Agreement Morphology, Argument Structure and Syntax (8. Revision)

Marcus Kracht
Fakultät für Linguistik und Literaturwissenschaften
Universität Bielefeld
Postfach 10 30 31
33501 Bielefeld
Germany
marcus.kracht@uni-bielefeld.de

June 29, 2016

Avant Propos

This manuscript has a long history. The origins go back to 1992, when I first met Albert Visser and Kees Vermeulen while I was working in a project on the parallels between natural languages and programming languages. I learned about referent systems and slowly the idea idea of trying to implement this on a larger scale for natural languages formed itself in my head. Then I taught a course about this at the 1999 ESSLLI in Utrecht. After that the matter came to a halt. When I was asked to teach advanced computational linguistics I thought of using that manuscript as a guide. I also decided that since the theory had become quite complex it was best to write some computer program to verify it and perhaps provide the interested student with an evaluative tool. This took much longer than I hoped. Eventually, it resulted in a rather stable version and I decided that it would be best to incorporate the software into the teaching. This book reflects this long development. I try to make the main points independent of the software, so that it can be read and understood even without it. On the other hand, I do think that the best use one can make of the theory is actually applying it. The computations involved are lengthy, however, and best done by a machine. It is for this reason that I advise the reader to use it. It is free and completely open source.

Albert Visser's ideas concerning semantics in general and how to set up a really clean framework for dynamic semantics in particular have had a profound impact on me. It has always been his intention to provide a mathematically elegant and sound semantical framework for natural language. Yet, it is one thing to believe that such a framework is possible and another to actually provide it. This book is about how his ideas on semantics can be made fruitful in linguistic theory. I had to sacrifice some features of the original system. My only excuse here is that language just isn't the way we would like it to be. There are many facts to deal with, and they tend to mess up the system a fair bit. There is however also a fair

chance that I haven't managed to make things as simple as I could have done and I apologise for that.

This work has been presented on various occasions and in various stages of incarnation in Paris, Tübingen, Potsdam, Berlin, Saarbrücken and Los Angeles. I wish to thank those in the audience who have helped me to bring out my ideas more clearly and who have pointed out numerous deficiencies of earlier versions. Thanks go to Katherine Demuth, Alan Dench, Jan van Eijck, Hans-Martin Gärtner, Willi Geuder, Hubert Haider, Ed Keenan, Ben Keil, Hap Kolb, András Kornai, Anoop Mahajan, Gereon Müller, David Perlmutter, Ed Stabler, Markus Steinbach, Kees Vermeulen, Albert Visser and Ralf Vogel. I owe special thanks to Markus and Ralf for long discussions in Potsdam on argument structure and polyvalency. I am deeply indebted to Albert, András and Hans-Martin for their enthusiasm, without which such an endeavour is not possible. Above all, thanks to Johanna Domokos for her patience with me, for helping me with Finnish and Hungarian and her rich knowledge of languages about which I had never heard before.

The work by Udo Klein was funded by the Alfried Krupp von Bohlen und Halbach Stiftung.

For the errors that remain I remain solely responsible. I appreciate any remarks from my readers, as they will help me to improve on this subject.

Bielefeld, October 2015, *Marcus Kracht*

Introduction

The last 30 or so years have seen an enormous unfolding of formal semantics sparked off by Montague Grammar. Likewise, Generative Grammar for somewhat longer has been the major syntactic theory. Both have established themselves as something of a lingua franca in linguistics. Yet, there is a problem that besets both of them: they disregard agreement morphology. For both theories, structure is all that counts. A sentence is analysed in structural terms and morphology often appears to be a mere luxury. Both Montague Grammar and Generative Grammar thus share a certain disregard for the surface. This is in sharp distinction to the earlier stages of linguistics where form was the primary target of study. Time has come for a synthesis.

The central claim of this book is that there is an interface between syntax and semantics called *argument structure*, whose main responsibility is to declare how and where the semantic arguments of a unit show up on the surface. The argument structure consists of a list of declarations which in their simplest form look like this:

(1) $$\langle \triangledown : \begin{bmatrix} \text{CASE} : & nom \\ \text{NUM} : & pl \end{bmatrix} \rangle$$

Here, x is a variable, and $\triangledown$ is a declaration about the way the variable is manipulated. $\triangledown$ says that the structure needs another constituent (the complement) from which it will expects the value of the variable when merged. The remaining part, an attribute value structure, formulates expectations on the form (or morphology) of the argument. They are expressed by means of certain attribute value matrices, but may also be understood as abstract properties. In terms of categorial grammar we are working with a very flat structure; a head can only declare what kinds of arguments it needs and what morphological properties they have. There is no recursion: the space of properties of the arguments is finite. Syntax is reduced to

a question of argument handling, while morphology feeds into the attribute value structure. Directionality is conspicuously absent from (1), it is handled by a separate component that talks about string manipulations. It may be referred to as surface morphology. This is not so unusual; we can interpret the variable handling in terms of X-bar rules, while the morphological component expresses the category of the item. This will of course be refined later; but it works as a rough guide for things to come.

This theory assumes no syntactic structure and no movement. Also, it does not even distinguish morphology and syntax. However, more realistically one should think of it as a lexicalist theory on a par with categorial grammar. The combinatorics of the words are encoded in the argument structure, and there is nothing beyond it that matters. The argument structure also contains information about string manipulations. One will inevitably find that such a theory meets a number of challenges. Verb second is a case in point. In German, the finite part of the verb occupies the second place in the sentence; however, it typically leaves behind the verbal prefix and all the other stuff that normally precedes it in a subordinate clause. The solution that I have adopted is to allow constituents to be discontinuous. This is in line with recent trends that also read the Minimalist Program as a theory of discontinuous constituents (Michaelis 2001, Stabler 1997).

