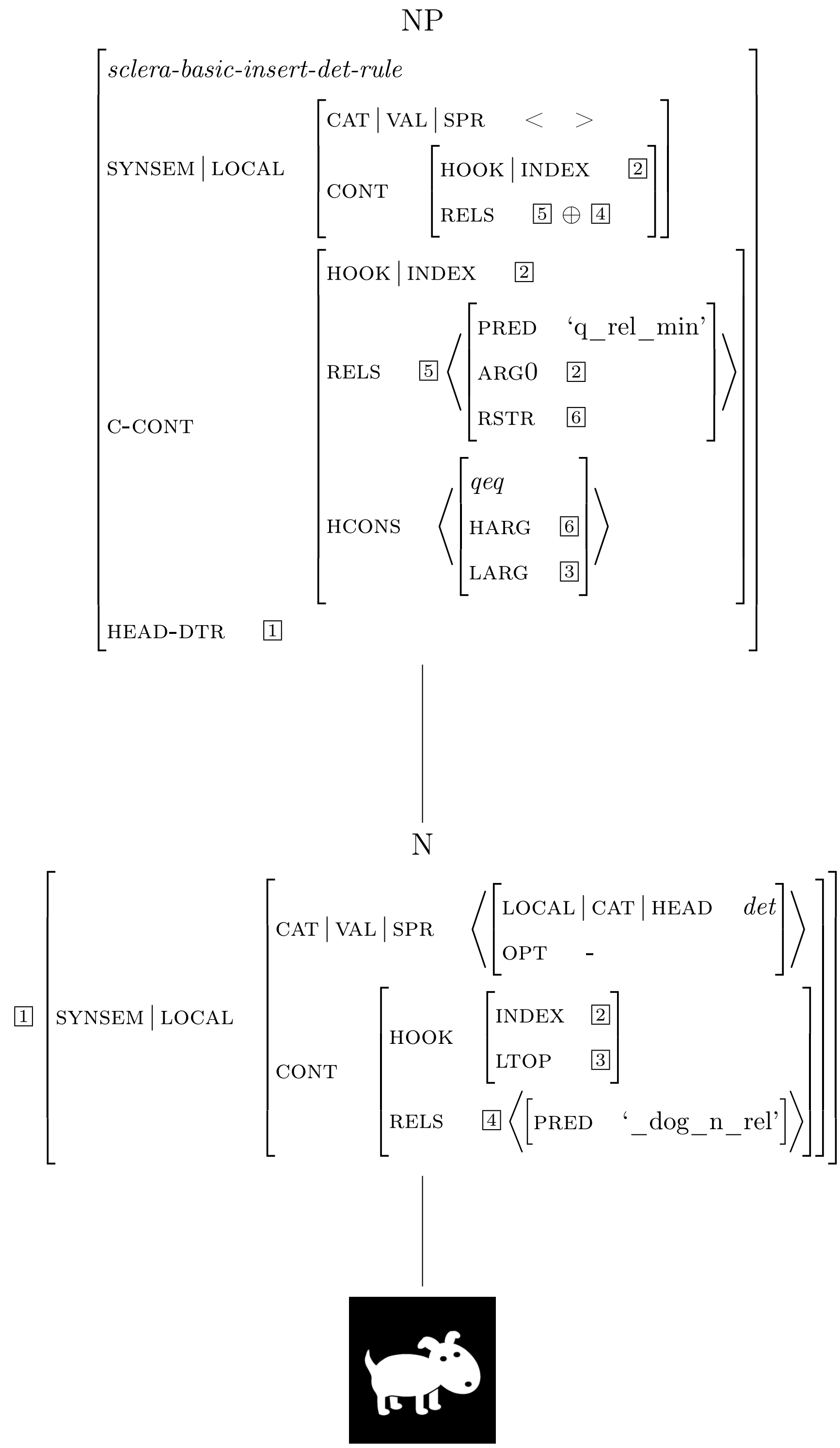
```
         [ HARG #harg,
           LARG #larg ] !>,
ICONS <! !>,
HOOK [ INDEX #index,
       LTOP #larg ] ] ].
```

(3.36)



The parse of *Sclera* 'dog', represented as a tree diagram in (3.37), is clearly similar to that licensed by the rule upon which it is based. In fact, the differences are hard to spot. Yet the importance of the *sclera-insert-det-rule* for the grammar of *Sclera* cannot be overstated.

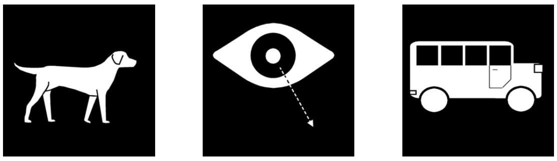
(3.37)



With the *sclera-insert-det-rule* in place, the grammar can now parse simple

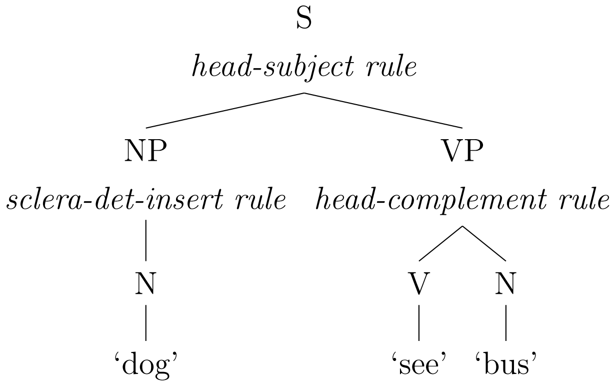
Sclera strings of the kind shown in (3.8), repeated for convenience here by (3.38). The tree diagram representation of the parse is given in (3.39).

(3.38)



dog *see* *bus*

(3.39)



3.3 Understanding the MRS output

When using the ACE processor (Packard 2015), the output of parsing is a *plain-text* MRS representation. An example of such an MRS representation is given in (3.40), which shows the fully semantic structure to which the string in (3.39) resolves, i.e., the semantic information found at the root node 'S'.

(3.40) Output MRS representation of 'dog see bus'

```
SENT: dog see bus
[ LTOP: h0
  INDEX: e2 [ e SF: prop-or-ques E.TENSE: tense E.ASPECT: aspect E.MOOD: mood ]
  RELS: < [ "_dog_n_rel"<-1:-1> LBL: h4 ARG0: x3 [ x PNG.PER: person
                                                     PNG.NUM: number
                                                     PNG.GEND: gender ] ]
```

```
           [ "q_rel_min"<-1:-1> LBL: h5 ARG0: x3 RSTR: h6 BODY: h7 ]
           [ "_see_v_rel"<-1:-1> LBL: h1 ARG0: e2 ARG1: x3 ARG2: x8 [ x PNG.PER: person
                                                                        PNG.NUM: number
                                                                        PNG.GEND: gender ] ]
           [ "_bus_n_rel"<-1:-1> LBL: h9 ARG0: x8 ]
           [ "q_rel_min"<-1:-1> LBL: h10 ARG0: x8 RSTR: h11 BODY: h12 ] >
HCONS: < h0 qeq h1 h6 qeq h4 h11 qeq h9 > ]
```

Although the representation in the example has been slightly simplified (and beautified with extra indentation), it may be observed that its general format approximates that of the AVM diagram given in Section 2.2.2. However, there are also some differences. The attributes LTOP and INDEX are taken out of the HOOK feature and promoted to the top level of the representation, and, presumably for legibility, the *qeq* constraints are not presented as individual feature structures, nor are they separated by a comma. More salient, however, is the appearance of attribute-value pairs such as `E.TENSE: tense` and `PNG.GEND: gender` after referential variables (e.g., `x8`) and event variables (e.g., `e2`). These convey additional semantic properties associated with the values of the attributes INDEX or ARG0. In example (3.40), all but one are left underspecified: the value of `SF` (third line from the top) conveys that the illocutionary force of the parsed string is either declarative or interrogative. When the parsed string contains pronouns, however, the properties of the referential index/variable are more specific, as the fragment in (3.41) shows. The referential (‘`ARG0`’) argument of the predicate ‘_pronoun_n_rel’ has the property of being first person singular (*I* in English), but leaves the value of the `GEND` property underspecified, since it is gender-neutral.

(3.41) Fragment of MRS representation of ‘I walk’

```
RELS: < [ "_pronoun_n_rel"<-1:-1> LBL: h4 ARG0: x3 [ x PNG.PER: 1st
                                                       PNG.NUM: singular
                                                       PNG.GEND: gender ] ]
```

3.4 Extending the *Sclera* grammar

This section presents three case studies which extend the grammar of *Sclera* developed in the previous section and, in so doing, offer some indication, albeit tentative, of the practicability of a rule-based approach to modelling *Sclera* as a language in its own right. Section 3.4.1 and Section 3.4.2 introduce two new phrase rules that demonstrate how the semantic accuracy of the grammar can be enhanced so that it can distinguish, on the one hand, between past and 'other' tense readings of the input and, on the other hand, between declarative and illocutionary readings. Abstractly, both phrase rules function by identifying a specific non-head daughter 'particle' and modifying the semantic content of the head daughter to match some feature contributed by this 'particle'. (Such particles can also be thought of as 'indicators' or 'operators'. In part-of-speech terms, they correspond to adverbs, though there are some differences in overall function.) Moving on, the third case study (Section 3.4.3) details a first attempt at extending the grammar's coverage to conceptually complex pictographs. I say 'first attempt' because (a) only a selection of complex pictograph types has so far been incorporated; (b) the local type hierarchy which they deploy is fairly rudimentary; and (c) as a result, the division of labor between the lexicon and the type hierarchy is currently biased toward the lexicon. Nevertheless, the discussion in this section demonstrates that complex pictographs *can* in principle be incorporated, provided that a handful of new rules and two new additions to the feature geometry are introduced.

3.4.1 Temporal disambiguation

Sclera 'verbs' are underspecified for tense. As a result, the clause structures which they project allow as many readings as there are tenses available in the semantic universe. Thus, supposing a minimal theory of tense types, the string in (3.38) can correspond, among other, to the three different readings shown in (3.42), all other sources of underspecification kept equal.

(3.42) ‘dog see bus’ ⇔

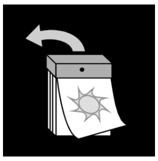
a. ‘The dog *sees* the bus.’ (Present tense domain)

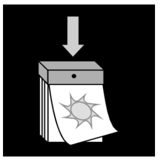
b. ‘The dog *saw* the bus.’ (Past tense domain)

c. ‘The dog *will* see the bus.’ (Future tense domain)

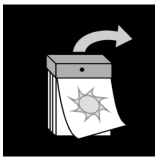
Similar to the ‘re-introduced’ determiner relation in Section 3.2.3, which, as we saw there, is actually a generalization over several concrete kinds of determiners, the *intended* tense of a simple string such as ‘dog see bus’ is grammatically undecidable, and is therefore best left underspecified. It is arguably a limitation of the *Sclera* set that it does not contain pictographs which explicitly specify the intended tense. However, the set does contain a number of pictographs that depict temporal concepts and which can conceivably function as adjuncts of time in a *Sclera* clause. A selection is given by (3.43).

(3.43)


‘yesterday’


‘today’

‘now’


‘tomorrow’

In natural language, such time adjuncts are generally appropriate to a single specific temporal domain. As (3.44) illustrates for ‘yesterday’, the effect in *Sclera* is that they restrict the felicitousness of individual tense readings.

(3.44) ‘dog see bus yesterday’ ⇔

a. ‘The dog *saw* the bus yesterday.’

b. * ‘The dog *sees* the bus yesterday.’

c. * ‘The dog *will* see the bius yesterday.’

From a procedural perspective, these adjuncts can be understood as *modifying* the tense of the verb phrase with which they combine so that its tense is constrained to a specific type. This behavior is modelled by the introduction of a new phrase rule type called ***temp-adverb-verb-phrase*** (the adjuncts of time in (3.43) are analysed as simple adverbs here). As its $\mathcal{TDL}$ description shows, this rule belongs to the 'head-modifier' category (Pollard & Sag 1994). Essentially, it identifies the TENSE value of the index of its non-head daughter, which must be an adverb[4], with the TENSE value of the index of C-CONT. The index of this last feature is passed up to the mother of the rule through constraints inherited from its supertypes. While C-CONT shares its TENSE value with the adverbial daughter, all features found under C-CONT's HOOK attribute are shared with those of the head daughter, including the TENSE attribute. By virtue of unification, however, the compatible yet more specific value of the non-head daughter's TENSE feature 'wins', as it were, over the more general value of the head-daughter's TENSE feature. For the time being, finally, the rule requires the head daughter to be a finite clause, specified by `[ CAT.MC + ]`. This goes against conventional wisdom, which analyzes adjuncts as modifying the verb phrase, not the clause as a whole, and should probably be revised later on.

(3.45)

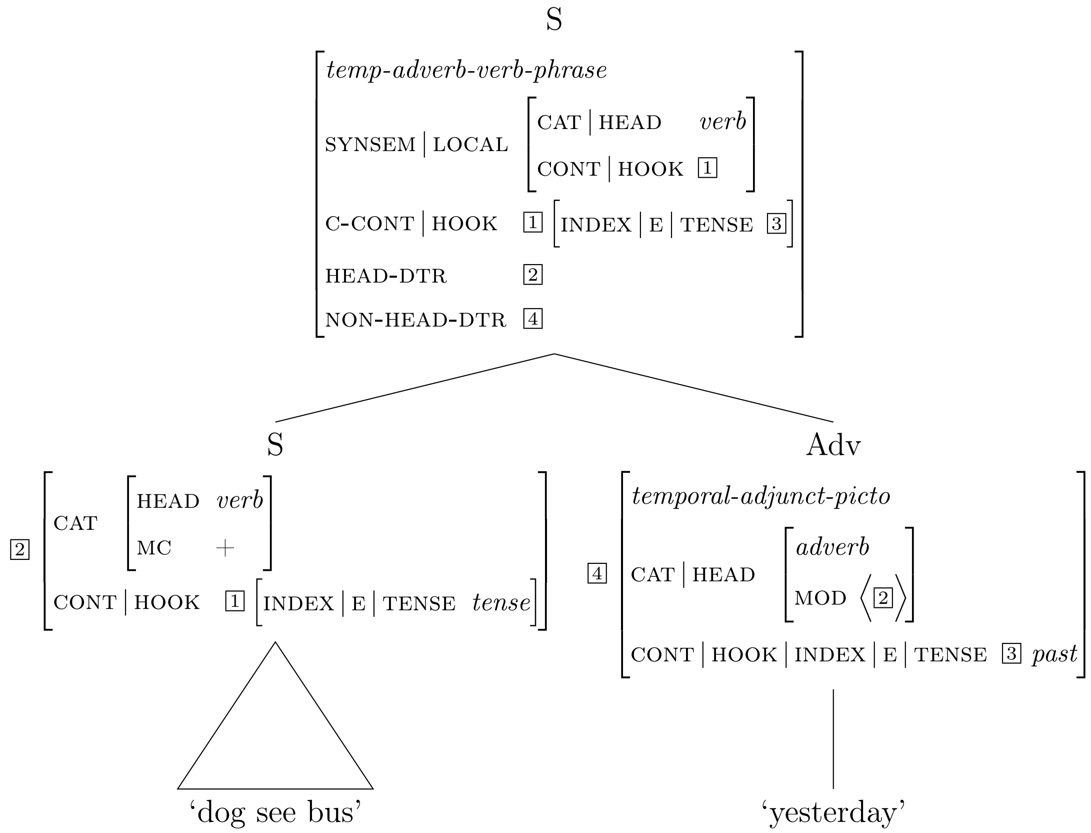
```
temp-adverb-verb-phrase := basic-head-mod-phrase-simple &
                               head-initial & isect-mod-phrase &
 [ C-CONT.HOOK #hook & [ INDEX.E.TENSE #tense ],
   NON-HEAD-DTR.SYNSEM.LOCAL [ CAT.HEAD adv,
                               CONT.HOOK.INDEX.E.TENSE #tense ],
   HEAD-DTR.SYNSEM.LOCAL [ CONT.HOOK #hook,
                           CAT.MC + ] ].
```

As the tree representation of the parse of the string 'dog see bus yesterday' in (3.46) shows, the effect of the *temp-adverb-verb-phrase* rule is that the adjunct of time 'imparts' its TENSE constraint to the clause/verb phrase which it

[4] In HPSG, adverbs are treated semantically as 'events' (Pollard & Sag 1994)

modifies.

(3.46)



Just like the rule responsible for the 're-introduction' of determiners, the *temp-adverb-verb-phrase* rule offers a means of enriching the semantic structure that is extracted from the input string during parsing. Later on in the translation process (in the generation stage, to be specific), the resulting increase in specificity will amount to increased accuracy. Itself still a proof of concept, the rule is not as 'strict' as it should be: it is not constrained to time adjuncts *only.* However, since the grammar currently models only one adverb and this adverb is the time adjunct 'yesterday', this does not pose any problems at the moment. To be clear, however, adding the remaining time adjunct pictos listed in (3.43) to the grammar would be a relatively simple task.

Finally, the *temp-adverb-verb-phrase* rule exhibits two limitations that will need to be overcome if it is to be useful in a real-life context. The first is that the head daughter is required to come *before* the modifying daughter and that this head daughter needs to be a finite clause. Ignoring the second issue, the restriction on word order prevents even simple variations on the string in (3.44) from being accepted by the grammar: e.g., 'yesterday dog see bus'. The second main limitation is that the rule incorporates no 'claims' about aspect. Thus, 'dog see bus now', though appropriately interpreted in the present tense, allows both the continuous reading '? the dog is seeing the bus now' and the more bounded, simple present reading 'the dog sees the bus now'. The decidedly odd character of the continuous reading is difficult to dismiss, though it is possible that this is something that might need to be assigned to a different component of the grammar, one that deals with, say, pragmatics, for instance.

3.4.2 Triggering the interrogative mood

As shown by the MRS output of parsing 'dog see bus' in (3.40), the relevant portion of which is repeated below in (3.47), *Sclera* clauses are by default also underspecified for illocutionary force, or *mood.*

(3.47) `... INDEX: e2 [ e SF: prop-or-ques ...`

The value of the SF (*Sentence Force*) property of the event variable, viz., *prop-or-ques*, indicates that the clause may be either a proposition or a question, these two options corresponding to the declarative and interrogative mood, respectively. As a result, the clause 'dog see bus' allows both readings in (3.48) – again, all other sources of underspecification kept equal.

(3.48) 'dog see bus' ⇔

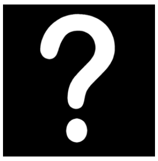
a. 'The dog sees the bus.' (Declarative mood)

b. 'Does the dog see the bus?' (Interrogative mood)

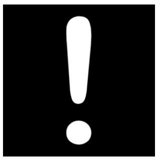
In a real use context, such unrestrained alternation between, interactively, two very different readings would probably cause some confusion. Therefore, the default reading is constrained to the declarative mood. (In the current version of the system, this constraint is contributed by the target grammar, so it is discussed later.) Nevertheless, there is no reason to assume that a hypothetical user might not at some point want to use the system to ask a question. In fact, despite the comparatively higher frequency of the declarative mood, this is almost inevitable. The grammar should provide some means, therefore, of 'sensing' when the intended reading concerns the interrogative mood.

The *Sclera* set contains a small set of pictographs which depict literal punctuation marks. Two such symbols are given in (3.49). (The exclamation mark is provided on account of its possible association with the imperative mood.)

(3.49)



'question'



'exclamation'

Assuming that users can be taught to associate a 'question mark' symbol with the interrogative mood, using it as a sort of explicit mood marker, the string in (3.50) only has one appropriate reading, namely, as having interrogative illocutionary force.

(3.50) 'dog see bus question' ⇔
a. 'Does the see the bus?'
b. * 'The dog sees the bus.'

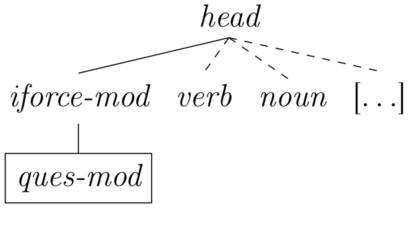
Thus, just as the time adjunct 'yesterday' in Section 3.4.1 *triggers* the past tense, so the illocution operator 'question' can be interpreted as triggering the interrogative mood. This 'triggering' behavior is modelled by a new rule which, based on the same basic mechanism as the *temp-adverb-verb-phrase* rule in

Section 3.4.1, identifies the illocutionary force value (found on the feature path SYNSEM|LOCAL|CONT|HOOK|INDEX|SF|) of a verbal head daughter with the more specific illocutionary force value of the operator ‘question’, which serves as the non-head daughter.

For the current purposes, the illocution operator ‘question’ could be treated as an adverb. However, since its function does not really have an equivalent in natural language, where the question mark is generally categorized under the orthographic domain, it is added to the grammar as a new lexical type called *question-picto*. As the partial AVM description in (3.51) shows, this type modifies a single saturated verb-headed feature structure and bears an SF (illocutionary force) feature with the value *ques* (a subtype of *prop-or-ques*). Note, however, that its RELS list is empty: it does not contribute a predicate relation. Finally, the type bears a newly posited HEAD value *ques-mod*, which itself is introduced to the type hierarchy as a subtype of a new head type *iforce-mod*, as (3.52) shows.

(3.51)
$$\begin{bmatrix} \textit{question-picto} & \\ \text{SYNSEM}\,|\,\text{LOCAL} & \begin{bmatrix} \text{CAT}\,|\,\text{HEAD} & \begin{bmatrix} \textit{ques-mod} & \\ \text{MOD} & \left\langle \begin{bmatrix} \text{LOCAL}\,|\,\text{CAT} & \begin{bmatrix} \text{HEAD} & \textit{verb} \\ \text{VAL} & \begin{bmatrix} \text{SUBJ} & < > \\ \text{COMPS} & < > \end{bmatrix} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix} \\ \text{CONT} & \begin{bmatrix} \text{HOOK}\,|\,\text{INDEX}\,|\,\text{SF} & \textit{ques} \\ \text{RELS} & < > \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

(3.52)

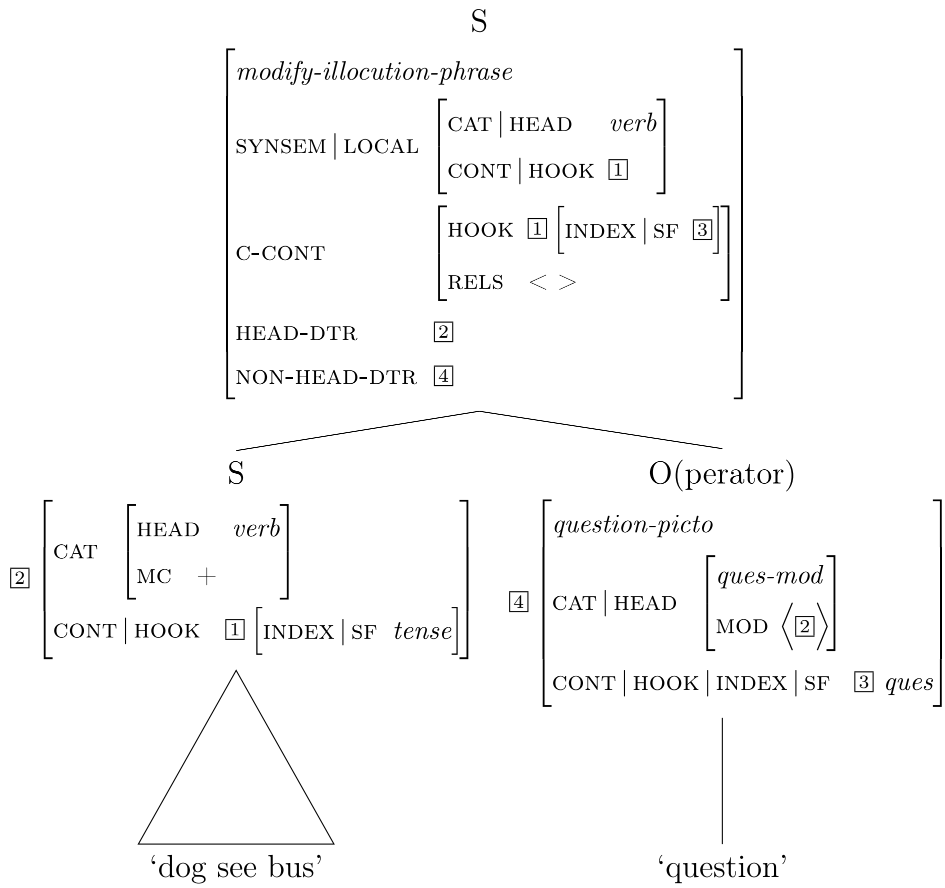


The rule type that is introduced to enables this new lexical type to combine with the structure on its MOD list is called the ***modify-illocution-phrase*** rule. Its 𝒯𝒟ℒ description is given in (3.53). Similar to the *temp-adverb-verb-phrase* type in Section 3.4.1, it is essentially a head-modifier rule (Pollard & Sag 1994) that stipulates a structure sharing constraint between a property of the index of the mother node and the same property of the index of the non-head daughter. Its definition differs from that of the *temp-adverb-verb-phrase* type in that it states no explicit identity relation between the HOOK path of the head daughter and the HOOK path of the mother, and that it explicitly specifies that the RELS under C-CONT is empty. The first difference is due to the fact that the HOOK constraint is already covered by constraints inherited from the *head-compositional* phrase type. The empty RELS list is a feature which ensures that the rule type is compatible with the mechanisms of semantic composition defined in the Matrix type hierarchy.

(3.53)
```
modify-illocution-phrase := basic-head-mod-phrase-simple &
                                      head-initial & head-compositional &
   [ C-CONT [ HOOK.INDEX.SF #illocution,
              RELS <! !> ],
     HEAD-DTR.SYNSEM.LOCAL [ CAT.MC + ],
     NON-HEAD-DTR.SYNSEM.LOCAL [ CAT.HEAD ques-mod,
                                 CONT.HOOK.INDEX.SF #illocution ] ].
```

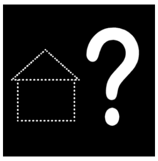
An example of how the *modify-illocution-phrase* rule is used to analyse the string ‘dog see bus question’ is given in (3.54).

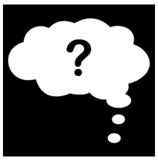
(3.54)

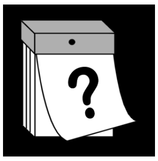


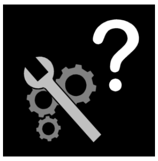
Like the *temp-adverb-verb-phrase* rule, this new addition represents a step in the direction of modelling *Sclera* as a system with its own rules. Just as, in natural language, a rising tone contour generally signals a question (in non-tone languages, at least) the 'question' pictograph serves as a contextual marker of the interrogative mood and, as such, is treated as a part of the grammar. The significance for *Depicto* as an assisitve writing tool is an overall increase in expressivity – provided that users are able to recognize the interactional contribution of the 'question' pictograph. As with much of the work discussed in this chapter, the purpose of the rule presented here is primarily to offer an early indication of the practicability of a rule-based approach to modelling *Sclera.* Thus, although it works fine in its current function, the *modify-illocution-phrase* rule is still in an experimental stage. As a result, there are some limitations. In the first place, it only supports *polar* interrogatives, i.e., yes-no questions. *Wh*-questions, which could be formulated using the *Sclera* symbols in (3.55) (these symbols each have a number of variants), should at some point be covered by the model, although, admittedly, such questions types probably warrant a different rule altogether.

(3.55)


'where'


'why'


'when'

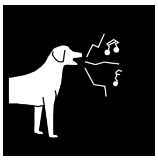

'how'

Another limitation is that, like the *temp-adverb-verb-phrase* rule, the non-head daughter 'question' operator is required to come *after* the head of the phrase. This will eventually be fixed by adding rule variations that invert this order.

3.4.3 Semantically complex pictos

Recall from the inventory of *Sclera* (Section 2.1.2) that a large portion of the symbol set comprises pictographs which depict not one, but several concepts simultaneously. In natural language terms, and in contrast with the 'simplex' pictographs with which we have so far dealt, these semantic compounds correspond to syntactic phrases of varying saturation and with internal structure. For example, as shown by (3.56), the complex picto 'dog+bark' (as it is identified by the grammar) depicts both a referent 'dog' and an event of 'barking', whereby, thematically, 'dog' is to be interpreted as the agent in, or, grammatically, the subject of the 'barking' event. Structurally, the pictograph corresponds to a complete clause, which can be translated, inter alia, as 'The dog barks'.

(3.56)

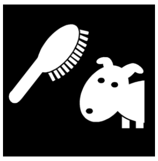


‘dog’, ‘bark’

‘The dog barks.’

Such ready-made clause pictographs are by no means rare within the *Sclera* set. A second, related class of pictographs forego the ‘subject’ argument, but bundles one or more ‘complement’ arguments, such as ‘brush+dog’ in (3.57) or ‘go+school’ (which we have already encountered in (3.12)). These pictographs constitute verb phrases with an open ‘subject’ slot and sometimes an open ‘complement’ slot, such as ‘give+present’, which, as shown in (3.13) above, can arguably take a second complement, namely an indirect object.

(3.57)



‘brush’, ‘dog”

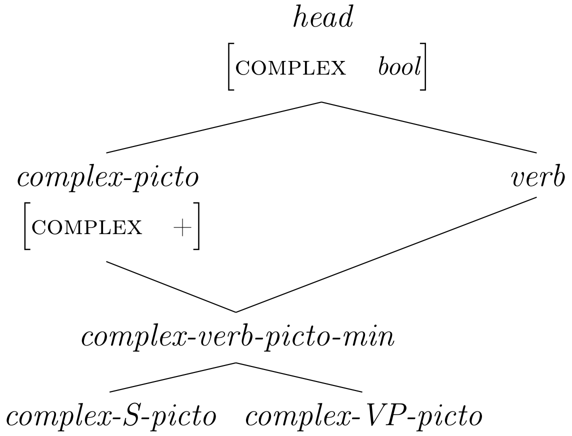
Focusing on (a subset of) the class of complex pictographs that have a semantically process-oriented (‘verbal’) head[5], we will now see how the grammar is adapted to include this new class of pictographs.

First, since complex pictographs do not belong *entirely* to any traditional part-of-speech category, a new head type called *complex-picto* is added to the type hierarchy. (Head types form the value of the attribute HEAD and serve as an important entry condition to many grammar rules.) To the most general head type, viz., *head*, a new appropriate feature is added (COMPLEX) that, by means of a boolean value, specifies whether a head type is complex (in which case its

[5] Complex pictographs with ‘nominal’ heads (e.g. ‘nominal compounds’) constitute a comparatively small portion of the symbol set, although they too will eventually need to be incorporated into the grammar

HEAD value is positive) or simplex (in which case this value is negative). The general head type *head* leaves this value underspecified, so that, by default, all head types can be either simplex or complex. However, in a different area of the hierarchy, 'simplex' lexical types are individually constrained as `COMPLEX - .` Most grammar rules, by contrast, are underspecified in this respect. Glossing over the technicalities, the net effect later on is that, where appropriate, complex pictograph types are able to unify with the daughter slots of pre-existing (simplex) grammar rules, while simplex pictographs are blocked from unifying with the daughter slots of rules designed specifically for complex pictographs. Back to the type hierarchy itself, in order for the grammar to be able to distinguish between 'verb'-headed complexes and, say, 'noun'-complexes, the type *complex-picto* is defined as a supertype of *complex-verb-picto-min*, which additionally inherits from the pre-existing head type *verb*, so as to reflect the fact that this specific class of complex pictographs crosscuts with the verbal category, and, thus, to enable complex pictographs of this type to occur (under certain conditions) as the daughter of pre-existing *verb*-constrained grammar rules. The type *complex-verb-picto-min* takes in turn the subtypes *complex-S-picto* and *complex-VP-picto*, which correspond, respectively, to the example pictographs in (3.56) and (3.57). These to the type hierarchy are visualized in (3.58).

(3.58)



The next step involves defining new lexical types for each kind of complex pictograph. In the object-oriented spirit of HPSG, an attempt is made to factor shared constraints out into more general supertypes. Thus, all complex pictographs inherit their constraints from the general type *complex-picto-lex*, which itself is a subtype of the higher-level Matrix type *lex-item*. In its current form, the main contribution of *complex-picto-lex* is the stipulation that the value of HEAD is of the type *complex-picto*. The $\mathcal{TDL}$ definition is shown by (3.59). (The HOOK value *sclera-hook* can safely be ignored for the moment.)

(3.59)
```
complex-picto-lex := lex-item &
  [ SYNSEM.LOCAL [ CONT.HOOK sclera-hook,
                   CAT [ HEAD complex-picto ] ] ].
```

All lexical types for verb-headed complex pictographs are subtypes of the intermediate type *main-complex-v-picto-lex*, which inherits from *complex-picto-lex* and is specified for an empty SPecifieR (SPR) and an empty COMPlementS (COMPS) list. Note that, while appropriate in most cases, the empty COMPS list is problematic for pictographs such as 'give+present', which, as discussed above, may take an additional complement not included in the conceptual content of the pictograph itself. (It may be said that this one of several indicators of the experimental status of the current treatment of complex pictographs presented here.)