Another important issue is computational complexity. Montague Grammar relies on the typed λ-calculus to do the argument handling and variable substitution. It is known that reduction of typed expressions is very expensive unless they are of the form that we assume here: a function being applied to several arguments, none of which is complex. Since Montague Grammar is quite inflexible in the way it handles its arguments a lot of argument shuffling is needed to assume correct processing. This constantly requires applying a function to a dummy variable and reabstracting it. The present framework deliberately makes variable handling more flexible and thereby achieves a flat type structure. The gain is an algorithm that processes sentences in polynomial time, the exponent being quite low.

Below is an summary of the contents of the chapters.

Chapter 1 briefly introduces the software that accompanies with book. You get introduced to the basic facts of how to install and use it.

Chapter 2 introduces the special methods for handling strings. Basic facts concerning the syntax side are explained. We shall deal in particular with discontinuous constituents and with what is called “glued-strings”. These are strings that

come attached with expectations on their left and right context.

Chapter 3 deals with the basics of Montague Grammar and how the composition of meaning is achieved in it. We shall briefly comment on the problematic aspects of it and introduce a new semantics based on Referent Systems, due to Kees Vermeulen and Albert Visser. Referent systems treat variables as anonymous; during the merge of two semantical representations, the names that they have in each representation cannot be shown to the outside. There is however an agreed set of so-called *names*, by which variables can be identified under merge. We shall assume that the names are principally form related; that is to say, they contain information about the morphological shape of the sign. Additional information is the sort of the variable and the direction where the sign is found. For example, the variable of the subject of a sentence in German is the one carrying nominative case, while in English it is both case (for pronouns) and the fact that it is to the left of the verb. The collection of statements that tell us which variable is identified under what name is called *argument structure*.

Chapter 4 introduces another novelty: parameters. It is claimed that in addition to making certain formal variables (referents) cross identify each each, there are plenty of variables over specified domains (mainly time, world, person and location) that are taken along and get unified in tandem with the other referents. The mechanics of parameters is however somewhat different, as they consist mainly of contextual parameters known already from Montague's work on pragmatics. It is claimed here that parameters each induce sequencing effects, as are now well known from the literature. For example, property ascriptions typically are time dependent, in which case they are also called stage-level predicates. (We avoid using the terminology since it is of no further significance here and we want to avoid any commitment to an accompanying theory of such predicates.) One is the director of a company for a certain stretch of time only. On the other hand, the time dependency hardly shows up in the form of an argument. It does matter on the other hand in expressions like `former` or `ex-`. The time variable has a different behaviour from typical argument variables simply because it is not identified by an overt property. Parameters therefore function differently. There is a small number of roles each of which address a context variable. For time variables these are *story time*, *predication time* and *reference time*. Parameter statements link actual variables to these roles. They eventually get their values through the context. It is possible to relink variables to different roles, and this causes what is known as *sequence of tense*. This mechanism is not restricted to tense; Philippe Schlenker

has observed that it also applies to person and world, while Kracht & Smith have shown that it additionally applies to location.

Chapter 5 deals with the question of how it is that morphology can shape the name of variables in a representation. We shall assume that lexical roots contain only a minimum of information on names. Most names are added in the process of forming the actual word. This shall give flexibility in the names under which various words expect their variables. A case in point is diathesis; by applying diathesis to a verbal root before the actual case requirements are being fixed we can account for the different case marking pattern in passives. Furthermore, it follows that agreement morphology overtly expresses the form requirements of the head for its arguments. In addition, the actual morphs may be conditioned by the names of the variable they modify. For example, in Latin the person suffixes are different in the passive. Since there is no overt marking of passive, this ensures that passive is overtly expressed even though not at the place where we expect it. We may call this *delayed exposure*. If however the conditioning morpheme is nonzero, this can lead to cumulative exposure. For example, the person endings in the Latin perfect are different from the ones in the other tenses. So, the presence of perfect person endings signals perfect in addition to the perfect morpheme itself, which is nonzero.

Chapter 6 presents a detailed study of case. Case is both a morphological property and a syntactic one. We start by outlining some morphological case systems and subjecting them to an analysis. Then we look at the way morphological case translates into syntactic case. The basic insight is that case that is not selected is actually semantic, while a case that is selected is syntactic. Whether or not a case is selected is a property of the head, and cannot be fixed a priori. This is our solution to the debate whether Finnish local cases are structural or semantic (see Vainikka and Niikanne). It is argued that selection is quite different from agreement. Selection is selection of a particular morpheme. Selection will make the semantic contribution of morpheme void. Agreement requires an agreement controller, and the semantics on the agreement controller is determined only by the properties of the controller. In this way plural agreement can still give rise to plural semantics, while selection of ablative case cancels its meaning completely. Of course, selection of plural has the same effect.

Contents

1 The Software: Installation and Use **13**

1.1 Installation . . . 13

1.2 The Structure of the Program . . . 18

1.3 Making Dictionaries . . . 24

1.4 Multilingualism and Keycodes . . . 28

1.5 Handling User Data . . . 29

1.6 System Settings . . . 31

2 Exponents and Rules **35**

2.1 Strings, Morphs and Morphemes . . . 35

2.2 Glued Strings . . . 38

2.3 Morphological Classes . . . 45

2.4 Discontinuity . . . 53

2.5 Reduplication . . . 61

2.6 The Morph . . . 63

2.7 Implementation Issues . . . 68

3 Argument Structure **71**

3.1 Overview 71
3.2 Basic Semantic Concepts: DRT 75
3.3 A New Theory of Semantic Composition 83
3.4 The Transmission of Referents 90
3.5 Signs 101
3.6 Basic Syntax 112

4 Features **119**

4.1 Different Kinds of Features 119
4.2 Syncretism 127
4.3 Agreement 137
4.4 Infinitives and Complex Predicates 147
4.5 Logical Connectives, Groups and Quantifiers 159
4.6 Implementation Issues 168

5 Parameter **171**

5.1 Properties 171
5.2 The Mechanics of Parameters 176
5.3 Tense and Aspect 185
5.4 Time in the Noun Phrase 190
5.5 Reconsidering the Structure of the Noun Phrase 196
5.6 Predicative and Attributive Adjectives 205
5.7 Sequence of Tense 213

6 Latin **219**

6.1 The Morphology of Latin 219
6.2 The Verbal Paradigm: Relation Change and Verbal Agreement . . 228
6.3 Tense, Mood and Aspect 231
6.4 The Simple Clause 234

Appendix: Coding and Notation 235

A Symbols 250

B Index 251

Chapter 1

The Software: Installation and Use

THIS chapter explains how the software can be installed and used. We will consider only Unix based systems, as my expertise about other systems is limited. This chapter is written for a user with just a little experience in Linux, so it assumes very little knowledge of the platform.

1.1 Installation

Before I start with the explanations, I will point to a few more place where information about the system and its inner workings can be found. The last chapter provides an overview of the notation and the coding of the various elements introduced throughout this text. This is helpful because it provides everything at a glance. Additionally, every chapter ends with a section where implementation issues are being discussed. Apart from these sections, however, I will not make reference to the implementation. This is because the implementation is designed to support the understanding of the particular theory explained here. The theory should in principle be able to speak for itself.

The software is written in a language called OCaml. Although still widely unknown it is gaining in popularity. Its advantages are, among other, that it is fast and reliable. For someone who is not a programmer the strict typing (though at the beginning a big frustration) is a blessing: when we get the types right we know

that the program will most likely do what it is supposed to do. OCaml is developed at INRIA Lorraine. The site is `http://caml.inria.fr`. The current version is 4.02.3 (October 2015). The program should work under older versions as well. To use the system described here you need to have Tcl/Tk installed. Please check beforehand and proceed with the installation of OCaml only after that. Tcl/Tk can be obtained for example from `http://www.tcl.org`. To install OCaml, go to the website and follow the instructions. It is worth doing it yourself, because it teaches you some elementary facts about your system. One is to download software to a safe place and installing it. You need to create a temporary directory, store the source file (*ocamlversion*`.tar.gz`) there, unpack the software, read the instructions under INSTALL and follow them. To ensure that the referent systems software runs, you need two more modules:

- `Xml-light`. This is not included in the distribution, but absolutely necessary. It provides a parser for XML.
- `camomile`. It contains the locales for different languages, which we shall talk about in Section 1.3. It has tools for handling UTF8 and other Unicode standards. Although one could in principle do without it, we have chosen to use it for the reason that increasingly UTF8 becomes native on systems (and editors alike).

Once OCaml is working, you can take the next step and install LaTeX.

The reason for using LaTeX is that the program outputs the data to LaTeX, which is a text processing software. I have considered using HTML but have been unable to provide output of a similar quality. Fortunately, LaTeX, is also free software. Its installation is worth your while, since it produces output of superior quality to any other text processing software. For our purposes, it has the benefit that the source code to LaTeX is in ASCII, and can be read and manipulated with any standard software. (This is important, since the program writes the source code to the text processor that will then process and view the pages to you.)

LaTeX and OCaml are absolutely necessary to run the software. You may use it without Tcl/Tk, but that is rather clumsy. Tcl/Tk gives you a more comfortable input method than the standalone system.

All of the above can be downloadad as Unix-packages in your favourite distribution (for example, Ubuntu).

You are now ready to install the software. The procedure is simple and uses very few sophisticated machines. All you need to do is download and unpack the version, make the scripts executable and run them. We explain now how that is done.

Create a directory, referred to here as *RefSys*. Here, the italics denote an arbitrary name; but remember it is a complete path name specified from the root, so it begins with `/home`. For example it may be

(1.1) `/home/marcus/Documents/sprache/agr/programme`

Now say

(1.2) `cd` *RefSys*

That is, do not type `RefSys` but rather type whatever the directory's name is where the program should go into. Remember, italics indicate variables for certain strings. Alternatively, I type the following.

(1.3) `ln -s ~/Documents/sprache/agr/programme Referent`

thus creating what is called a symbolic link. And then I can simply type

(1.4) `cd Referent`

Next say

(1.5) `tar xvf referent_v5-8.tar`

or whatever version you have obtained. This unpacks the archive and adds many files to it (and creates some subdirectories as well). You may delete the original file after that (the file with the ending `tar`, that is, not the files ending in `ml`, which you may want to keep if you want to manipulate the program yourself). Furthermore, there is a subdirectory `bin` where all executable files are stored. Information concerning the system can be found in the file `00readme`. We recommend you take a brief look at it. Now type

(1.6) `./bin/setup -h`

This should show you how to run the program `setup`. Among other things it will tell you that setup can be done in German (default) or in English by typing

(1.7) `./bin/setup -l en`

Minimally, not only the language is set, but also all files in the subdirectory `bin` are made executable.

Further, you can simplify your life with regards to the usage of the system somewhat. If you do not like the have to always add the prefix `./bin`, make the directory searchable for the system. This can be done by typing

(1.8) `bin/setup -p`

This adds the current path to the executables to a (hidden) file called `.profile` in your home directory. In actual fact, what the scipt does is append `/bin` to the directory where you currently are and add the result to the list of searched paths. However, as many systems have their individual preferences as to where the proile is stored, it is advisable to add the path manually. Thus, search for the correct file and look for the definition that sets `PATH`. Change that definition by squeezing in the directory *RefSys*`/bin` after the equation sign. Separate it from the next directory by a colon like this:

(1.9) `$PATH=`*RefSys*`/bin:/home/marcus/bin:$PATH`

If you do not manage all this, do not worry. It means basically that when I tell you to issue a command, say `compile`, you must prefix that command by `./bin/`, so it becomes `./bin/compile`. This is because the commands are actually names of executable files, and the shell (bash or other) does not know where to look for it unless either you say where it is or you have given it a path to search for.