(3.60)
```
main-complex-v-picto-lex := complex-picto-lex &
  [ SYNSEM.LOCAL [ CAT [ VAL  [ SPR < >,
                                COMPS < > ] ] ] ].
```

Pictographs like 'brush+dog', 'go+school', and 'give+present', i.e., single-pictograph verb phrases, are modelled by the type *complex-vp-picto-lex*. Defined as in (3.61), the constraints associated with this type achieve two effects. First, the value of the type's SUBJect feature is identified with the first (and only) item on the type's Argument Structure list (ARG-ST) and, thus, is defined as a saturated nominal sign structure (i.e., a noun phrase). Second, the first (but *not*, as we shall see, only) elementary predication on the type's

RELS list is defined for multiple re-entrancies with the type's HOOK-features. The value of ARG0, an *event*, is identified with the INDEX path; the handle (LBL) of the predication is identified with the top handle (LTOP) of the type; and the value of ARG1 (typically the referent of the 'subject' of the *event* relation) is identified with the path XARG, which is a DELPH-IN shorthand that, as the definition shows is equivalent to the index of the first sign on the argument structure list.

```
(3.61)  complex-vp-picto-lex := main-complex-v-picto-lex &
         [ SYNSEM.LOCAL [ CAT  [ HEAD complex-VP-picto,
                                 VAL.SUBJ < #subject > ],
                          CONT [ HOOK [ LTOP #lbl_head,
                                        INDEX #event,
                                        XARG  #xarg-subj ], ; optional
                                 RELS.LIST.FIRST [ LBL #lbl_head,
                                                   ARG0 event & #event,
                                                   ARG1 #xarg-subj ] ] ],
           ARG-ST.FIRST #subject &
                        [ LOCAL [ CAT [ HEAD noun,
                                        VAL [ SPR < >,
                                              COMPS < > ] ],
                                  CONT.HOOK.INDEX #xarg-subj ] ] ].
```

The function of these identity constraints becomes clear when one considers examples (3.62) and (3.63), which show the descriptions of the lexical entries for 'brush+dog' and 'go+school'. Both entries contribute *all* the relations associated with the complex pictograph, including the determiner relation 'q_rel_min' in (3.62) and the existential quantifier relation 'exist_q_rel' in (3.63), as well as the associated scope constraints (under HCONS). The first item on the list of relations corresponds consistently to the verbal head of the conceptual complex, contributing the predicate '_brush_v_rel' in (3.62) and '_go_v_rel' in (3.63). Through inheritance of the second set of identity constraints in (3.61), these lexical entries are able to properly identify this verbal predication as the head of the structure. Semantically, this guarantees that entries of the type *complex-vp-picto-lex* are treated as standard verb phrases as they propagated up the parse tree.

(3.62)
```
brush_dog := complex-vp-picto-lex &
 [ STEM  < "brush_dog" >,
   SYNSEM.LOCAL.CONT [ HOOK.SCLERA.N1 #N1,
                       RELS <! [ PRED "_brush_v_rel",
                                 ARG2 #object-handle ],
                               #N1 & [ PRED "_dog_n_rel",
                                       LBL  #noun,
                                       ARG0 #object-handle ],
                               [ PRED "q_rel_min",
                                 ARG0 #object-handle,
                                 RSTR #rstr ] !>,
                      HCONS <! [ HARG #rstr,
                                 LARG #noun ] !> ] ].
```

(3.63)
```
go_school := complex-vp-picto-lex &
 [ STEM < "go_school" >,
   SYNSEM.LOCAL.CONT [ HOOK.SCLERA.N1 #N1,
                       RELS <! [ PRED "_go_v_rel",
                                 ARG2 #locative-comp ],
                               #N1 & [ PRED "_school_n_rel",
                                 LBL  #noun,     ; LARG handle
                                 ARG0 #locative-comp ],
                               [ PRED "locative_rel",
                                 ARG2 #locative-comp ],
                               [ PRED "exist_q_rel",
                                 ARG0 #locative-comp,
                                 RSTR #rstr ] !>, ; RSTR handle
                      HCONS <! [ HARG #rstr,
                                 LARG #noun ]  !> ] ].
```

Notice that the descriptions in (3.62) and (3.63) contrast with the relative minimalism of the entries for simplex pictographs that we saw earlier. This is not a necessary property of such entries, however. As more entries are added, and the commonalities between them become more obvious, much of the information currently present on the descriptions of the entries will be factored out and moved to new intermediate types in the hierarchy, such that the entry in (3.62) will ultimately, at the very least, boil down to (3.64):

(3.64)
```
brush_dog := complex-vp-picto-lex &
  [ STEM  < "brush_dog" >,
    SYNSEM.LOCAL.CONT.RELS <! [ PRED "_brush_v_rel" ],
                              [ PRED "_dog_n_rel" ] !> ].
```

Turning now to another kind of 'verbal' complex pictographs, i.e., those which *also* convey the *subject* of their verbal head (e.g., 'dog+bark'; see (3.56)), we introduce the lexical type ***complex-subj+vp-picto-lex***. Because this type already 'has' a subject, its definition does not require any subject-related identity constraints. As a result, its *current* definition, shown by (3.65), is somewhat simpler than that of *complex-vp-picto-lex* above. Like in the previous definition, the first item on the RELS list is of the type *event* and partially identified with the sign's HOOK features. Differently than the previous type, however, *complex-subj+vp-picto-lex* bears a negative value on the boolean attribute MC (Main Clause). This has the unfortunate effect of preventing this pictograph type from being accepted as a well-formed clause by the parser, despite the fact that, to all intents and purposes, it amounts to a complete, i.e., saturated, verb phrase. The reason for this requirement is a side effect of a small constraint inherited from the grammar of Dutch which requires that the MC attribute of the root structure (or start symbol) of a parse tree is positive: that is, that only main clauses are accepted.

(3.65)
```
complex-subj+vp-picto-lex := main-complex-v-picto-lex &
  [ SYNSEM.LOCAL [ CAT  [ HEAD complex-S-picto,
                          VAL.SUBJ < >,
                          MC - ],
                   CONT [ HOOK [ LTOP #lbl_head,
                                 INDEX #event],
                          RELS.LIST.FIRST [ ARG0 event & #event,
                                            LBL #lbl_head ] ] ] ].
```

As a simple, temporary workaround, a new unary rule is added to the grammar whose sole function is to 'pump' lexical signs of the type *complex-subj+vp-picto-lex* up the parse tree, while switching the polarity of the value of the MC so that it becomes positive. As defined in (3.66), this rule is sufficiently constrained to prevent both simplex lexical types and other complex picto types from unifying with its head daughter slot: simplex lexical types are constrained as (specified as `[ complex - ]`), which is incompatible with the `[ complex + ]` constraint associated with the head type *complex-S-picto*; other complex picto types are blocked on account of their non-empty SUBJ list, which conflicts with the empty SUBJ list constraint on the head-daughter. It bears repeating, however, that this rule serves as a makeshift solution to a problem that may not exist once the grammar of *Sclera* is made fully independent of the grammar of Dutch (from which it derives its basis).

(3.66)
```
complex-subj+vp-picto-rule := head-only & declarative-clause &
 [ SYNSEM.LOCAL.CAT [ HEAD complex-S-picto,
                      VAL [ SPR < >,
                            SUBJ < >,
                            COMPS < >,
                            SPEC < > ],
                      MC + ],
   SYNSEM.LOCAL.CONT.HOOK #hook,
   HEAD-DTR lex-item & [ SYNSEM.LOCAL [ CONT.HOOK #hook,
                                        CAT [ HEAD complex-S-picto,
                                              VAL [ SPR < >,
                                                    SUBJ < >,
                                                    COMPS < > ],
                                              MC - ] ] ],
   C-CONT.RELS <! !> ].
```

The lexical entry for ‘dog+bark’ is provided by (3.67). As the example suggests, the descriptions used for entries of this type use a similar strategy of verbosity. Again, future revisions should aim to relocate all save the essence of such descriptions to the type hierarchy.

(3.67)
```
dog_bark := complex-subj+vp-picto-lex &
   [ STEM < "dog_bark" >,
     SYNSEM.LOCAL.CONT [ HOOK.SCLERA.N1 #N1,
                         RELS <! [ PRED "_bark_v_rel",
                                   ARG0 #event,
                                   ARG1 #subject ],
                                 #N1 & [ PRED "_dog_n_rel",
                                   LBL #noun,
                                   ARG0 #subject ],
                                 [ PRED "q_rel_min",
                                   ARG0 #subject,
                                   RSTR #rstr ] !>,
                         HCONS <! [ HARG #rstr ,
                                   LARG #noun ] !> ] ].
```

As I have pointed out throughout this section, there is much room for improvement in the way that the grammar currently incorporates complex pictographs (or rather, the small set of pictographs so far analysed). Nevertheless, within the grammar of *Sclera*, these pictographs are first-class citizens, on a par with simplex types. Thus, they are compatible (where appropriate, of course) with the basic rules of the grammar, as well as with the mechanisms developed in sections Section 3.4.1 and Section 3.4.2 for tense and illocution modification.

3.4.3.1 Bonus round: Adding adjectives

Semantically, the approach to complex pictographs as internally structured 'bundles' of relations poses problems for grammar rules which require access to a different relation than that which is made accessible by the lexical type (via its HOOK features), i.e., the conceptual head, which in this case is always a verb. For instance, adjectives (which, though not discussed above, are, in fact, included in the grammar of *Sclera*) require access to the values of both INDEX and LTOP associated with the noun relation that they modify. As a result, adjectives cannot modify the noun relation 'dog' present in the complex picto 'dog+bark', and the string 'happy dog+bark' is not considered well-formed, even though, intuitively, it should be.

To remedy this, a new type called ***sclera-hooks*** is posited. This type comprises four attributes which take elementary predications as their value. For N1 and N2 these predications are constrained to the general type *noun-relation.* For V1 and V2 this is *event-relation.*

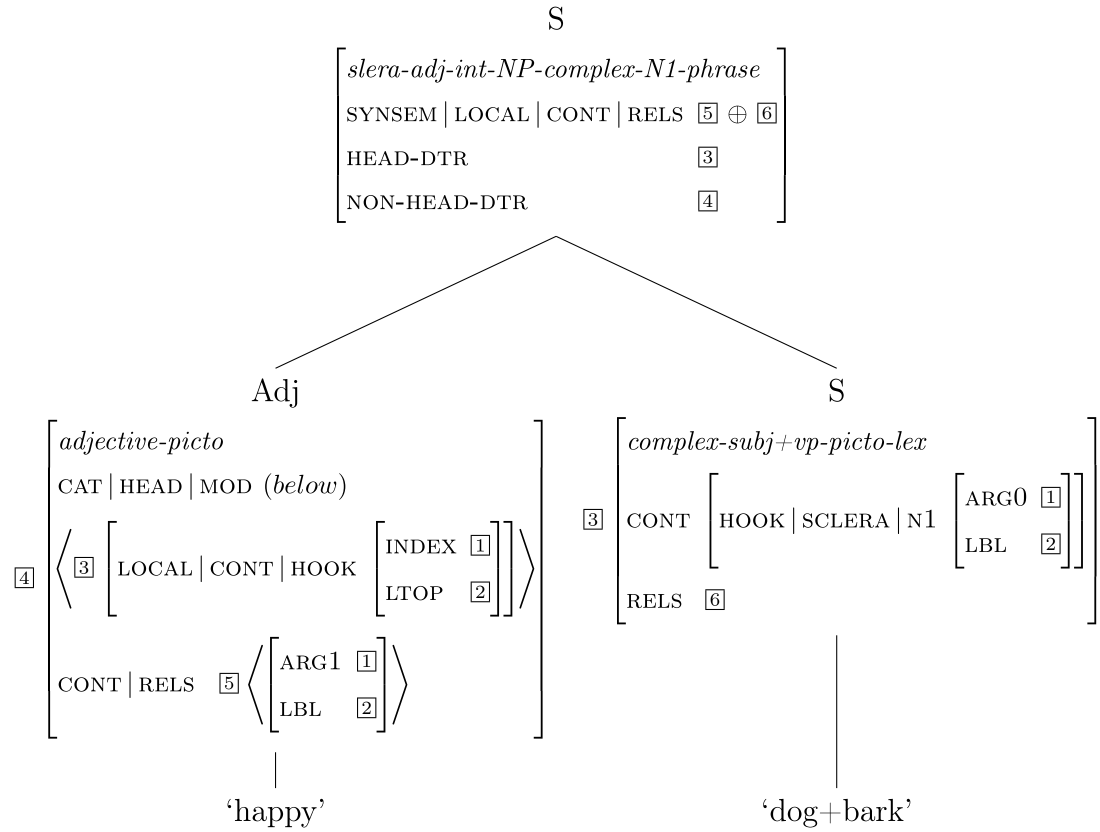
(3.68)
```
sclera-hooks := avm &
    [ N1 noun-relation,
      N2 noun-relation,
      V1 event-relation,
      V2 event-relation ].
```

The new *sclera-hooks* type forms the value of a new feature SCLERA, which is added to the HOOK path of the general type *complex-picto-lex* (i.e., so that it is only appropriate for complex picto types). This is the contribution of the `[ SYNSEM.LOCAL [ CONT.HOOK sclera-hook ...  ]` constraint in (3.59).

As in the examples above, lexical entries (or, in the future, types) can now make the noun-relations on their RELS list 'public' by identifying it with the path `HOOK.SCLERA.N1`. In (3.67), for example, this involves the relation with the PREDicate value '_dog_n_rel'.

By means of the specialized adjective-noun phrase rule *slera-adj-int-NP-complex-N1-phrase*, it is now possible to combine adjectives and complex (essentially verbal) pictographs into well-formed structures, particulary with respect to their semantic integrity. This is illustrated by (3.69), which shows how the *Sclera* string 'happy dog+bark' can now be parsed to yield the MRS equivalent of the logical form in (3.70).

(3.69)



(3.70) $quantifier(x,\ dog(x) \wedge happy(x) \wedge bark(x))$

Chapter 4

Depicto II: Transfer & generation

In the previous chapter, we saw how a string of *Sclera* pictograph identifiers can be analyzed to yield an MRS representation (or simply: MRS) of its semantic content. This chapter introduces the next two modules in the *Depicto* pipeline. In Section 4.1, we see how *Sclera* MRSs are 'translated' to a specific target language grammar; and in Section 4.2, we see how these modified MRSs are used to generate well-formed surface strings in a given target language. Both stages involve the development of new grammars, which are explained in detail. Note, however, that while the transfer grammar is essential to realizing the objective of extensibility set out in the introduction, and while the target language grammar is crucial insofar as it provides proof that the concept of the *Depicto* can be put into practice, the development of these resources was fairly straightforward, requiring *comparatively* little innovation as compared to the previously developed grammar of *Sclera.* To reflect this fact, the discussion is kept brief when there is no risk of confusion.

4.1 Transferring MRSs to the target language

The MRSs produced by *Depicto*'s analysis module (Chapter 3) represent a sort of interlingua, albeit in a very loose sense of the term. That is, the predicates, variable names, and other constituent structures contained in these MRSs are defined independently of any specific natural language[1]. As a result, despite using English for their predicate names, *Sclera* MRSs are equally compatible with any DELPH-IN (target) grammar, which is in fact to say that they are equally *incompatible*, for, as I explain in Section 2.3, DELPH-IN grammars encode semantic information in a highly language-specific way. Thus, in order for *Sclera* MRSs to be used for generation with a natural language grammar, they must first be adapted, or *transferred*, to the model of semantics used by that grammar. This is the function of the second module in the *Depicto* processing chain: the transfer module.

The transfer grammar used by this second module is described here. In the present case, it is a prototype for translation from *Sclera* to Dutch. It is written in the LOGON transfer formalism and processed by the ACE implementation of the LOGON transfer engine (see Section 2.3). Besides providing an ad hoc bridge between two grammars (and serving as the keystone in the *Depicto* system as a whole), it gives an idea of how other, more extensive DELPH-IN target grammars could be 'plugged in' in the future. Without modifying either the *Sclera* or chosen target language grammar, a developer wishing to extend the *Depicto* system to a new language would simply need to provide a transfer grammar for the new language pair.

[1] A good example of this can be found in (3.63), where the prepositional complement of 'go' in the lexical entry 'go+school' is identified not (as in the ERG(Flickinger et al. 2003) grammar) by the predicate 'to_p_rel', but by the more abstract predicate 'locative_rel'.

4.1.1 A *Sclera*-Dutch transfer grammar

With a common basis in the Matrix type hierarchy, and developed largely in parallel (as explained in Section 3.2), the grammar of *Sclera* and the grammar of Dutch (next section) are, in most respects, mutually compatible as regards the MRS structures which they produce or accept. In fact, the only source of semantic incompatibility between the current grammars, concerns predicate names, as specified on each elementary predication's PRED attribute. As a result, the *Sclera*-Dutch transfer grammar has the relatively simple function of translating *Sclera* predicate names to Dutch predicate names, as visualised in (4.1).

(4.1) $$\begin{bmatrix}\textsc{pred} & \text{‘\_headphones\_n\_rel’}\end{bmatrix} \Longrightarrow \begin{bmatrix}\textsc{pred} & \text{‘\_koptelefoon\_n\_rel’}\end{bmatrix}$$

As explained in Section 2.3, the transfer grammar consists of two main components: a hierarchy of transfer rule types (or 'general patterns of translational correspondence' (Oepen 2008)) and a sequentially ordered list of transfer rule instances. In addition, a basic theory of MRS types is required[2]. The file structure of the transfer grammar is given in (4.2):

[2] Not mentioned so far is that referential and event variables tend to vary across grammars in the additional properties (e.g., person, number, tense) that they take as well as the names which these properties are given. In most cases this is handled by a separate part of the grammar, which, using a different formalism, achieves so-called *Variable-Property Mapping.* Currently, though, there is only minimal variation between the grammars of *Sclera* and Dutch.

(4.2)	`mrs.tdl`	Basic implementation of Minimal Recursion Semantics (borrowed from JaEn (Bond et al. 2011))
	`mtr.tdl`	A very minimal hierarchy of MRS Transfer Rule types (also borrowed from JaEn (Bond et al. 2011))
	`transfer.mtr`	ordered list of MTR entries
	`in.vpm`, `out.vpm`	Variable Property Mapping applied to input and output of transfer grammar. (Not discussed here; see `moin.delph-in.net/RmrsVpm`)
	`config.tdl`, `top.tdl`	Configuration files passed to ACE during compilation.