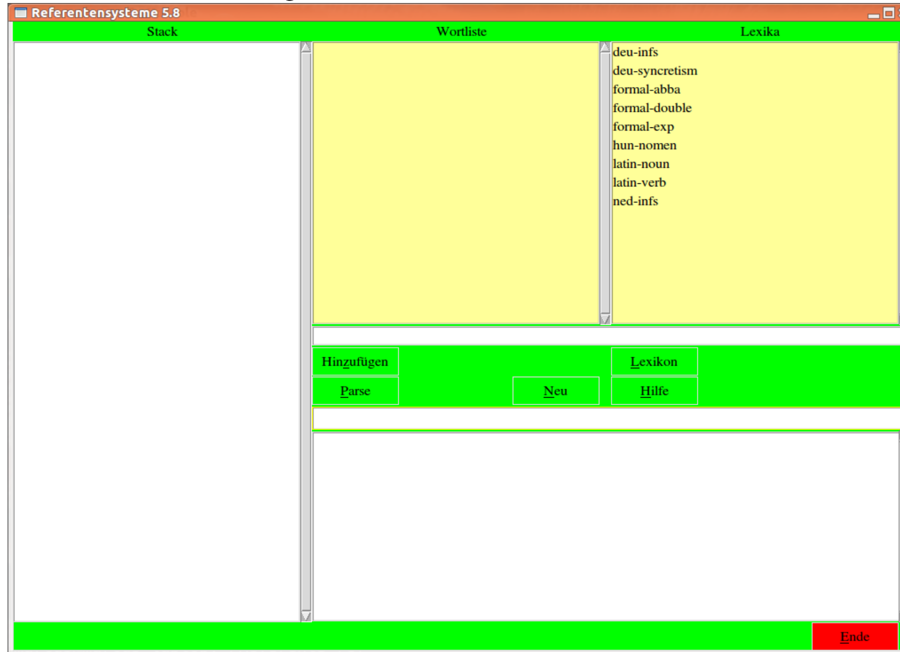
Having given it the basic path name in that file avoids having to invoke the line (1.9) everytime you start the shell. You are now ready to compile the system. To do that, type

(1.10) `compile -r`

(or `./bin/compile -r` if you haven't bothered about path names). The result of the compilation is output to a file `compile.log`. If that file is empty, your are prompted with a success message otherwise you are told to look into `compile.log` to learn about the errors. (Type "`more compile.log`" and you get a listing of its content.) The magical incantation of the system is now this:

(1.11) `ref`

Figure 1.1: The Tcl/Tk Window



And the following should be the reaction by the computer (where the version number is that of your installation, thus might be different):

(1.12)
```
Objective Caml version 4.00.1
#
```

After that, the Tcl window will appear, see Figure 1.1. More about the layout and functionality of this window later. Suffice it to say that the lower of the two line-shaped windows is the one where you can type in commands. To exit, type *quit* and hit <`return`> or press the red button and the lower right corner.

When you exit that window, you get back to the OCaml shell. You can use that shell by typing after the prompt (#). Remember that the program is written in OCaml; what you have opened is an OCaml session where the modules for the program are loaded. So you can actually type in OCaml code at this point, or just continue to use the system as given to you. The first will at some point will become a convenient option for you. To exit OCaml, type `#quit ;;` (yes, you need to type the sequence hash-followed-by-`quit` and double semicolon) in addition to the hash prompt of OCaml.

The procedure `compile` produces a file `.ocamlinit`. The period means that it is not shown under a standard listing (the command `ls`). To see that it is nevertheless there, type `ls -al` instead. This file is used by OCaml when starting the session. It contains a series of OCaml commands to be executed before asking you for any input. This ensures that all modules are properly loaded and paths to OCaml-subdirectories are properly set. Do not remove or edit this file. If it needs editing, the problem is most likely somewhere else.

This manuscript does not intend to teach you neither shell scripting, nor OCaml nor LaTeX. For any of these there are good books available. Moreover, for what you will have to do you do not need to know any details (though that might be helpful). If you want to have an overview, type

(1.13) `man bash`

and Linux/Unix will give you an introduction to bash. For OCaml, the language comes with a manual, which is terse but quite accurate and useful. LaTeX comes with extensive documentation, and there are plenty of good books on that subject.

Let me end by saying that all source files are ASCII files and can be read using any simple file viewer. You may decide to view them using a text editor (I use `gvim`, again free software, others prefer `emacs`). This applies to the source files (ending in `.ml`) as well as to the "binaries", which are just bash scripts that schedule the execution of certain shell commands. I have not made use of sophisticated installation stuff like make tools for the reason that I wanted to stay in control of the implementation with the fewest possible intermediaries. (Also, in my experience, there is no installation of Unix that has all these things in them, so the fewer extra tools I use the easier.)

1.2 The Structure of the Program

The site *RefSys* has several subdirectories:

- `./` (In other words, this is the directory *RefSys*.) This is where the sources (ending in `.ml`) are stored as well as the files ending in `.mli`, `.cmi`, `.cma` and `.cmo`.

- bin/ The is where the executable files are put. Once the system is up and running, you need not worry about them.
- parse/ This is where the output of parses goes. LaTeX-source files can be found there as well.
- dict/ The place for the dictionaries. You may put dictionaries there or modify the existing ones.
- language/ The place for the headers. They are needed for the system to generate texts and headers. You do not need to worry about them.
- tmp/ This is where occasional files are being put, for example dump files.

For you as the user, the place where you shall most likely be doing work—unless you want to modify the program itself—is dict/ (because of the need to create dictionaries).

The source files end in .ml. There is about two dozen of them. When you invoke the scripts they run a series of procedures over them (see also the file bin/process). The effect is that OCaml creates a number of auxiliary files. If your file is called fib.ml then it will create fib.cmi, fib.mli, fib.cma and fib.cmo (and maybe some more). You need not worry about their use. What is important is that the files provide the modules of the system and must be processed in a certain order. This is taken care of by the shell scripts, which call on OCaml to process them in precisely that order.

The scripts can create several versions of the program. The easiest one to use is the standalone version. It consists in an executable file ref stored in bin/. If you do not have adapted the paths you need to type

(1.14) ./bin/ref

to invoke it; otherwise, typing ref is enough. It opens a window, which has a complex functionality. Basically, it allows you to load and manipulate dictionaries, and parse strings using dictionaries. Also, based on a dictionary you can also try to draw items from a dictionary and merge them so see what happens.

There is a command line, where you type a command and hit <enter> to execute it. When you do not know what the options are, type

(1.15) help <enter>

Almost any command can be issued from the command line, but there are some buttons for convenience.

In order to proceed, you need to first load a dictionary. They are listed in the upper right column. Now, either you move the cursor over the name and click on it and hit the button <`dictionary`>, or you enter the command

(1.16) `read dict` *`dictname`* <enter>

To see the dictionary, type

(1.17) `show dict` <enter>

You will be shown a big document, but items in it cannot be selected by clicking. Instead, the potential items are shown in the middle column, and can be clicked on for parsing. Items have IDs (for the command line) and a string, by which you can select them by clicking. Clicking adds them into the parse window. The other method is a way to quote an item in the command line.

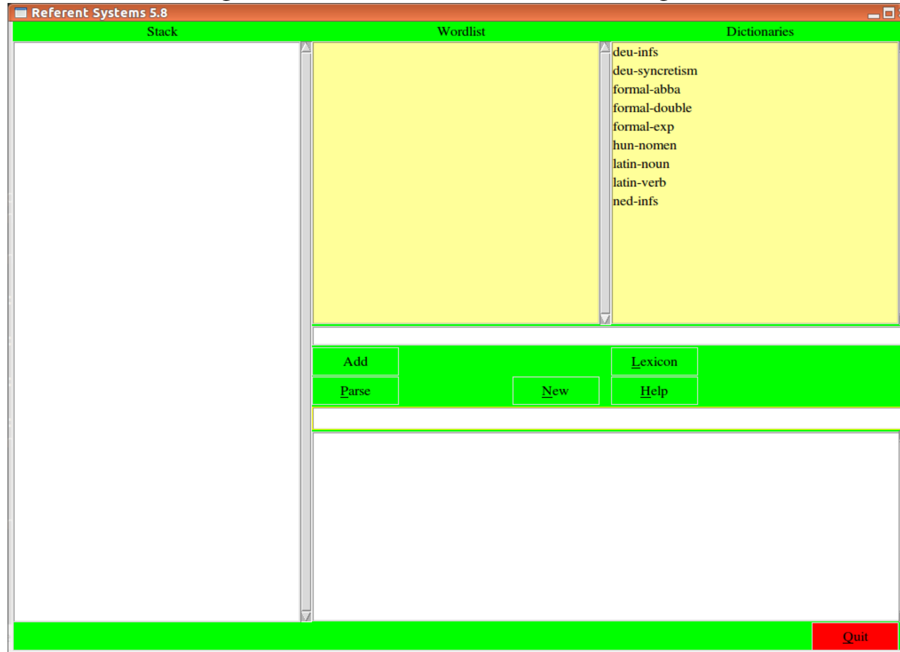
The commands given by you are recorded in a command stack just below the command line. The system answers each command and writes its answer into that window, where you can see it. Mostly it will just say "OK". (It means that your command is meaningful and has successfully been executed.) This dialog appearing in the stack can be dumped in a file called *`anyname`*`.dmp` for later analysis. (The directory for that file is `tmp/`.) If you do not specify a string for *`anyname`*, `session` is used.

The command line lets you edit dictionaries (to be explained later), and manipulate items. This allows you to construct complex items by giving an explicit analysis. Since this means going through several steps of merging items, the results are recorded in a stack shown on the left. The stack contains items by item or complex items by what is known as an analysis term. Items can be drawn from that stack by their number. The functionality of the stack is explained in the help menu.

What Happens Now?

Suppose now you are in the window session, shown in Figure 1.2. The header of the window says "Referent Systems 5.8". This means that the version number is

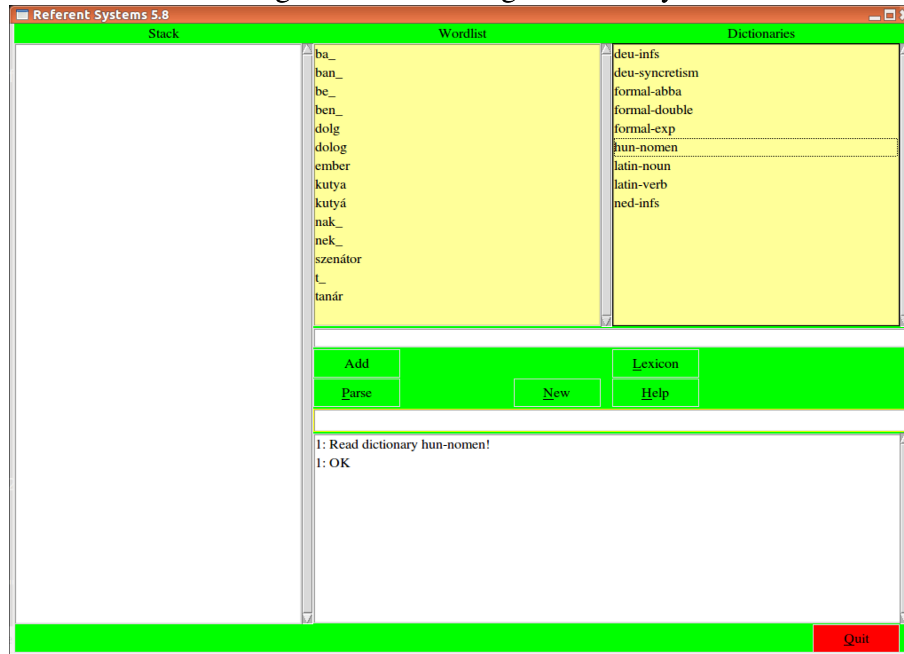
Figure 1.2: The Tcl/Tk Window (English)



5.8 (current at the time of writing). The left hand white window called "Stack" shows you the stack of elements that you have produced. You cannot type into that window. There are additionally two yellow window at the top. The one labelled "Lexicon" offers you a choice of dictionaries. Just click on one, and two changes happen (Figure 1.3). First, the window "Word list" suddenly contains a list of elements. If you click on them, they will be added to the upper white linear window, which is below. This window, the parse window, can be edited either by typing into it or by adding from the wird list. This is because sometimes using the keyboard to enter things is difficult. Also, it may be useful to know which elements one can at all chose from.