In accordance with the LOGON transfer formalism, all transfer rules in the grammar inherit their basic structure from the top-level type *mrs_ transfer_ rule*, whose definition is shown in (4.3).

(4.3)
```
mrs_transfer_rule := *top* &
     [ FILTER mrs,
       CONTEXT mrs,
       INPUT mrs,
       OUTPUT mrs,
       FLAGS flags ].
```

This happens via the intermediate subtype *monotonic_ mtr* (4.4), which uses identity constraints to ensure that MRSs passing through the rule retain the values of their local top handle (LTOP) and index.

(4.4)
```
monotonic_mtr := mrs_transfer_rule &
     [ CONTEXT [ LTOP #h, INDEX #i ],
       INPUT [ LTOP #h, INDEX #i ],
       OUTPUT [ LTOP #h, INDEX #i ] ].
```

With regard to the content of the elementary predications themselves, a number of subtypes of *monotonic_ mtr* are posited which correspond to different kinds of predication. *EP*s containing noun relations, for instance, are transferred by rules of the type *noun_ mtr* . As (4.5) shows, this type specifies that the label and the referential index (the value of ARG0) of the *EP* on the input MRS is to be kept constant throughout the rule.

(4.5)
```
noun_mtr := monotonic_mtr &
    [ INPUT.RELS < [ LBL #h1, ARG0 #x1 ] >,
      OUTPUT.RELS < [ LBL #h1, ARG0 #x1 ] > ].
```

In the non-type hierarchy part of the grammar – `transfer.mtr`; see (4.2) – *hond_ mtr* is specified as an instance of *noun_ mtr*. As shown in (4.6), *hond_ mtr* consumes the PREDicate value '_dog_n_rel' and replaces it with the Dutch equivalent '_hond_n_rel'.

(4.6)
```
hond_mtr := noun_mtr &
     [ INPUT.RELS < [ PRED "sclera:_dog_n_rel" ] >,
       OUTPUT.RELS < [ PRED "_hond_n_rel" ] > ].
```

Notice that the input PREDicate value in (4.6) is prefixed by 'sclera:'. This is added automatically to all relations of the transferred MRS by virtue of the optional ACE configuration parameter in (4.7). Although there exists (to my knowledge) no actual documentation for it (it has in fact been gleaned from the JaEn transfer grammar (Bond et al. 2011)), it functions as a measure to prevent cyclical rule application of the sort that occurs when a rule's input and output are identical, as is the case in (4.8), where the name of the predicate 'school' is the same in both *Sclera* and Dutch. This is just one of many examples of words whose orthographic form and meaning are shared between the English metalanguage of *Sclera* and Dutch. An alternative solution to the problem of identical predicate names might be to add a separate class of *passthrough* rules which allow problem cases to simply exit the transfer grammar at the end. For the sake of simplicity, however – which is to say, this solution might be a dead end – passthrough rules are currently not included.

(4.7)
```
input-relation-prefix := "sclera:".
```

(4.8)
```
school_mtr := noun_mtr &
     [ INPUT.RELS < [ PRED "_school_n_rel" ] >,
       OUTPUT.RELS < [ PRED "_school_n_rel" ] > ].
```

Back to the type hierarchy, here a number of subtypes of *monotonic_ mtr* are further posited for 'event' predications, i.e., where the value of ARG0 is an event variable. The prototypical case involves 'verb' predicates. The rule types used here differ in the number of ARG-N attribute variables between which they enforce identity. For example, *arg123_ v_ mtr* (4.9) supports three such variables in addition to the ARG0 index. These ARG-variables correspond to the verb's subject, direct object, and indirect object.

(4.9)
```
arg123_v_mtr := monotonic_mtr &
    [ INPUT.RELS < [ LBL #h1,
                    ARG0 #e1, ARG1 #x1, ARG2 #x2, ARG3 #x3 ], ... >,
      OUTPUT.RELS < [ LBL #h1,
                     ARG0 #e1, ARG1 #x1, ARG2 #x2, ARG3 #x3 ], ... > ].
```

Example (4.10) shows the MTR entry for the optionally ditransitive predicate 'buy'. When used as a simple transitive predicate, 'buy' is still consumed by the rule, but the value of ARG3 is left underspecified.

(4.10)
```
kopen_mtr := arg123_v_mtr &
     [ INPUT.RELS < [ PRED "sclera:_buy_v_rel" ] >,
       OUTPUT.RELS <[ PRED "_kopen_v_rel" ] > ].
```

Within the model of semantics implemented in the Matrix (Flickinger et al.

2003), adverbial and prepositional relations are analysed as contributing an event variable, just like verbs. As a result, the same rule transfer types used for verb relations apply to other relations of the 'event' kind kind. For example, (4.11) shows an MTR instance for the adverbial relation 'yesterday'. This relation involves a single argument variable, so the MTR is identified as an instance of *arg1_v_mtr*.

(4.11)
```
yesterday_r_mtr := arg1_v_mtr &
       [ INPUT.RELS < [ PRED "sclera:_yesterday_r_rel" ] >,
         OUTPUT.RELS < [ PRED "_gisteren_r_rel" ] > ].
```

In the previous section we saw how the locative complement (or *obligatory locative adjunct*) 'to school' in the complex pictograph 'go+school' is analysed as consisting of the predicate '_school_n_rel' and the very abstract (prepositional) relation 'locative_rel' (which could, I admit, have been labelled slightly more clearly). In the MTR instance in (4.12) this abstract relation is translated into the concrete prepositional relation 'naar_p_rel', which corresponds, in the current context, to English *to* . This rule does not take into account, however, the fact that the formal realization of locative complements varies in accordance with the valency requirements of the individual verbs that select them. While this does not pose a problem for the current system, which covers only one relation that subcategorizes for a locative adjunct, viz., 'go_v_rel', the situation changes once one wishes to add other relations, such as 'sit_v_rel', whose locative adjunct is realised in English with the presposition *on*, rather than with *to.* The MTR in (4.12), therefore, should in principle be modified so as either to output a more general relation whose correspondence to a specific prepositional form is determined by the target grammar, or to take contextual features into account (using the CONTEXT part of the transfer rule) such that specific prepositional forms are constrained to specific verb relations. These improvements are deferred to future revisions.

(4.12)
```
locative_naar_mtr := arg12_v_mtr &
```

```
    [ INPUT.RELS < [ PRED "sclera:locative_rel" ] >,
      OUTPUT.RELS < [ PRED "naar_p_rel" ] > ].
```

The type hierarchy also contains a subtype of *monotonic_ mtr* which, intended for use with quantification predications, keeps the values of RSTR and BODY constant throughout the rule. (Incidentally, the list of handle constraints, to which none of the rules in the current transfer grammar refer, is simply passed through unaltered, such that all scope constraints are maintained.)

(4.13)
```
quantifier_mtr := monotonic_mtr &
    [ INPUT.RELS < [ LBL #h1, ARG0 #x1, RSTR #h2, BODY #h3 ] >,
      OUTPUT.RELS < [ LBL #h1, ARG0 #x1, RSTR #h2, BODY #h3 ] > ].
```

Example (4.14) shows the MTR for the general quantifier predicate 'q_rel_min' (introduced by the *sclera-det-insert* rule in Section 3.2.3). While the input and output of this rule are specified for the same relation name, notice that the value of PRED on the input is a string, whereas the same value on the output is not. In the DELPH-IN formalism any attribute value that is not a string (i.e., not surrounded by single or double quotation marks) is interpreted as a type. In the transfer grammar, this type is defined in `mrs.tdl`. As we move on to the third module in the *Depicto* chain, the role of this type within the (target) grammar of Dutch will become clear.

(4.14)
```
quant_mtr := quantifier_mtr  &
    [ INPUT.RELS < [ PRED "sclera:q_rel_min" ] >,
      OUTPUT.RELS < [ PRED q_rel_min ] > ].
```

The transfer grammar developed in this section contains MTRs for each of the relations defined in the grammar of *Sclera* and is able to transfer any output produced by *Depicto*'s analysis module. So far, the grammar has been kept purposefully simple. Currently, neither the optional CONTEXT nor the optional FILTER feature of the LOGON formalism is used; all rules are marked as obligatory; and rule order currently makes no difference (which is preferable, anyway). Much of this has to do with the fact that documentation is both scarce and fragmented: there is no single source that offers a clear-cut set of instructions on how to set up a LOGON transfer grammar, so that the approach taken is one of trial and error. This has limited the amount of progress I have been able to make on this front. Nevertheless, the current transfer grammar serves the needs of the *Depicto* system well.

4.1.2 Output of the transfer stage

The output of the ACE implementation of the LOGON transfer engine is a plain-text MRS representation, similar to that in Section 3.3. To illustrate this, (4.15) shows the MRS obtained by feeding the transfer module the output of the analysis module for the input string 'I see dog yesterday'. Except for the translated relation names, which are summarized in (4.16), the output is identical to that produced by the analysis module. The bash script responsible for this step is provided in (4.17). The curly brackets `{}` must be substituted for either `linux` or `osx`, depending on the operating system (Windows is not supported). (Please see Appendix A for more information about getting up and running with *Depicto*.)

(4.15) Result of transferring *Sclera* analysis of 'I see dog yesterday'

```
[ LTOP: h13
  INDEX: e12 [ e SF: prop-or-ques E.TENSE: past E.ASPECT: aspect E.MOOD: mood ]
  RELS: < [ q_rel_min<-1:-1> LBL: h0 ARG0: x9 [ x PNG.PER: person
                                                  PNG.NUM: number
                                                  PNG.GEND: gender ]
                                     RSTR: h1
                                     BODY: h2 ]
          [ exist_q_rel<-1:-1> LBL: h3 ARG0: x8 [ x PNG.PER: 1st
                                                    PNG.NUM: singular
                                                    PNG.GEND: gender ]
                                       RSTR: h4
                                       BODY: h5 ]
          [ "_hond_n_rel"<-1:-1> LBL: h6 ARG0: x9 ]
          [ "_vnw_n_rel"<-1:-1> LBL: h7 ARG0: x8 ]
          [ "_zien_v_rel"<-1:-1> LBL: h10 ARG0: e12 ARG1: x8 ARG2: x9 ]
          [ "_gisteren_r_rel"<-1:-1> LBL: h10 ARG0: e11 [ e E.TENSE: tense
                                                            E.ASPECT: aspect
                                                            E.MOOD: mood ]
                                                  ARG1: e12 ] >
  HCONS: < h13 qeq h10 h4 qeq h7 h1 qeq h6 > ]
```

(4.16)	‘q_rel_min’	⟶	*q_rel_min*
	‘exist_q_rel’	⟶	*exist_q_rel*
	‘_dog_n_rel’	⟶	‘_hond_n_rel’
	‘_pronoun_n_rel’	⟶	‘_vnw_n_rel’
	‘_see_v_rel’	⟶	‘_zien_v_rel’
	‘_yesterday_r_rel’	⟶	‘_gisteren_r_rel’

(4.17)

```
 cd PATH/TO/ACE/BINARY/IN/REPOSITORY/DIRECTORY
echo "i see dog yesterday" | ./ace_{} -g ../mini-sclera.dat | ./ace_{} -g ../transfer-nl.dat -f
```

4.2 MRS-based target language generation

In this section, we see how the third and final module in the *Depicto* processing chain uses logical form representations of the kind obtained at the end of the transfer stage (Section 4.1) to generate surface strings in accordance with the constraints of a given target language grammar model. The version of the *Depicto* system developed here translates to Dutch. Since no ready-made DELPH-IN grammars exist for this language at the time of development, a small ‘toy’ grammar is used instead, which, as Section 3.2.2 explains, is developed in parallel with the grammar of *Sclera* used in the analysis module. While certainly not the focus of the work presented in this thesis, this grammar plays an integral part in demonstrating the viability of the *Depicto* pipeline as a whole and, therefore, warrants a brief but proper introduction first.

4.2.1 Setting up a toy grammar of Dutch

Dutch is a Germanic language that exhibits both SVO word order, in main clauses, and SOV word order, in subordinate clauses (Haeseryn 1997). The finite verb of a Dutch clause generally occurs in the second sentence position, a pattern referred to as *V2 order* (Zwart 2011). Non-finite verb complements are generally postponed to the so-called *second pole* position, which is found at the end of the clause. Raising, control, and auxiliary verbs can trigger verb clustering (Augustinus 2015). These are just some of the syntactic characteristics which an adequate grammar of Dutch needs to be able to account for and which require such a grammar to incorporate machinery and implementations of analyses very different to those required for a grammar of, say, English. In short, the task of developing a true implemented grammar of Dutch is worth a master's thesis (if not several) in its own right.

To keep things simple, therefore, the aforementioned characteristic are largely ignored and Dutch is approached as a simple SVO language comparable to English. As explained in Section 2.2.3, this analysis yields an initial customized Matrix grammar of Dutch. (We have already seen how the grammar of *Sclera* 'borrows' its types from this grammar.) The result is manually extended and refined with the aid of snippets and inspiration found in the English Resource Grammar (Flickinger et al. 2014) and in several grammars of German related to work by Fokkens (2014), so that, ultimately, the grammar is able to cover a number of basic phenomena. To save space, these are summarized in (4.18):

(4.18) Phenomena covered by grammar of Dutch

- Verbs
  - Subject-verb agreement (number and person)
    - with support for different agreement patterns under subject-verb inversion
  - Present and simple past (preterite) tense
  - Intransitive, intransitive with prepositional complement, transitive, and ditransitive valency frames

        - with correct nominative-accusative case marking (Note: indirect object slot incorrectly marked as 'accusative', rather than as 'ditransitive')
        - lexical rule that allows ditransitive verbs to drop their indirect object (i.e., be transitive).
        - lexical rule that allows for dative alternation: e.g., 'the astronaut gives the martian a present' vs. 'the astronaut gives a present to the martian'.

- Nouns
    - determiner-noun agreement (gender (neuter v. non-neuter) and number)
    - Common nouns and mass nouns
    - Pronouns
        - Specified for person, number, gender and case.
- Determiners
    - definite determiner (two forms *de* and *het*)
    - indefinite determiner
    - 'zero' determiner (simply analysed as bare noun phrase)
- Prepositions, including
    - Case marker *aan* ('to') (Interacts with dative alternation rule)
    - Locative relation *naar* (also 'to').
- Adjectives
    - Intersective adjectives
    - Adjective-Noun agreement
        - Combination of lexical and inflectional rules for $+e$ affix.
- Adverbs
    - simple adverb-verb phrase
- **General phrasal patterns**
    - head-complement phrases (head first)
    - subject-head phrases (subject first)
    - head-specifier phrases (specifier first)

  - (and interrogative-clause phrases)
    * interacts with lexical rule for subject-verb inversion.

In illustrating the relative underspecificity of lexical entries in the grammar of *Sclera*, the previous chapter (see Section 3.2.2) already provides two examples of lexical entries from the grammar of Dutch. There, in (3.17), the entry for *hond* ('dog') is shown to be appropriately specified for number (*sg*) and gender (*femmasc*, aka 'non-neuter'), while, in (3.21), the entry for *slaapt* ('sleeps' and 'sleep' (second person)) is specified for number (*sg*), person *2nd-or-3rd* and tense (by default: present). Because the grammar of Dutch currently does not use inflectional rules (except for adjectives), the surface strings with with these lexical entries are associated (as specified, somewhat misleadingly, under the attribute STEM) are the inflected forms of the lexeme, not its stem. The complete inflectional paradigms of the lexemes *hond*('dog') and *slapen* ('sleep') (as far as the grammar is concerned, at least) are shown in (4.19) and (4.20), respectively.

(4.19)
```
hond := femmasc-count-sg-noun-lex &
  [ STEM < "hond" >,
    SYNSEM.LKEYS.KEYREL.PRED "_hond_n_rel" ].

honden := femmasc-count-pl-noun-lex &
  [ STEM < "honden" >,
    SYNSEM.LKEYS.KEYREL.PRED "_hond_n_rel" ].
```

(4.20)
```
slaap := intrans-1st-sg-verb-lex &
  [ STEM < "slaap" >,
    SYNSEM.LKEYS.KEYREL.PRED "_slapen_v_rel" ].

slaapt := intrans-2nd-or-3rd-sg-verb-lex &
  [ STEM < "slaapt" >,
    SYNSEM.LKEYS.KEYREL.PRED "_slapen_v_rel" ].
```

```
slapen := intrans-pl-verb-lex &
 [ STEM < "slapen" >,
   SYNSEM.LKEYS.KEYREL.PRED "_slapen_v_rel" ].

sliep := intrans-sg-preterite-lex &
 [ STEM < "sliep" >,
   SYNSEM.LKEYS.KEYREL.PRED "_slapen_v_rel" ].

sliepen := intrans-pl-preterite-lex &
 [ STEM < "sliepen" >,
   SYNSEM.LKEYS.KEYREL.PRED "_slapen_v_rel" ].
```

Though not discussed so far, pronouns are relatively well specified in the grammar of *Sclera*, marked for both person and number, as shown in (4.21). Yet even this entry, which is not specified for case, has more specific counterparts in the grammar of Dutch, where pronouns encode either accusative/object or nominative/subject case. (This is technically incorrect, since there is a small number of Dutch pronouns that encode dative or genitive case, but this is ignored for now.) The paradigm corresponding to the first person singular *Sclera* pronoun *I-me* is given in (4.22).

(4.21) Lexical entry for first person singular pronoun in *Sclera* grammar (case-agnostic)

```
I-me := 1st-sg-pronoun-noun-lex &
      [ STEM < "I" >,
        SYNSEM.LKEYS.KEYREL.PRED "_pronoun_n_rel" ].
```

(4.22) Lexical entries for first person singular pronoun in Dutch grammar (case-specific)

```
ik := 1st-sg-pronoun-noun-lex &
```

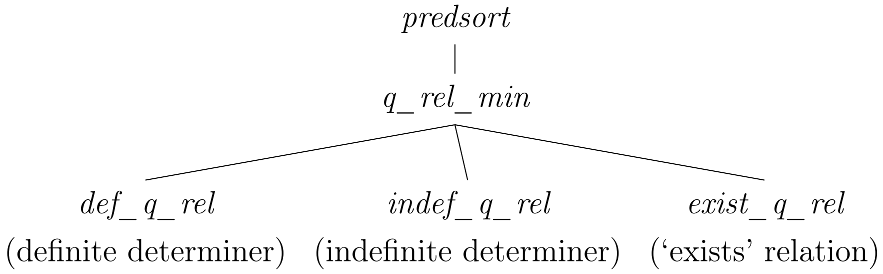
```
  [ STEM < "ik" >,
    SYNSEM.LOCAL.CAT.HEAD.CASE nom, ;;subject
    SYNSEM.LKEYS.KEYREL.PRED "_pronoun_n_rel" ].

me_1 := 1st-sg-pronoun-noun-lex &
  [ STEM < "me" >,
    SYNSEM.LOCAL.CAT.HEAD.CASE acc,  ;;object
    SYNSEM.LKEYS.KEYREL.PRED "_pronoun_n_rel" ].
```

4.2.1.1 Typed determiner relations

In the introduction to the *Sclera*-Dutch transfer grammar in (Section 4.1.1), we saw that the relation name (or PRED value) 'q_rel_min' is translated from an (atomic) string to an identically named type, viz., *q_rel_min*. In the type hierarchy of the grammar of Dutch, this relatively abstract relation is defined as a supertype of three relation types that each relate to a concrete kind of determination or quantification. As a preview of things to come, this was already explained in Section 3.2.3. The visualisation of the type hierarchy provided there by (3.36) is repeated here in (4.23):

(4.23)



While the relation type *exist_q_rel* is used in the construction of bare noun phrases (as a sort of 'zero' determiner), the types *def_q_rel* and *indef_q_rel* are used by lexical entries for definite and indefinite determiners, which are defined as shown in (4.24) and (4.25), respectively.

(4.24)
```
de_sg := definite-determiner-lex &
  [ STEM < "de" >,
    SYNSEM [ LKEYS.KEYREL.PRED def_q_rel,
             LOCAL.CONT.HOOK.INDEX.PNG [ GEND non-neuter,
                                         NUM  singular ] ] ] .

de_pl := definite-determiner-lex &
  [ STEM < "de" >,
    SYNSEM [ LKEYS.KEYREL.PRED def_q_rel,
             LOCAL.CONT.HOOK.INDEX.PNG [ NUM plural ] ] ] .

het := definite-determiner-lex &
  [ STEM < "het" >,
    SYNSEM [ LKEYS.KEYREL.PRED def_q_rel,
             LOCAL.CONT.HOOK.INDEX.PNG [ GEND neuter,
                                         NUM singular] ] ].
```

(4.25)
```
een := indefinite-determiner-lex &
  [ STEM < "een" >,
    SYNSEM [ LKEYS.KEYREL.PRED indef_q_rel,
             LOCAL.CONT.HOOK.INDEX.PNG [ NUM singular ] ] ].
```

Recall from the 'grammar profile' of *Sclera* in Section 3.2 that it is largely impossible to infer how the quantification of a given *Sclera* noun is intended to be read. As we see in the next section, by situating determiner/quantifier relations within a subsumption hierarchy and using only the maximal types in the definition of lexical entries, we achieve the effect that for each occurrence of *q_rel_min* passed to the grammar during generation, three quantification types exist, each of which is explored equally and in accordance with the other constraints of the grammar. In other words, each occurrence of *q_rel_min* 'expands' to three different quantification types, which is, of course, reflected by the determiner tokens instantiated on the generated surface string.

This solution works for the current version of the *Depicto* pipeline. However, it should also be noted that defining types in the target language grammar that are specifically tailored to analyses produced by the *Sclera* module is a violation of the design aim that the target language grammar in the *Depicto* pipeline be kept wholly independent of the rest of the modules in the chain. Future extensions of the system cannot assume that third-party DELPH-IN grammars use the same system of typed relations for quantifiers, nor would it be feasible to alter these grammar to do so. One of the central aims of the *Depicto* system, in fact, is that third-party grammars can be plugged into the target language slot, such that the only work left for developers resides in the construction of a transfer grammar. Ultimately, therefore, the treatment of quantification and determination will need to be relocated to a different stage of the pipeline. The most likely candidate is the transfer stage, where a small sequence of non-obligatory transfer could achieve much the same effect, albeit at the cost of an increase in the work which the transfer engine would have to do.

4.2.1.2 Experiments in synonymy

The toy grammar of Dutch also features a basic implementation of synonymy. This rests on the principle that lexical entries for lexemes with a shared meaning bear the *same* relation name. This is illustrated in (4.26) for the Dutch verbs *wandel* and *loop*, which both refer to a process of 'walking', and in (4.27) for the verbs *kammen* and *borstelen*, which refer to a process of 'brushing' in the 'combing' sense (i.e., where the conceptual object relates to 'hair').

(4.26) Near synonyms of 'walk' (v) (ignoring regional variation)

```
wandel := intrans+pp-1st-sg-verb-lex  &
  [ STEM < "wandel" >,
    SYNSEM.LKEYS.KEYREL.PRED "_wandelen_v_rel" ].
```

```
loop := intrans+pp-1st-sg-verb-lex  &
  [ STEM < "loop" >,
    SYNSEM.LKEYS.KEYREL.PRED "_wandelen_v_rel" ].
```

(4.27) Near synonyms of ‘brush’ (v), when object = hair.

```
kammen := trans-pl-verb-lex &
  [ STEM < "kammen" >,
    SYNSEM.LKEYS.KEYREL.PRED "_borstelen_v_rel" ].

borstelen_1 := trans-pl-verb-lex &
  [ STEM < "borstelen" >,
    SYNSEM.LKEYS.KEYREL.PRED "_borstelen_v_rel" ].
```

The effect of synonymy on the generation process is that for one relation encountered on the input MRS, multiple lexical items may be retrieved from the lexicon and instantiated as an edge on the generation chart (Carroll et al. 1999), i.e., the initial search space. Because most relations in the grammar of Dutch, which follows the example of other Matrix-based grammars, are defined as strings rather than types, this approach to synonymy is rather limited in the degree of semantic... ‘nuance’ that it can capture. No two lexical items are exactly synonymous, be it on a purely semantic, pragmatic or collocational level. One way to capture the complexity implied by this fact might, after all, be to use types instead. These could be set up hierarchically so as to multiply inherit from an ontology of semantic knowledge. Of course, implementing such an ontology would be no small undertaking, so we keep things simple here. Whichever solution is eventually chosen, it should be mentioned that the discussion here invites the same criticism as above: either solution requires modifications to the target language grammar. However, because in this case the target language grammar would be modified independently of the rest of the pipeline, this does not form a problem for any of *Depicto*’s design principles.

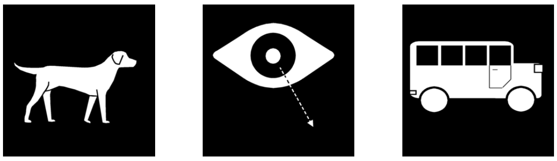
4.2.2 Exhaustive generation of well-formed surface strings

Just like the analysis and transfer modules, the generation module in the *Depicto* processing chain is powered by the ACE processor (see Section 2.2.4). Using the grammar of Dutch presented above, ACE takes a compatible input MRS and generates a list of *all* strings that are related to this MRS by the grammar. To put this slightly differently, all strings included on the generator's output are (a) well-formed with regard to the grammar's constraints, and (b) subsumed (via the grammar) by the semantic representation provided as input. In addition, they constitute an exhaustive list of surface string hypotheses: the chart generator (Carroll et al. 1999) explores all possible combinations, permutations, and derivations of the lexical entries associated with the input MRS.

Taking the *Depicto* system as a whole, we can now see how the MRS analysis of the *Sclera* string 'dog see bus' (4.28), which is discussed in detail in Section 3.3, ultimately (after being transferred) gives rise to the translationally 'equivalent' Dutch sentences listed in (4.29). In total, there are 25. Notice that, while not all of them make as much pragmatic sense as the next, they are all, syntactically speaking, well-formed[3]. Underspecified for number, *Sclera* 'dog' is realized both by singular and plural nouns. All singular nouns are determined as having either definite or indefinite reference, and all plural nouns have a definite determiner or form a bare noun phrase. Underspecified for tense, *Sclera* 'see' is realized in both the present and past tense. All subjects are congruent with the verb phrase, and all determiners are congruent with the head of the noun phrase. Combining all the possible permutations of these realization options shows just how many different translations a simple *Sclera* string can correspond to, at least from a syntactic point of view (i.e., bearing in mind that not all of the generated sentences are equally felicitous).

[3] At the time of writing (2016-07-14), the target grammar contains an error which allows count nouns to occur in singular bare noun phrases. This error should be easy enough to fix if I have time to spare. For now, however, any instances of overgeneration resulting from this error have been erased from the example output.

(4.28)

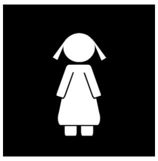


(4.29)

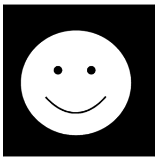
`De honden zagen de bussen`	‘The dogs saw the buses’
`De hond ziet de bussen`	‘The dog sees the buses’
`De hond zag bussen`	‘The dog saw buses’
`De honden zagen bussen`	‘The dogs saw buses’
`Een hond zag de bussen`	‘A dog saw the buses’
`Een hond zag de bus`	‘A dog saw the bus’
`De hond zag de bussen`	‘The dog saw the buses’
`Een hond ziet bussen`	‘A dog sees buses’
`De honden zien bussen`	‘The dogs see buses’
`Een hond ziet de bussen`	‘A dog sees the buses’
`De honden zien bus`	‘The dogs see bus’
`Een hond ziet de bus`	‘A dog sees the bus’
`Honden zagen bussen`	‘Dogs saw buses’
`Honden zien de bussen`	‘Dogs see the buses’
`Honden zagen de bussen`	‘Dogs saw the buses’
`Honden zien de bus`	‘Dogs see the bus’
`Honden zagen de bus`	‘Dogs saw the bus’
`Honden zien bussen`	‘Dogs see buses’
`De hond ziet bussen`	‘The dog sees buses’
`De honden zien de bussen`	‘The dogs see the buses’
`De honden zagen de bus`	‘The dogs saw the bus’
`De honden zien de bus`	‘The dogs see the bus’
`Een hond zag bussen`	‘A dog saw buses’
`De hond zag de bus`	‘A dog saw the bus’
`De hond ziet de bus`	‘The dog sees the bus’
(25 results)	

To give another example, this time involving complex pictographs and adjectives at the *Sclera* end and synonymy at the target language end, (4.31) lists the strings that ACE generates based on the post-transfer MRS representation of the *Sclera* string in (4.30). To save space, this list is presented in a somewhat abridged form, omitted portions indicated by `(...)`). Note that in this instance generation results in a total of 48 well-formed translation hypotheses. Both the alternation between the synonyms *borstel* (‘brush’) and *kam* (‘comb’), and the presence of the adjective *blij* (‘happy’), inflected to agree with the gendered noun *hond* ‘dog’, are new here. As in the previous example, because the input MRS is underspecified for tense, both present and past tense realizations are licensed.

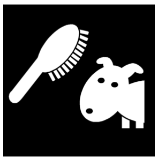
(4.30)



girl



happy



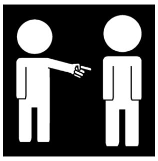
brush+dog

(4.31)

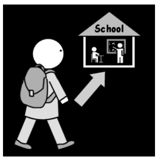
`Het meisje borstelde de blije honden`	'The girl brushed the happy dog'
`Een meisje borstelde de blije honden`	'A girl brushed the happy dogs'
`Meisjes kammen blije honden`	'Girls comb happy dogs'
`Meisjes kammen de blije hond`	'Girls comb the happy dog'
`Het meisje kamt blije honden`	'The girl combs the happy dog'
`Meisjes borstelden de blije hond`	'Girls brushed the happy dog'
`Het meisje borstelde de blije hond`	'The girl brushed the happy dog'
`Het meisje borstelt blije honden`	'The girl brushes happy dogs'
`Een meisje kamt blije honden`	'A girl combs happy dogs'
`Een meisje borstelde de blije hond`	'A girl brushed the happy dog'
`(...)`	
`Het meisje kamt een blije hond`	'The girl combs a happy dog'
`Een meisje kamt een blije hond`	'A girl combs a happy dog'
`Meisjes borstelen de blije hond`	'Girls brush the happy dog'
`Meisjes borstelen een blije hond`	'Girls brush a happy dog'
`De meisjes kammen een blije hond`	'The girls comb a happy dog'
`De meisjes borstelen de blije hond`	'The girls brush the happy dog'
`De meisjes borstelen een blije hond`	'The girls brush a happy dog'
`Een meisje borstelde een blije hond`	'A girl brushed a happy dog'
`(48 results)`	

When the input MRS *is* marked for a specific tense, however, the grammar is sufficiently constrained for the generator to produce sentences accordingly. This is illustrated in (4.33) for the post-transfer *Sclera* analysis of (4.32). In contrast to the previous examples, the output consists of a mere five strings (which is, of course, actually a good thing). One reason for this, aside from the past tense constraint contributed by 'yesterday', has to do with the fact that personal pronouns have only one case-appropriate realization option within the current target grammar. (Granted, the grammar currently does ignore alternation between the clitic (short, non-focus) and non-clitic (focus-attracting) pronominal forms of Dutch.) Another, less fortunate, reason is that the grammar currently overgenerates in two respects. First, as reflected in the English translation of the second string in (4.32), it allows adverbs (here: *gisteren* ('yesterday')) to occur on second sentence position, which violates the *V2* word order principle of Dutch (Zwart 2011). Second, the third string in the example is, in fact, repeated two more times in the section that is left away. These sources of overgeneration will eventually be removed. Once that is done, generation will produce only the first and third strings in the example. The example output in (4.33) additionally demonstrates the target grammar's ability to invert the subject and verb of the clause when the MRS's index is specified for the interrogative mood.

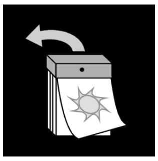
(4.32)



you (singular)



go+school



yesterday

question

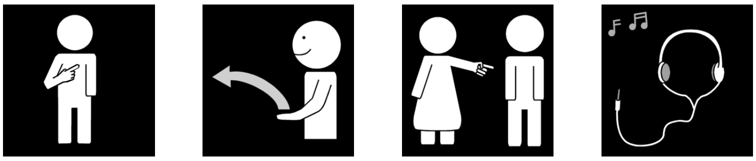
(4.33)

```
Ging je gisteren naar school      'Did you go to school yesterday?'
Ging gisteren je naar school      *'Did yesterday you go to school?'
Ging je naar school gisteren      'Did you go to school yesterday'
(...)
(5 resuts)
```

Finally, as an example of generation with three-place verb predicates, (4.35) lists the Dutch sentences licensed by the post-transfer MRS of (4.34). Both complements of the ditransitive verb 'give' are correctly treated as objects, as indicated by the objective/accusative[4] form of the personal pronoun *hem* ('him'), which serves as the indirect object of the verb. By virtue of the Dutch grammar's support for dative alternation, this pronoun is realized both as a noun phrase and inside the prepositional phrase *aan hem* ('to him').

[4] Technically, as the indirect object of the verb, this item should be marked for the dative case; but since in Dutch this distinction is significant only for one personal pronoun pair, viz., third person plural *hen* (accusative) vs. *hun* dative, it is disregarded here.

(4.34)



I *give* *he* *headphones*

(4.35)

`Ik geef hem de koptelefoon`	'I give him the headphones'
`Ik gaf de koptelefoon aan hem`	'I gave the headphones to him'
`Ik geef de koptelefoon aan hem`	'I give the headphones to him'
`Ik geef hem een koptelefoon`	'I give him (a pair of) headphones'
`Ik gaf een koptelefoon aan hem`	'I gave (a pair of) headphones to him'
`Ik geef een koptelefoon aan hem`	'I give (a pair of) headphones to him'
`Ik gaf hem de koptelefoon`	'I gave him the headphones'
`Ik gaf hem een koptelefoon`	'I gave him (a pair of) headphones'
`8 results`	

As the output of generation shown in examples (4.29), (4.31), (4.33), and (4.35) suggests, rule-based generation from MRS representations yields surface strings which, from a strictly grammatical standpoint, are of very high quality indeed. Moreover, even when there are 48 string hypotheses (as in (4.31), although by no means does this represent the maximum), the ACE processor is able to produce them instantly. (Granted, speed is a trait of all of *Depicto*'s modules.)