The second change is the found in the bottom white window. It now contains two lines, one saying "1: Read dictionary hun-nomen!" and the next saying "1: OK". This window cannot be edited. It is maintained by the system and keeps track of the dialog. It records he commands by the user and the reactions by the system. For each user input two lines are added and given a number. The number just counts the commands, and the first line is a repetition of the command and the second line the response by the system.

Figure 1.3: Selecting a dictionary

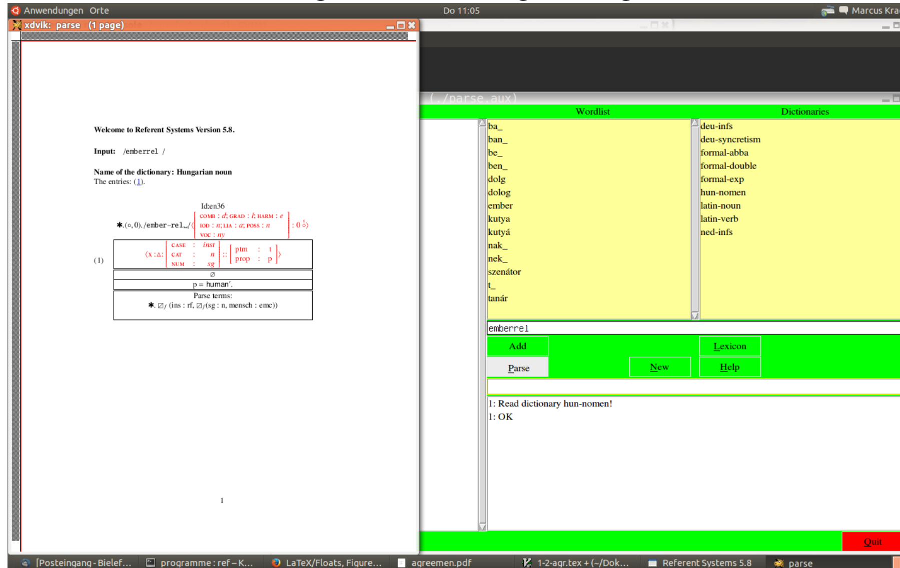


Now that a dictionary is loaded, you can parse a certain string. If you use the upper line, you can enter either by selection or by typing. To get a parse, you now need to press the "Parse" button. Say we have typed in `emberrel␣`.

The program then takes the string, and first does some preprocessing. For example, as we will see later, you can enter foreign characters that are not on your keyboard through a special sequence. This sequence will be translated into the character. Additional preprocessing needs to be done to take care of upper case letters, for example. The string so obtained is then scanned for occurrences of lexical entries. This is the initial phase of the parsing. Everything that the program does is completely surface oriented: the lexical entries must be found verbatim in the string (though they can consist of several parts). The result is then given to a parser that tries to build complex constituents from the ones already identified. Whenever a constituent is found the system enters the parse term. When the parse is finished, it evaluates all terms corresponding to the entire string and reports the result to the user. If no parse is found, that is reported, too.

The result is output to a LaTeX-source document, `parse.tex`, which is then processed in the standard way into a PDF-document or a DVI-document and dis-

Figure 1.4: Parsing a string



played, opening a separate window. You can find it in the directory `tmp/`. The file will be overwritten if you do another parse, so if you want to use it, copy it to another location before continuing.

Here is now a screenshot of the entire desktop. The extra window on the left must be closed before continuing. It contains a hyperlinked document, which once again states the version number and then shows the string again and its parses. (There is only one for this item, but that is just an accident.) The details will be explained later.

What If It Does Not Work?

How do you quit? Enter

(1.18) `quit <enter>`

or press the red button, at the bottom to the right. This lets you return to OCaml. Now enter

(1.19) `#quit;; <enter>`

(yes, the hash is what you enter as well) and you are back to the shell. Here you find all the files, which you can (and sometimes have to) freely edit, though at your own risk.

A hundred things can go wrong. The dictionary may be faulty. For that there is no other recipe but to pay close attention to what you wrote and what report you are getting. However, other problems may be more serious. One problem that may occur is that the system is unable to make a proper choice of the initial settings. To review the default settings, take a look at the file `options.ml`. These may however not be the settings the system has when you run it. Therefore, hit

(1.20) show options <enter>

for a complete listing while executing the system. It should have recognised whether you are running standard Linux or Mac OS X. This is seen in the value of the entry osys. However, that is in itself not the most important value. More important are meth and view. The first has two values, `dvi` (which means that it calls the command `latex` to execute LaTeX) or `pdf` (which means that it produces pdf-output, though not via `pdflatex`, which has become somewhat unreliable). To see whether this is your problem, quit the system and look at the file `parse/parse.tex`. Type

(1.21) `ls -al parse/parse.tex`

and look at the time it has been created. Now check whether running latex (and the other commands) has an effect.

If LaTeX halts at some point there is a problem with the output. In this case the software contains a bug. In such and other cases you should drop me a line at

(1.22) `marcus.kracht@uni-bielefeld.de`

stating what the problem is that you are experiencing. (Remember to include all data necessary to identify the problem, such as the dictionary (the file!), the sentence you parsed, the version you used, and what settings you have.)