In the current version of the *Depicto* pipeline, however, this multiplicity of strings also forms the main output, which raises questions about *Depicto*'s suitability for use as an independent translation system, particularly in the context of an AAC application, where users can hardly be expected to sift through, let alone understand (the differences between), 40+ translation hypotheses. Therefore, if the pipeline is to be used as an independent translation system for real world applications, the next logical step should be to devise a means of selecting among the various translation hypotheses. That would almost certainly require the addition of a stochastic component, such as described in Velldal (2009). Alternatively, rather than being constituting an independent system, the *Depicto* pipeline could be integrated into an existing statistical machine translation system, such as the *Picto2Text* system

(Sevens et al. (2015); see Section 1.2.2.1), so as to form a hybrid system. The potential for such hybridization is discussed further in the next chapter.

Chapter 5

Winding down

5.1 Progress so far

As of May 2016, the basic three-part architecture of the *Depicto* system is in place and all module-specific extensions (described in Chapter 3 and Chapter 4) are in working order (insofar as their experimental status permits). This means that the system as a whole is able to take a string of *Sclera* symbol identifiers as input, parse this string for its semantic structure, modify, or 'transfer', the resulting 'MRS' so as to accommodate a given target language (in this case: Dutch), and, finally, as demonstrated in Section 4.2.2, use this MRS as the basis for generating one or more grammatical sentences as output.

Of course, only those input strings are accepted which are well-formed with respect to the constraint-based model of *Sclera* used by the analysis module (Chapter 3). Starting from a (purely hypothetical) sketch grammar of *Sclera* (Section 3.2.1), this model treats *Sclera* as a simple SVO language that is unique in the fact that it has 'invisible' determiners (Section 3.2.3), complex pictographs corresponding to phrasal constituents of varying saturation (Section 3.4.3), and 'particles' which actively modify the illocutionary force (Section 3.4.2) or temporal orientation (Section 3.4.1) of a *Sclera* utterance.

Currently, the model covers 30 *Sclera* symbols (see Appendix B), each of which has an equivalent in the next two stages in the processing chain, and accepts root structures (i.e., typed feature structures that can serve as the start symbol for parsing) that correspond to a single clause, although independent noun phrases and various elliptical structures, including discourse markers, will soon be supported.

Moving on to the other two modules in the system (Chapter 4), we first saw how a simple semantic (MRS-based) transfer grammar for the language pair *Sclera*-Dutch is set up (Section 4.1). In the actual course of development, this step formed something of a milestone, offering welcome reassurance that the grammar of *Sclera* (*Depicto*'s main contribution) could indeed stand independently of the rest of the pipeline, amenable to 'co-operation' with other third-party grammars by virtue of an additional transfer grammar. Since (to the best of my knowledge) the practicalities of setting up a transfer grammar are not documented in any great detail anywhere, special attention is paid to the process (Section 4.1.1) so that this section may additionally serve as a resource in the future. Finally, for the last module, a simple grammar of Dutch is developed (Section 4.2) which, in addition to its basic feature geometry, phrase structure rules, and lexicon, features two temporary loci of experimentation involving determiners (Section 4.2.1.1) and synonymy (Section 4.2.1.2), the former eventually to be relocated to another stage of the pipeline so as to maintain the target language grammar's independence. As discussed in Section 4.2.2, the current version of the *Depicto* system does not discriminate among the various translation hypotheses produced by the generation module. As a result, it is still some way away from being ready for use in real-world (in this case: AAC) applications.

5.2 Evaluation

In describing the development of *Depicto*'s individual modules, Chapter 3 and Chapter 4 also provide critical reflection on the appropriateness of the design decisions taken, particularly when the proposed solution has the potential of posing a limitation to the module currently under development. This section takes up a similarly critical tone, but does so from a holistic standpoint, considering the system not in terms of its component parts, but as a whole. In Section 5.2.1, the system is evaluated with respect to its ability to produce quality output ('precision'), its coverage over possible input strings, and its overall performance in terms of speed. Because *Depicto* is still in an early stage of development, its lexicon of *Sclera* still relatively minimalistic and its output unfiltered, metric-based methods for evaluating the system are not appropriate. Instead, the evaluation process is guided by simple intuition, i.e., human judgement. In Section 5.2.2, the *Depicto* system is pitted against the *Picto2Text* system developed by Sevens et al. (2015) (introduced in Section 1.2.2.1). This (largely playful) comparison serves to illustrate both the strengths and limitations of *Depicto*'s rule-based architecture in contrast to the (at heart) statistical approach taken by *Picto2Text.* The results are largely in keeping with the traditional trade-offs between Rule-Based Machine Translation (RBMT) systems and Statistical Machine Translation (SMT) systems, but instructive nevertheless.

5.2.1 Precision, coverage & performance

5.2.1.1 Precision

Semantically, syntactically, morphologically, and – in certain respects – lexically, the *Depicto* pipeline can be said to produce natural language translations of very high quality. As demonstrated in Section 4.2.2, these are composed of semantically relevant lexical items, which are arranged in appropriate order and inflected in accordance with agreement or case requirements, and, more important, consistently correspond to the full set of all possible readings of the input *Sclera* string, which, as we saw earlier, the pipeline is able to narrow down if markers of, e.g., mood or tense are present on the input.

Orthographically, output strings are of adequate quality: spelling is not an issue, and the first character of output strings is automatically capitalized. However, the absence of a theory of punctuation in the grammar of Dutch used here for generation means that the conventions of the target language's orthographic system are only partially respected. Slightly less trivial (for the present purposes, at least) is that, while the target language grammar can be fitted out for synonymy (see Section 4.2.1.2), it currently cannot account for how different synonyms enter into different collocation patterns, which results in some target language strings sounding oddly unidiomatic.

Similarly, on a pragmatic level, not all strings produced by *Depicto* seem as felicitous as the next, yet as far as the target grammar is concerned they are all equal. Idiomaticity and pragmatics are common sources of difficulty in rule-based translation systems (Chan 2014), and for the coverage of either phenomena the pipeline is largely at the mercy of the machinery provided by the target grammar, which, as explained, could in principle be any DELPH-IN or MRS-compatible grammar. For the time being, therefore, we can only put this down as a limitation, although, given the 'deep', i.e., semantic, approach of the *Depicto* system (or indeed, all LOGON-based MT systems), there is room to imagine a grammar in which, if not a system of pragmatics, a system of collocation is worked out.

Of course, the most obvious limitation as regards the quality of the *Depicto* system's output is that there is simply too much of it: it is an unfiltered, unsorted list rather than a single translation. One could argue that this is a feature of the system, in which case the output should be handled by either an entirely different, possibly third-party, tool or an additional module. In any case, until a solution is found, *Depicto* does not amount to a true translation tool in the sense that users are not required to be familiar with *both* the source and the target languages in order to use it successfully. We will briefly look at two possible solutions in Section 5.4.

5.2.1.2 Coverage

The *Depicto* system's coverage, i.e., the set of input strings which can be parsed, transferred, and generated from, is primarily determined by the model of *Sclera* used by the analysis module (Section 3.2), although, of course, that is assuming that the *Sclera* grammar has equivalents in the grammars used by the transfer and generation modules, as is the case in the version of the *Depicto* system developed here.

Currently, the coverage of the *Depicto* pipeline is limited in (at least) three ways by the grammar of *Sclera*.

In the first place, its lexicon covers only 30 of the (in total) 13,000 symbols comprised by the *Sclera* set. The main reason for this limitation is that the lexicon, which has been crafted by hand, is costly to extend, especially since each new addition to the grammar of *Sclera* requires corresponding additions to the transfer grammar and, in this particular case, to the target grammar as well (since the grammar of Dutch is developed 'in house'). Because *Sclera* is a closed symbol set, however, the lexical coverage of the grammar can in theory be made complete, and large parts of this objective could be realized fairly inexpensively once the lexical modelling process becomes automated.

In the second place, the grammar of *Sclera* is specified for a fairly strict set of syntagmatic restrictions which stem from its origins as a Subject-Verb-Object grammar. As a result, even if the lexicon is extended to include all *Sclera* symbols, so that any symbol is covered by the grammar, the analysis module fails to parse the input if it does not adhere to SVO ordering, which, in this case, has side-effects for adjective-noun and verb-adverb/adjunct ordering. Given that the *Depicto* system is primarily designed for users suffering from aphasia, dysphasia, or otherwise exhibiting agrammatism, this restriction is – for want of a better word – nothing short of silly. At the same time, however, because *Sclera* does not have a case marking system (which, again, would run counter to the requirements of the target audience, anyway), total freedom of word order would open the door to syntactic ambiguity: 'dog' in the utterance ‘dog see bus’ could be parsed either as the subject of the verb or as the object of the verb, and both parses would be equally valid. Needless to say, this could lead to some fairly serious confusion in interpersonal communication, as, for example, with ‘I hate you’ versus ‘You hate me’, which, communicatively, are clearly two very different beasts. Eventually, a compromise will have to be devised between the usability of the pipeline for the target audience and the risk of potentially serious confusion. Pending an adequate solution, the main SVO word order restriction is maintained, but noun-adjective and verb-adverb order restrictions are soon to be relaxed.

The third way in which the coverage of the *Depicto* system is constrained by the grammar of *Sclera* falls into two parts. First, the grammar only accepts complete clauses or noun phrases as the start symbol for parsing. To support a more fragmentary style of communication, this restriction will eventually be dropped, so that *any* phrase or lexical item can be parsed, the effect of which will be that the analysis module never blocks, i.e., returns an empty parse. Of course, whether the result of this more permissive style of parsing can be used for generation will ultimately depend on the start symbol constraints of the target language grammar. Second, the current grammar is designed to model discrete utterances, not discourse. When dealing with clausal input (as in most of the examples in this thesis), it requires the input to be a single, well-demarcated clause, which, in practice, means that multiple clauses equals multiple calls to the pipeline. It is common form for simple constraint-based grammars to focus on parsing individual sentences rather than stretches of text/discourse; however, in the context of machine translation, this restriction implies the assumption that one is either translating from some source text that can be pre-processed into individual clauses (on the basis of, say, punctuation) or that users have the ability to identify and enter clauses individually. Since the translation system is intended for real-time use and punctuation symbols (which are not popular among target users anyway) have been repurposed (e.g., the question mark symbol in Section 3.4.2), the first option is not relevant. Nor is the second: it stands to reason that if a user is able to identify a discrete clause correctly, his/her use for the *Depicto* pipeline might be limited. Given the invalidity of this assumption, the grammar of *Sclera* should eventually be extended to cover larger stretches of discourse. (Though not documented anywhere, the ERG (Flickinger et al. 2014) already appears to do this for English, so a solution to this last restriction might be relatively easy to implement.)

5.2.1.3 Performance

Whether or not the input string falls within the coverage of the system, however, the pipeline itself consistently achieves fast processing times. Initial (and largely informal) tests performed on an Apple Macbook Pro with a 2.6 GHz Intel Core i5 processor and 8 GB of memory suggest that, when passed an 'ill-formed' *Sclera* string or when the target grammar fails to generate, the pipeline blocks within 10ms to 20ms. When all stages of the pipeline execute successfully, total processing times tend to range from 20ms to just under 100ms, the bulk of which is spent on generation. The more results the generation module can produce, the more time is required. Thus, an artificially complex (yet well-formed and thus plausible) input string such as `happy dog give happy girl happy kiss yesterday question`, which corresponds to a staggering 768 (!) translationally equivalent Dutch sentences, takes an average of 250ms longer to process. Using this string, (5.1) illustrates how these times have been collected. The shell script in this example is identical to that used to test the pipeline (see Appendix A), except that (ignoring the first line) it is prefixed by the UNIX `time` utility, which measures the total time taken for a command or series of commands to execute. Of the the three times measured by this utility (i.e., `real`, `user`, and `system`), all data points are based on the value of the `real` measurement, which tends to be the least charitable.

(5.1)

```
  cd PATH/TO/ACE/BINARY/IN/REPOSITORY/DIRECTORY
  time echo "happy dog give happy girl happy kiss yesterday question" |
# ^^^                   ^^^                         ^
# UNIX time utility    string to be parsed       Pipe
      ./ace_osx -g ../mini-sclera.dat |
#                ^^^^^^^^             ^
#             Analysis module        Pipe
      ./ace_osx -g ../transfer-nl.dat |
#                ^^^^^^^^             ^
#          Transfer module           Pipe
      ./ace_osx -g ../mini-dutch.dat -e
#                ^^^^^^^^             ^
#         Generation module          -e flag sets ACE to generate
```

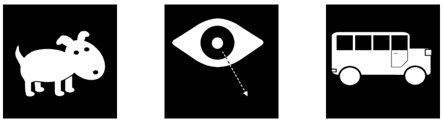
While generation is certainly the most costly stage in the the pipeline, the ACE processor (to which, needless to say, *Depicto*'s processing times owe a great debt) provides several optimizations, some by default, others optionally, that speed the process up. For example, when enabled, the `index-accessibility-filtering` optimization reduces total processing times by an average of 30%. This is already reflected in the measurements reported above. Of course, no evaluation of performance is truly complete without some comparison to existing benchmarks. Unfortunately, I am not adequately familiar with these at present. For now, therefore, it will have to suffice to note that, when passed 'well-formed' input, the *Depicto* pipeline is capable of producing (if necessary) a *lot* of high quality output in a period of time which is more or less experienced as an instant, but the question of whether this 'instant' is more instant than the benchmark speeds of other (kinds of) translation systems is left blank for now.

5.2.2 Comparing the *Depicto* and *Picto2Text* systems

Let us now take a quick look at how *Depicto* fares against a similarly purposed translation system that takes a statistical rather than rule-based approach. A suitable candidate for this purpose is the promising *Picto2Text* system developed by Sevens et al. (2015) at the KULeuven. As explained in Section 1.2.2.1, the current version of the system draws on a trigram language model of the target language (plus Viterbi decoding) to select the most likely surface form permutations of (reverse lemmatized) word tokens derived from those WordNet synonym sets which are associated with selected pictographs. This approach has the advantage of increasing the coverage of the translation system at a relatively low cost. At the same time, however, the quality of the output is only as good as the three-item horizon of the trigram model can account for, and the model, in turn, is only as good as the corpus upon which it has been trained. So while *Picto2Text*'s coverage (or *recall*) is comparatively high, its precision is prone to ceiling effects and its output less easy to predict. This situation is of course the inverse of the *Depicto* system, where coverage is low (and relatively expensive to extend), but precision is high and entirely predictable based on the constraints of the grammar model. The effect of these opposing traits can be observed when both systems are passed the same input string. (All interactions with *Picto2Text* happen through the online demo available at `picto.ccl.kuleuven.be/DemoP2T.html`. Note that this demo may not represent the most up-to-date version of the *Picto2Text* system.)

Consider the output of translation by each system for the (by now much-loved) *Sclera* string 'dog see bus', as listed in (5.2). (Notice that this string, like most that follow, is already known to fall within *Depicto*'s coverage, which does bias the comparison to an extent; I will make up for this toward the end.) As expected, *Depicto*'s output comprises multiple translation hypotheses, three of which are shown. *Picto2Text* outputs only the most likely translation hypothesis, which in this case is the (100% grammatical) Dutch sentence *De hond ziet een bus* ('The dog sees a bus').

(5.2) Translating 'dog see bus' into Dutch:



a. *Depicto* (3 of 32):

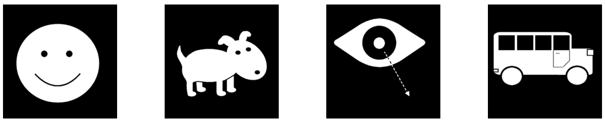
- `De hond ziet een bus` (‘The dog sees a bus’)
- `Een hond zag de bus` (‘The dog sees a bus’)
- `Honden zien bussen` (‘Dogs see buses’)

b. *Picto2Text*:

- `De hond ziet een bus` (‘The dog sees a bus’)

The output of *Picto2Text* in (5.2) is already fairly impressive, since it suggests that somewhere in the training corpus a trigram occurs which consists (in addition to a third element) of the token *hond* (‘dog’) combined with the correctly conjugated form of ‘barking’, i.e., *blaft.* When ‘happy’ is added to the start of the *Sclera* string, however, as in (5.3), the result is somewhat different. Here, in contrast to *Depicto*, whose output shows the adjective *blij*(+e) inflecting so as to agree with the gender and/or number of the modified noun *bus*, *Picto2Text* produces an ungrammatical result: the synonymous adjective *gelukkig* is not inflected. The inclusion of this extra token on the input appears to have an obscuring effect on the trigram model, which now fails to find both the correct third person singular conjugation of the verb ‘see’ (in this case *ziet*) and a determiner for the count noun *bus* (although this is technically handled by a separate part of the system). The reason for this sudden drop in quality is most likely due to the absence of ‘happy dog’ combinations in the training corpus, as well as to the general ‘oddness’ of the input string itself, for which a generative approach can account, but a statistical approach, which by definition favors conventional language use, cannot.