1.3 Making Dictionaries

Source files are generally in two formats: they are OCaml-files or they are XML-files. If you are not accustomed with XML, that is not tragic. The basic ideas

can be grasped quickly. Basically, XML provides for a rich structure by enclosing items in a so called tag. A tag begins with <*tagname*> and ends with </*tagname*>. Whatever occurs between these two, is the value of the tag. If the value is empty, we may also write <*tagname*/>. For example, here is a snippet of XML.

(1.23)

```
<ma>
    <mi/>
    <mo/>
    <hdl>
        <unit>
            <prt fct="true" pos="0"/>
        </unit>
    </hdl>
</ma>
```

This means that some "ma" structure consists of a "mi"-element, which is empty, a "mo"-element, which is also empty, and a "hdl"-element. The latter consists of a "unit"-element, which in turn consist of an (empty) "prt"-element. The "prt"-element also has two adjectives, "fct", with value "true", and "pos", with value 0. What this means concretely will become clear later. For the moment it is irrelevant. We only need to understand that it is valid XML-code and that it encodes the structure as we just explained.

Dictionaries presently have to be hand crafted. This means the following. You need to open a text editor (for example vi, gvim, emacs) to create or edit a dictionary. To do that, you can issue the command `open`. It requires the name of the file *name*, and then calls an editor (currently Gvim) to open the file named `dict/`*name*`.xml`, The extension `xml` is used for XML-files. The extension is important for the platform and the system so they know how to handle the file. If file already exists in the directory `dict/` you can edit that one, or else you will create a new dictionary. The directory can be changed by resetting the value of the option `dictdir`.

There is a schema file called `refsysschema.xsd` in the directory `dict/`. This is useful when you use a special XML-editor such as oXygen. If you declare that the dictionaries are subject to that schema then you can take advantage of syntax highlighting, automatic completion and more. Writing dictionaries can be quite easy that way. If you create another dictionary directory, be sure to copy `refsysschema.xsd` into if you want to make use of it.

A dictionary has three parts: a preamble, a section to enter the morphs and a section to enter words to be displayed. The preamble consists of two lines of XML-code.

(1.24)
```
<?xml version="1.0"?>
<dict xmlns:xsi="http://www.w3.org/2001/XMLSchema-
                                                    instance"
   xsi:noNamespaceSchemaLocation="refsysschema.xsd">
```

This portion is standard XML, all the rest will have to be explained at some point. Notice that the preamble mentions the schema file `refsysschema.xsd`. It is always a good idea to include that.

First, you should give the dictionary a name. This name is not to be confused with the filename. The system is designed to be language independent so that you can in principle make it speak Hindi to you. In that case you might want to give the dictionary a name that is recognisable for a Hindi speaker. The file `hun.xml`, for example, might have these lines in it:

(1.25)
```
<name vlg="de" string="Ungarisch"/>
<name vlg="en" string="Hungarian"/>
```

This means that if the system language is `de` (which, according to the ISO-639 two letter coding refers to German) then the system uses the name "Ungarisch" to call the dictionary; if the system language is `en` (English) then it uses the name "Hungarian". Finally, if you want words to be sorted in a particular way you need to specify the locale:

(1.26)
```
<locale>hu<locale>
```

This means that the words are sorted according to the rules of Hungarian. You can set the locale in any way you want. Be aware, though, that the orders may be radically different in different countries. For example, in Hungarian the letter `ö` appears after `o` and before `p`; in Finnish, it is at the end of the alphabet, after `z` (and `ä`). Locales have in general two parts: the first is again a two letter code for the language (and usually sufficient to identify the ordering). After that follows an optional underscore and a two letter country (not language) code. This is because languages may be subject to different rules in different countries. For example, English spelling is slightly different in the US and in Britain. That's why you have locales like `en_GB`, `en_US` and so on. German spelling similarly is different,

for example, in Germany (`de_DE`) and Switzerland (`de_CH`). Moreover, the letter ß is not used in Switzerland, and the letters ä, ö and ü are substituted by `ae`, `oe` and `ue`. The different ordering schemes are implemented in the module `camomile`, which is not part of the standard distribution of OCaml. It is for this reason that you need to *install* this software separately. Go to the website at INRIA and look for camomile and how to install it.

The last section of the dictionaries is completely independent from the second. You do not have to put anything in there to have a successful parser. It consists of a series of statements of the following form

(1.27)
```
<word>kutya<word>
```

These statements are just needed to know which words are going to be displayed in the user interface. This is because the units of analysis may be much smaller than words but you may decide the user will not have to input the parts, but rather only the words. Also, this allows to input symbols that may be difficult to enter on a keyboard.

The structure of items is too complicated to be explained here. However, one thing may be important to note right away. Most types of element that occur in a structure can be defined independently of all others and then recalled. This works as follows. In the Hungarian dictionary, we find the following line.

(1.28)
```
<hdl id="pre" gen="rg" lg="1"/>
```

The attribute `id` assigns the identifier `pre` to the item just defined (`lg` is of no concern right now). It can subsequently be used by issuing