(5.3) Translating ‘happy dog see bus’ into Dutch:



a. *Depicto* (3 of 32):

- `De blije hond zag de bus` (‘The happy dog saw the bus’)
- `Blije honden zien de bussen` (‘Happy dogs see the buses’)

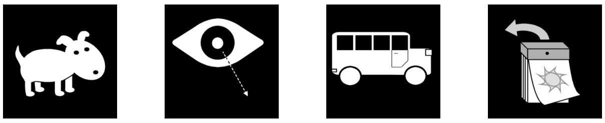
- `Honden zien bussen` (‘A happy dog sees the buses’)

b. *Picto2Text*:

- `*Gelukkig hond zien bus` (‘*Happy dog *see bus’)

A slightly improved situation can be observed in (5.4). Here, the output of *Picto2Text* is grammatical with respect to determination and conjugation, but presents the present tense form of the verb ‘see’ as the most likely candidate, in spite of the presence of the temporal adjunct ‘yesterday’. (Again, the comparison here is very much biased toward *Depicto*, since we have already established that *Depicto* contains a set of rules to deal precisely with the phenomenon of temporal adjuncts – that is, provided the encountered temporal adjunct is ‘yesterday’ and nothing else.)

(5.4) Translating ‘dog see bus yesterday’ into Dutch:



a. *Depicto* (3 of 48):

- `De hond zag de bus gisteren` (‘The dog saw the bus yesterday’)
- `De honden zagen gisteren de bus` (‘The dogs saw the bus yesterday’)
- `Een hond zag gisteren de bus` (‘A dog saw the bus yesterday’)

b. *Picto2Text*:

- `?de hond ziet een bus gisteren` (‘Happy dog ?sees a bus yesterday’)

As we saw in Section 3.4.3, *Depicto*’s analysis module contains, as it were, specific instructions on how to deal with complex pictographs. The result is a fair degree of regularity among the various translation hypotheses produced, as shown in (5.5) for ‘dog+bark’. The output of *Picto2Text*, by contrast, is less easy to predict – indeed, surprising: according to the system, the most likely translation of 'dog+bark' is the modified noun phrase 'barking dogs', which, from a strictly communicative perspective, seems like a strange contribution to make in a conversation (although, of course, that observation is beside the point in the context of *Picto2Text*'s translation strategy).

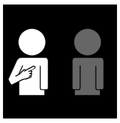
(5.5) Translating 'dog+bark' into Dutch:

a. *Depicto* (3 of 8):
   - `De hond blaft` ('The dog barks')
   - `Een hond blafte` ('A dog barked')
   - `Honden blaffen` ('Dogs bark')

b. *Picto2Text*:
   - `Blaffende honden` ('Barking dogs')

However, when the first person singular possessive pronoun 'my' is added to the input string, so that we get 'my dog+bark', *Picto2Text*'s output, as shown in (5.6), is very different. In this particular case, the most likely translation according to the language model is the clause corresponding to 'my dog barks'. As far as we are concerned, this is a perfectly acceptable translation: it is grammatical and its content makes a communicative contribution which, informationally, seems plausible. It is interesting to note, moreover, that in this example *Picto2Text* produces output, whereas *Depicto* does not: possessive pronouns are currently not covered by the model of *Sclera*, so the analysis module blocks.

(5.6) Translating 'my dog+bark' into Dutch:



a. *Depicto* (0 of 0):

- $\emptyset$

b. *Picto2Text*:

- `Mijn hond blaft` ('My dog barks')

This illustrates a key point of comparison between the two systems. Whatever the input, *Picto2Text* will produce output, even if this output is not of the highest natural linguistic quality. *Depicto*, by contrast, is picky about its input, and, even when it accepts it, can fail in any of the two subsequent stages, depending on how well they are integrated. Yet, when *Depicto* succeeds, its output is consistent (i.e., predictable) and grammatical. *Picto2tText* is robust, but offers no guarantees about the precision of its output; *Depicto* is geared toward precision, but is magnificently prone to failure. Thus, the two systems are each other's inverse – a point to which we return in Section 5.4.

5.3 Conclusions

The main objective of the work presented in this thesis is to design and implement the basic framework for an assistive communication tool that enables dysphasic users to compose primarily written natural language utterances based on pictographic symbols. The proposed translation system, *Depicto*, contrasts with alternatives (e.g., the *Sanyog*, *PVI*, and *Picto2Text* systems; see Section 1.2) in that its approach is 100% rule-based. This goes against the grain of more popular machine translation methodologies, which tend to favor data-driven or hybrid strategies, but is arguably appropriate in this particular case, given (a) the lack of resources for language pairs involving pictographic languages, and (b) the semantically underspecified and 'underquantified' character of pictographic languages in general. Targeting the *Sclera* symbol set in particular, the *Depicto* system shows that such languages are amenable to 'deep parsing' (i.e., semantic analysis) in a way that incorporates linguistic intuition (captured as rules/constraints) to yield consistent and reliable analyses which are semantically ‘richer’ than the input sequence itself, yet in a way that does not require structured input from the user (cf. the *Sanyog* system and its Query-Response input model; Section 1.2). The result of deep parsing (the logical form of the input string) is translated by a set of bilingual *transfer rules* and used as the basis for target language generation, which itself is determined by another constraint-based grammar resource. Because the current version of the system does not filter or rank its output, the result of generation is a *set* of high-quality translations of the pictographic input sequence. Insofar as the translation system is functional, part of this first objective can be said to have been achieved (the limitations discussed in Section 5.2.1 notwithstanding). A few important nuances are returned to in the third paragraph.

On a macro-level, the *Depicto* system requires a total of three resources, each of which is fairly costly to develop (as one finds out when one develops them from scratch), in addition to the open-source processor ACE, by means of which these resources are operationalized (Section 2.2.4). A second, more implicit, objective of the work presented here, therefore, has simply been to feel out the feasibility of this rule-based approach, especially with respect to the extensibility of the system to other target languages. To this end, the criterion is set that the system follow a modular design, the idea being that the grammar used by the analysis module can be developed independently of other parts of the grammar so that, in principle, several people can work on it at once, that is, collaboratively, even if the target language to which they each individually wish to translate is different. To keep costs down, the development of the target grammar should ideally not fall within the scope of the *Depicto* project, but be left to other developers instead; indeed, this is one of the great advantages of the reusability of DELPH-IN grammars. Once the grammar of *Sclera* reaches an adequate level of coverage (hypothetically, at least), developers should be able to extend the system to a new language (i.e., a different grammar) without requiring too much familiarity with the grammar of *Sclera* itself, except for its lexicon. In this case, the only remaining work is situated on the level of the transfer grammar. (As suggested by Bond et al.

(2011), much of the work required here can be automated.) In light of this goal, all parts of the version of *Depicto* presented here have been designed to adhere to the principle of modularity – with one important exception, however, which is that the relation between underspecified and concrete determiners is currently handled by the target language grammar, whose type hierarchy contains predicate types specifically for this purpose. If modularity is to be maintained, therefore, an alternative solution needs to be devised. This will probably involve the transfer stage, although it may be possible to relocate it even further 'back' in the pipeline, i.e., to the grammar of *Sclera* itself. Crysmann & Packard (2012) explain that the ACE processor allows for so-called *post-parsing rewrite rules*, which are essentially rewrite rules comparable to those found in the transfer grammar, except they are included with the parsing grammar and executed, as the term suggests, as soon as the parser has finished. A definitive solution is deferred to future work. Of course, it should be borne in mind that, while modularity certainly improves development costs, particularly if the system garners the interest of others, these costs remain high in comparison with statistical machine translation systems. For example, progress on the grammar of *Sclera* will have to be measured in weeks and months rather than days. Whether this is an adequate trade off for the relatively high translation quality remains to be seen.

Finally, let us return to the second criterion by which *Depicto*'s design is guided. Fairly obvious yet essential nonetheless, this criterion requires that the system stay true to the needs of its target users, who are assumed to be people with an intellectual (or developmental) disorder that prevents them from being fully able to produce natural language utterances of an adequately intelligible or grammatical level. As (part of) a hypothetical alternative communication tool, the *Depicto* system can be said to meet this criterion insofar as it is able to accurately translate from pictographic input and thus provide users with a means of formulating utterances in terms of those pictographic symbols which are accepted by the system. However, as explained in Section 5.2.1, the current system's analysis module makes a number of fairly restrictive assumptions about word order, the motivation for which is explained in more detail above, but boils down to an ad hoc measure to prevent excessive ambiguity. At present, in the absence of real-life *data* concerning the use of *Sclera* by the intended user group, it is difficult to say how great the impact of the SVO word order restriction would be on the usability of the system by the average user. The question arises whether some users *are* in fact aware of basic word order patterns, or could be taught to think in terms of one specific pattern. Of course, even if the answers to these questions come up positive, it would still imply that another group of users is excluded from using the system, which is an unfortunate possibility for a tool whose primary aim is to *improve* inclusion. With regard to this second criterion, therefore, there is still progress to be made. Closer interaction with the users themselves, e.g., through user testing or simple observation, as well as more in-depth research into the syntactic 'symptoms' of dysphasia and other communication disabilities, may serve as an interesting guideline in this respect.

5.4 Future work

In addition to devising near-term solutions to those design decisions that currently compromise *Depicto*'s criteria of modularity and usability (supra), future work will concentrate on three main areas.

The first involves the introduction of script-based automation in the modelling process. The result in the long run should translate to an overall reduction in the labor-intensitivity of the modelling process, which, of course, goes hand in hand with a reduction in development time (once an adequate modelling script has been devised, that is). The most immediate headway can probably be made at the level of the lexicon of the grammar used by the analysis module, so we will start there. However, the aim should be to extend script-based automation to as many parts of the grammar as possible, as well as to the set of transfer rules used by the next stage in the system. (Modelling the target grammar, by contrast, is assumed to be the responsibility of a different set of developers.)

In order to test the idea of modularity, the second area of work involves switching out the currently used 'toy' grammar of Dutch for more extensive (and mature) grammars of, first, Dutch and, later, other languages, such as English, German, and Japanese (for which DELPH-IN-based grammars are readily available; Flickinger (2000); Müller & Kasper (2000); Siegel & Bender (2002)). In Section 3.2, I note that no established DELPH-IN grammar of Dutch is available at the time of writing. This is only partly true. Fokkens (2011) includes a grammar of Dutch as part of the German branch of the CLIMB metagrammar engineering project. Hitherto unmentioned, the CLIMB project is an offshoot of the Matrix grammar customization system that builds further on the concept of script-based grammar generation based on a set of parameters. (The core type hierarchy is the same as in other Matrix grammars, however.) The CLIMB source code is publicly available, as are the parameters (which contribute an extensive lexicon) required for generation of a grammar of Dutch. Unfortunately, despite several attempts, I was not able to get the CLIMB system to generate a functioning grammar. This was fairly early on in the development of the *Depicto* system, however; as I became more familiar with the intricacies of the DELPH-IN environment (that is, while I was already working on my own grammar of Dutch), I came more and more to suspect that this failure was most likely due to error on my part. The first step in this stage, therefore, will be to revisit the CLIMB grammar of Dutch, not least because this will require most likely require minimal modifications to the existing transfer grammar. The next step will involve the extension of *Depicto* to English. For this we will draw upon the English Resource Grammar (Flickinger 2000) (the flagship grammar of the DELPH-IN project) as well as set up a new transfer grammar.

The third and (by far) most crucial area of future work, however, will concern the implementation of a method for filtering the output of the target language generation module down to a single most likely surface realization, which will bring the behavior of the *Depicto* system in line with that generally expected of a machine translation system. Currently, there are at least *two* directions in which we can proceed. As the 'likely' in the sentence above might lead one to expect, both involve some degree of hybridization with probabilistic strategies, although the approaches themselves do differ.

The first draws further upon the LOGON MT infrastructure, which, at the generation end, incorporates a data-driven *realization ranking* component (Velldal 2009; Velldal & Oepen 2006; Bond et al. 2011). This component functions to rank surface realizations (and their underlying syntactic trees) in order of likeliness as determined by a discriminative log-linear model trained on an annotated treebank of the target language (Velldal & Oepen 2006). The highest ranking surface realization forms the output of translation. This approach is certainly interesting, and should definitely be explored. Indeed, its developers claim that it represents a great improvement over 'traditional' n-gram approaches (more about which in a moment) (Velldal 2009). Its downsides are that compatible treebanks are few and far between. There is the LinGO Redwoods treebank for English (Oepen et al. 2002), but equally extensive alternatives for other languages appear do note appear to exist. At least one *non*-DELPH-IN treebank exists for Dutch (see `http://nederbooms.ccl.kuleuven.be/eng/`), but my understanding of treebanking is too limited to conjecture if its annotations and, most important, syntactic analyses could be transposed to a DELPH-IN-compatible format. It is currently also unclear whether the ACE processor supports realization ranking or whether some additional processor will be required. The answers to these questions, which will be explored in due time, will ultimately determine the practicability of the treebanking approach.

The second approach, by contrast, could in principle yield a prototype fairly quickly. Under this approach, the *Depicto* pipeline is incorporated into Sevens et al. (2015)'s *Picto2Text* system (see Section 1.2.2.1 and Section 5.2.2). The idea is as follows. All input is sent to *Depicto* first. If the pipeline succeeds, its output is passed to the Viterbi decoder, the least unlikely string according to which (based on the appropriate trigram language model) is selected as the output of translation. If the pipeline fails, the system simply falls back to *Picto2Text*'s original infrastructure. Alternatively, the two systems are run in parallel and their output is compared by a second round of Viterbi decoding. Either way, this approach amplifies the (inverse) strengths of the two systems, while reducing the effect of their limitations. That is to say, for cases that are easily covered by a general set of rules, the system as a whole – 'dePicto2Text', if you will – is almost guaranteed to produce high-quality output. At the same time, less easy-to-cover cases will still translate to *something*, the quality of which, as we see in Section 5.2.2, *is* more variable but can also be just as high as that achieved by *Depicto*. Thus, the average precision of the modified *Picto2Text* increases, while the robustness of the original system stays exactly the same. An added bonus to this approach is that *Depicto* gets to piggyback on (and, if desired, contribute to) *Picto2Text*'s user interface. Besides the convenience of freeing up initial development time, this could ultimately translate to more progress on the interface in less time. Given all its advantages, the second approach is the most interesting. Of course, is is *only* a proposal. For now, therefore, we keep our options open.

Appendix A

Demoing *Depicto*

This appendix provides instructions for obtaining and using the version of the *Depicto* system developed in this thesis. This version takes a relatively small set of *Sclera* pictograph identifiers as input and produces well-formed **Dutch** sentences as output. For a complete list of currently accepted pictograph identifiers, please see Appendix B.

Requirements

The *Depicto* system supports 64-bit versions of both GNU/Linux and Mac OS X. It has been tested on Ubuntu (verions 15.10 and 16.04.1) and on Mac OS X 10.11.5 (El Capitan).

Setting up

The grammars used by the ACE processor need to be compiled from source. The source code of the individual grammars, as well as up-to-date ACE processor binaries (for each of the two supported OSs), a shell script wrapper, and some very preliminary documentation, is available as a GitHub repository.

Getting the code

To obtain the repository, **either** ...

1. Visit the main page of the repository at `https://github.com/lemontheme/depicto-dutch`,
2. Click **Clone or download** > **Download Zip**
3. Extract the Zip archive

or, if Git is installed (preferred method), ...

1. Open Terminal
2. Change the current working directory (`cd`) to the location where you want the repository to be downloaded,
3. Type:

   ```
   $ git clone https://github.com/lemontheme/depicto-dutch
   ```

4. Press **Enter**.

Compiling the grammars

The most important directories in the downloaded repository are `ace-app` and `grammars`. The `ace-app` directory contains OS-specific ACE binaries, shell script wrappers, and compiled grammar images (as `.dat` files). The `grammars` directory contains those files which constitute the 'code' (`.tdl` files) from which the individual grammars are compiled. The subdirectories under `grammars` are as follows:

- `mini-dutch` = Grammar of Dutch
- `mini-sclera` = Grammar of *Sclera*
- `matrix-main` = Resources shared by both `mini-dutch` and `mini-sclera`
- `transfer-nl` = Sclera-Dutch transfer grammar

To compile the grammars, follow the steps below.

1. In Terminal, navigate to the subdirectory of `ace-app/` whose name corresponds to that of the current operating system, e.g.:

   ```
   $ cd depicto-dutch/ace-app/osx #(on OS X)
   ```

   or

   ```
   $ cd depicto-dutch/ace-app/linux #(on Linux)
   ```

2. Run the `compile-all` shell script:

   ```
   $ sh compile-all_osx.sh
   ```

   or

   ```
   $ sh compile-all_linux.sh
   ```

   (`compile-all.sh` is a simple wrapper containing three individual compilation calls to ACE processor.)

Usage

For ease of testing, the *Depicto* demo includes a second wrapper that calls all three stages of the pipeline in order, invoking ACE plus an appropriate grammar image for each, and 'piping' the output of preceding stages forward. This wrapper, which is broken down in (5.1), represents only one of many ways in which the *Depicto* can be implemented. For instance, while it relies on interactive user input itself, a slightly modified script could just as easily read input from a test file, or from any other kind of input stream.

1. When starting a new session, open Terminal and `cd` to the subdirectory of `ace-app/` whose name corresponds to that of the current operating

system.

2. Initialize the *Depicto* pipeline, with...

   ```
   $ sh test-pipeline_osx.sh
   ```

   or

   ```
   $ sh test-pipeline_linux.sh
   ```

   This opens up new line below, from which *Depicto* (or the ACE processor, to be precise) waits to read its input.

3. Enter a sequence of pictograph identifiers, seperated by any number of spaces, and press **Enter**. For example:

   ```
   $ sh test-pipeline_linux.sh
   dog see bus yesterday #<Enter>
   ```

   If the input is covered by the system, this step results in a complete list of translation hypotheses, as described in Section 4.2.2.

   Whether successful or not, the pipeline returns a new user input line, so that this step can be repeated infinitely.

4. To exit the pipeline, type `<CTRL-C>`.

Appendix B

Sclera-grammar lexicon

Picto identifier	**Approx. PoS**	**Main relation**	**Other relation(s)**
I	(pro)**n**oun (nom/acc)	‘_pronoun_n_rel’	
you_sg	(pro)**n**oun (nom/acc)	‘_pronoun_n_rel’	
you_pl	(pro)**n**oun (nom/acc)	‘_pronoun_n_rel’	
he	(pro)**n**oun (nom/acc)	‘_pronoun_n_rel’	
she	(pro)**n**oun (nom/acc)	‘_pronoun_n_rel’	
we	(pro)**n**oun (nom/acc)	‘_pronoun_n_rel’	
they	(pro)**n**oun (nom/acc)	‘_pronoun_n_rel’	
dog	count **n**oun	‘_dog_n_rel’	
girl	count **n**oun	‘_girl_n_rel’	
bus	count **n**oun	‘_bus_n_rel’	
kiss	count **n**oun	‘_kiss_n_rel’	
love	mass **n**oun	‘_love_n_rel’	
couch	count **n**oun	‘_couch_n_rel’	
headphones	plural **n**oun	‘_headphones_n_rel’	
happy	ad**j**ective	‘_happy_j_rel’	
brown	ad**j**ective	‘_brown_j_rel’	
yesterday	temp. adve**r**b	‘_yesterday_r_rel’	
sleep	**v**erb	‘_sleep_v_rel’	
bark	**v**erb	‘_bark_v_rel’	
buy	**v**erb	‘_buy_v_rel’	
see	**v**erb	‘_see_v_rel’	
give	**v**erb	‘_give_v_rel’	
brush	**v**erb	‘_brush_v_rel’	
dog_bark	complex-verb	‘_bark_v_rel’	‘_dog_n_rel’, ‘q_rel_min’
brush_dog	complex-verb	‘_brush_v_rel’	‘_dog_n_rel’, ‘q_rel_min’
go_school	complex-verb	‘_go_v_rel’	‘locative_rel’, ‘_school_v_rel’, ‘exist_q_rel’
question	illoc. ‘operator’	(none)	

Appendix C

Example test sentences

dog see bus

dog see bus yesterday

dog see bus yesterday question

brown dog see bus yesterday question

dog sleep

dog_bark

brown dog_bark

i buy headphones

i give you_sg headphones

girl go_school

she go_school yesterday

happy dog_bark

happy girl buy brown dog

i brown brush_dog

i brown brush_dog yesterday

you_sg give i kiss question

they see i yesterday

i see you_pl yesterday

Bibliography

Augustinus, Liesbeth. 2015. Complement raising and cluster formation in Dutch. a treebank-supported investigation .

Bender, Emily M. 2007. Combining research and pedagogy in the development of a crosslinguistic grammar resource. In *Proceedings of the workshop grammar engineering across frameworks*, .

Bender, Emily M., Scott Drellishak, Antske Fokkens, Michael Wayne Goodman, Daniel P. Mills, Laurie Poulson & Safiyyah Saleem. 2010. Grammar prototyping and testing with the lingo grammar matrix customization system. In *Proceedings of the acl 2010 system demonstrations*, 1–6. Uppsala, Sweden: Association for Computational Linguistics. `http://www.aclweb.org/anthology/P10-4001`.

Bender, Emily M & Dan Flickinger. 2005. Rapid prototyping of scalable grammars: Towards modularity in extensions to a language-independent core. In *Proceedings of the 2nd international joint conference on natural language processing ijcnlp-05 (posters/demos), jeju island, korea*, .

Bender, Emily M., Dan Flickinger & Stephan Oepen. 2002. The Grammar Matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In John Carroll, Nelleke Oostdijk & Richard Sutcliffe (eds.), *Proceedings of the workshop on grammar engineering and evaluation at the 19th international conference on computational linguistics*, 8–14. Taipei, Taiwan.

Bhattacharya, Samit & Anupam Basu. 2009. Design of an iconic communication aid for individuals in india with speech and motion impairments. *Assistive Technology* 21(4). 173–187.

Boas, Hans Christian & Ivan A Sag. 2012. *Sign-based construction grammar.* CSLI Publications/Center for the Study of Language and Information.

Bond, Francis, Stephan Oepen, Eric Nichols, Dan Flickinger, Erik Velldal & Petter Haugereid. 2011. Deep open-source machine translation. *Machine Translation* 25(2). 87–105.

Bond, Francis, Stephan Oepen, Melanie Siegel, Ann Copestake & Dan Flickinger. 2005. Open source machine translation with DELPH-IN. In *Proceedings of the open-source machine translation workshop at the 10th machine translation summit*, 15–22.

Carpenter, Bob. 1992. *The Logic of Typed Feature Structures*. Cambridge: Cambridge University Press.

Carroll, John, Ann Copestake, Dan Flickinger & Victor Poznanski. 1999. An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of the 7th european workshop on natural language generation (ewnlg'99)*, 86–95.

Carroll, John & Stephan Oepen. 2005. High efficiency realization for a wide-coverage unification grammar. In *Natural language processing–ijcnlp 2005*, 165–176. Springer.

Chan, Sin-wai. 2014. *Routledge encyclopedia of translation technology*. Routledge.

Chomsky, Noam. 1959. On certain formal properties of grammars. *Information and control* 2(2). 137–167.

Copestake, Ann. 1997. Augmented and alternative NLP techniques for augmentative and alternative communication. In *In proceedings of the ACL workshop on natural language processing for communication aids*, 37–42.

Copestake, Ann. 2000. Appendix: Definitions of typed feature structures. *Natural Language Engineering* 6(01). 109–112.

Copestake, Ann. 2002. *Implementing Typed Feature Structure Grammars*, vol. 110. CSLI publications Stanford.

Copestake, Ann & Dan Flickinger. 2000. An open source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the second international conference on language resources and evaluation (lrec-2000)*, Athens, Greece: European Language Resources Association (ELRA). `http://www.lrec-conf.org/proceedings/lrec2000/pdf/371.pdf`. ACL Anthology Identifier: L00-1276.

Copestake, Ann, Dan Flickinger, Carl Pollard & Ivan A Sag. 2005. Minimal Recursion Semantics: An Introduction. *Research on Language and Computation* 3(2-3). 281–332.

Crysmann, Berthold & Woodley Packard. 2012. Towards efficient HPSG generation for German, a non-configurational language. In *Coling*, 695–710.

Flickinger, Dan. 2000. On building a more effcient grammar by exploiting types. *Natural Language Engineering* 6(01). 15–28.

Flickinger, Dan & Emily M. Bender. 2003. Compositional semantics in a multilingual grammar resource. In Emily M. Bender, Dan Flickinger, Frederik Fouvry & Melanie Siegel (eds.), *Proceedings of the workshop on ideas and strategies for multilingual grammar development, esslli 2003*, 33–42. Vienna, Austria.

Flickinger, Dan, Emily M Bender & Stephan Oepen. 2003. MRS in the LinGO Grammar Matrix: A practical user's guide.

Flickinger, Dan, Emily M Bender & Stephan Oepen. 2014. Towards an encyclopedia of compositional semantics: Documenting the interface of the English resource grammar. In *Lrec*, 875–881.

Fokkens, Antske. 2011. Metagrammar engineering: Towards systematic exploration of implemented grammars. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, 1066–1076. Association for Computational Linguistics.

Fokkens, Antske Sibelle. 2014. *Enhancing empirical research for linguistically motivated precision grammars*: Department of Computational Linguistics, Universität des Saarlandes dissertation.

Haeseryn, W. et al. 1997. *Algemene nederlandse spraakkunst 2*. Groningen: Martinus Nijhoff.

Joshi, Aravind K & Yves Schabes. 1997. Tree-Adjoining Grammars. In *Handbook of formal languages*, 69–123. Springer.

Keskinen, Tuuli, Tomi Heimonen, Markku Turunen, Juha-Pekka Rajaniemi & Sami Kauppinen. 2012. SymbolChat: A flexible picture-based communication platform for users with intellectual disabilities. *Interacting with Computers* 24(5). 374–386.

Krieger, Hans-Ulrich & Ulrich Schäfer. 1994. TDL: a type description language for constraint-based grammars. In *Proceedings of the 15th conference on computational linguistics-volume 2*, 893–899. Association for Computational Linguistics.

Lønning, Jan Tore, Stephan Oepen, Dorothee Beermann, Lars Hellan, John Carroll, Helge Dyvik, Dan Flickinger, Janne Bondi Johannessen & Paul Meurer. 2004. LOGON. a Norwegian MT effort. In *In proceedings of the workshop in recent advances in scandinavian machine translation*, .

Miller, George A. 1995. WordNet: a lexical database for English. *Communications of the ACM* 38(11). 39–41.

Müller, Stefan, Freie Universität Berlin, Deutsche Grammatik & Habelschwerdter Allee. 2013. HPSG – a synopsis to appear in artemis alexiadou and tibor kiss (eds): Syntax – ein internationales handbuch .

Müller, Stefan & Walter Kasper. 2000. HPSG analysis of German. In *Verbmobil: Foundations of speech-to-speech translation*, 238–253. Springer.

Oepen, Stephan. 2008. The transfer formalism: general purpose MRS rewriting. Tech. rep. Technical Report LOGON Project. University of Oslo. http://www.emmtee.net/reports/11.pdf.

Oepen, Stephan. 2010. Efficient parsing and disambiguation for DELPH-IN grammars. Tech. rep.

Oepen, Stephan, Dan Flickinger, Kristina Toutanova & Christoper D. Manning. 2002. Lingo redwoods - a rich and dynamic treebank for HPSG. In *In beyond parseval. workshop of the third lrec conference*, 575–596.

Packard, Woodley. 2015. ACE, the Answer Constraint Engine. [Last accessed 2016-06-25]. http://sweaglesw.org/linguistics/ace/.

Pollard, Carl & Ivan A Sag. 1994. *Head-driven Phrase Structure Grammar*. University of Chicago Press.

Postma, Marten & Piek Vossen. 2014. Open Source Dutch WordNet .

Sag, Ivan A., Thomas Wasow & Emily Bender. 2003. *Syntactic Theory: A Formal Introduction*. Stanford: Center for the Study of Language and Information 2nd edn. http://csli-publications.stanford.edu/site/1575864002.html.

Sag, Ivan A, Thomas Wasow, Emily M Bender & Ivan A Sag. 1999. *Syntactic theory: A formal introduction*, vol. 92. Center for the Study of Language and Information Stanford, CA.

Sclera NPO. 2012. Voorstelling van de werken. Brochure. http://www.sclera.be/resources/img/vzw/Voorstelling%202012.pdf.

Sevens, Leen, Vincent Vandeghinste, Ineke Schuurman & Frank Van Eynde. 2015. Natural language generation from pictographs. In *Proceedings of the 15th european workshop on natural language generation (enlg)*, 71–75. Association for Computational Linguistics.

Shieber, Stuart Merrill, Hans Uszkoreit, Fernando Pereira, Jane Robinson &

Mabry Tyson. 1983. The formalism and implementation of PATR-II .

Siegel, Melanie & Emily M Bender. 2002. Efficient deep processing of Japanese. In *Proceedings of the 3rd workshop on asian language resources and international standardization-volume 12*, 1–8. Association for Computational Linguistics.

Steele, Richard D, Michael Weinrich, Robert T Wertz, Maria K Kleczewska & Gloria S Carlson. 1989. Computer-based visual communication in aphasia. *Neuropsychologia* 27(4). 409–426.

Vaillant, Pascal. 1998. Interpretation of iconic utterances based on contents representation: Semantic analysis in the pvi system. *Natural Language Engineering* 4(01). 17–40.

Van Eynde, Frank. 1998. The immediate dominance schemata of HPSG. In *Computational linguistics in the netherlands*, vol. 1997, 119–133.

Vandeghinste, Vincent. 2008. *A hybrid modular machine translation system: LoRe-MT: low resources machine translation*: dissertation.

Vandeghinste, Vincent & Ineke Schuurman. 2014. Linking pictographs to Synsets: Sclera2Cornetto. In *Lrec*, 3404–3410.

Vandeghinste, Vincent, Leen Sevens, Ineke Schuurman & Frank Van Eynde. 2015. Translating text into pictographs. *Natural Language Engineering* 1–28.

Vater, Heinz. 1975. Toward a generative dependency grammar. *Lingua* 36(2-3). 121–145.

Velldal, Erik. 2009. Empirical realization ranking .

Velldal, Erik & Stephan Oepen. 2006. Statistical ranking in tactical generation. In *Proceedings of the 2006 conference on empirical methods in natural language processing*, 517–525. Association for Computational Linguistics.

Vossen, Piek, Katja Hofmann, Maarten de Rijke, Erik Tjong Kim Sang & Koen Deschacht. 2007. The Cornetto database: Architecture and user-scenarios. In *Proceedings dir*, vol. 2007, 89–96.

Wahlster, Wolfgang. 2013. *Verbmobil: foundations of speech-to-speech translation.* Springer Science & Business Media.

Zwart, Jan-Wouter. 2011. *The syntax of Dutch.* Cambridge University Press.

Summary

This thesis describes the development of an automatic translation system that aims to enable language-impaired, intellectually disabled individuals to compose written messages simply by selecting a sequence of pictographic images. The advantages of using pictographs as an alternative to written (and spoken) communication are well-established, and several applications have already been proposed that capitalize on this fact. These applications focus primarily on face-to-face communication. In recent years, increasing attention has been paid to the potential of using pictographs to open up the online world, so that users with intellectual disabilities can benefit from the same tools for remote communication (email, instant messaging, social media) which define so much of what it means to be a socially active member of society nowadays. The work presented in this thesis positions itself in this new trend.

By way of contrast with existing approaches for pictograph-to-text translation, the system that we develop here, i.e., *Depicto* (short for *DePictofication Tool*), takes a 100% rule-based approach. That is, all stages in the translation process make use of linguistic rules, as opposed to statistical data. On paper, such an approach has several advantages over data-driven alternatives. In particular, it makes it possible to encode elegant generalizations about the pictographic input, such that translation is maximally expressive, the output is consistent, and one of the main challenges for statistical methods is overcome, i.e., that there is no data on which to train a model of the pictographic input. Rule-based approaches are also costly, though, requiring resources developed in large part by hand. Thus, aside from the obvious objective of testing whether this approach can be realized (i.e., by getting *Depicto* to work), we also explore how development can be made more *feasible.* In addition, we set two design criteria: first, the system must be *sensitive* to the needs of its users; second, it must be possible to *extend* to other target languages. (Currently, the system translates to Dutch.)

In Chapter 2, we introduce all third-party resources used by the system. In

Chapter 3 we show how the pictographic symbol set *Sclera* can be analysed as a natural language and how this language can be modelled by a constraint-based grammar. This grammar is written in an implemented variant of the HPSG framework and forms the core of the first of *Depicto*'s three modules: the analysis module. In the first half of Chapter 4, we see the semantic structure of analysed pictographic sequences is converted, or *transferred*, so as to be compatible with the input expected by the target language grammar. This happens in the second module in the *Depicto* chain. In the second half of Chapter 4, we describe a basic grammar model of Dutch and show how this is used to *generate* well-formed sentences based on the translated semantic structure yielded by the second module. Next, in the first half of Chapter 5, we evaluate the system as a whole. We argue that both its precision, i.e., ability to produce well-formed output, and performance are high, but are forced to concede that its coverage is limited. We also show how the *Depicto* system fares when pitted against a fundamentally statistical system, namely, the *Picto2Text* system ((Sevens et al. 2015)).

All in all, we conclude that *Depicto* is able to successfully translate (a very limited subset of) pictographic sequences into well-formed natural language sentences. Moreover, by adopting an explicitly modular design, we try to make the system as appealing as possible to developers dealing with other target languages. At the same time, while modularity certainly helps, extending *Depicto*'s analysis module remains a costly task. Whether such costliness is an adequate trade-off for the indisputably high quality of the system's output will be determined by future work. As a *translation system*, *Depicto* can be considered a success. As an *assistive writing tool*, however, it needs improvement. Its analysis module imposes overly stringent constraints on the order of elements on the pictographic input, and its generation module is set to output every single sentence that is well-formed according to the grammar of the target language. In future work, we will focus on minimizing the first limitation and removing the second altogether. This will involve exciting experiments in hybridizing *Depicto*'s rule-based core with other, more statistical approaches to translation/generation.