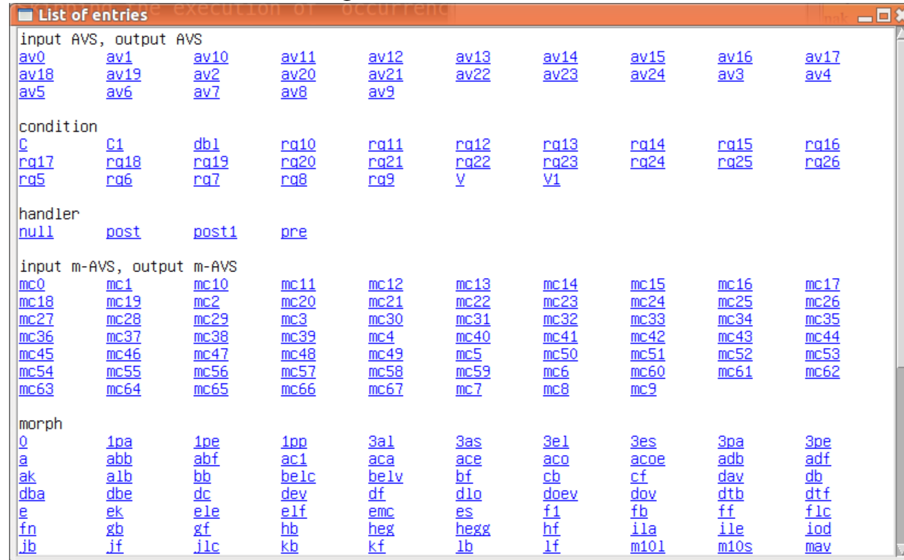
(1.29)
```
<hdl idref="pre"/>
```

In the present case this may not be such a great simplification since the alternative is just as concise:

(1.30)
```
<hdl gen="rg"/>
```

However, recall that the identifier can be given to any structure whatsoever, no matter how complex. Indeed, the internal mechanism is as follows: all structures are compactified by storing the subparts separately and giving them a name which is linked into the bigger structure. Any substructure that occurs twice is thus stored

Figure 1.5: The Entries



only once. By giving it an explicit name ourselves all we do is override the name that the system would give. To see what elements the systems has stored, type

(1.31) `show all <enter>`

You are given a list of items of different categories. It opens in a separate window, shown in Figure 1.5. Click on any of them and it will be shown. You can reference them by the name displayed in the list when calling them from a command line.

1.4 Multilingualism and Keycodes

Two aspects deserve special attention. One is that the system is designed to be used in different languages. Though the present version uses only English and German, other languages can be added easily. It effectively only requires to add translations for the files contained in the `language/` subdirectory. So, for a language with codename *`langname`* you will need the following files:

1. `errmessage_`*`langname`*`.xml` for generating the error messages in your language; these are the error messages that are issued when OCaml runs into trouble;

2. `headers_`*`langname`*`.xml` for the headers used in LaTeX when it shows you parses, dictionaries or other elements;

3. `messages_`*`langname`*`.xml` for the messages used by the system, for example the Tk-interface;

4. `realnames_`*`langname`*`.xml` for the names of the elements used in this theory. Basically, it provides a translation of the XML-tags into plain language.

The scripts could be upgraded as well, but after installation they are not needed again except for recompilation.

Using other languages creates a secondary problem: such languages use characters that are not used in English. A minor problem that this raises is that these characters need to be displayed. That problem is slowly disappearing thanks to the efforts of the Unicode group and the fact that text processing software (here LaTeX) can deal with plenty of languages (there is even an extension for Chinese, Japanese and Korean, though we have not used it at all). So from the side of handling Unicode (more specifically, UTF8), characters are increasingly easy to manage. One problem however still remains: foreign characters need to be entered by users with their own keyboards. Rather than fiddling with the keyboard layout we have decided to adopt a simple strategy for using such characters. There is a list of keycodes (the hashtable `unichar` in the file `zeichen.ml`) which can be entered to replace the characters. Presently, such a keycode is an ampersand followed by exactly two characters. (This allows for the easiest possible translation.) This character list can be changed either by editing the file or by adding single letters into the word list.

1.5 Handling User Data

The system is written in OCaml, but in order to use it, knowledge of OCaml should not be necessary. Users will only have to deal with one format: XML. All language data is entered in the form of an XML file. Moreover, the system itself dumps data in two forms: either as an XML file or as a LaTeX file. The latter is needed to produce the graphic display. This is because even though XML is simple and transparent it is visually not so attractive.

The way the system works is therefore as follows. When some data (a parse, a dictionary, some structure) is being stored, it can be stored in two ways: in XML or in LaTeX source code. Mostly, the LaTeX is actually derived from the XML code. You can decide whether or not you wish to save your data in XML mode or whether you just want to have it displayed in LaTeX without saving it as XML.

In this section we shall not explain in detail what structure these objects actually have; that is to say, the schema of the XML data files will not be explained here as it makes no sense to memorize that without knowing what all these different objects actually do. We will explain however how to manipulate them in the system.

Firstly, dictionaries are stored by default in the directory `dict/`. All other objects go into `tmp/`, we they usually are of temporary interest only. When you read in a dictionary, the system creates a so called workspace for you. This workspace contains the data structure of your dictionary, a temporary stack of objects that you wish to work on (shown on the left) as well as information about the dialog that you had with the system. This workspace can be cleared without clearing the dialog (`clear` <`enter`>) or by clearing also the entire session (`clear all` <`enter`>). The dictionary allows you to draw elements onto the stack by typing

(1.32) `push` *`element-identifier`* <`enter`>

The *`element-identifier`* is a name given either by you or by the system if you have not given any. Every time an object is created, it receives an identifier. Thus, as the session progresses the workspace is growing. Entering the command above will have the effect of taking the named element and putting in on the work stack. The stack can be manipulated by various commands. Issuing

(1.33) `pop` <`enter`>

will remove the upper element. Issuing

(1.34) `swap` <`enter`>

will swap the upper two elements. Crucially, two operations are of major interest:

(1.35) `diagnose` <`enter`>

This attempt a merge between the top most elements and display in detail what happens if it tries the merger. The order of the merge is as follows. Enter the

functor first and then its argument. So, the topmost element of the stack is the argument, while the element below it is the functor that will be applied to it. (Because we often want to add them in reverse order, the function `swap` has been added.) The merge is however *not* performed. Hence if you do want the merge to be executed, type

(1.36) `merge <enter>`

This actually performs that merge. Merging two elements eliminates them from the stack and adds the result instead.

1.6 System Settings

The system has various settings, which are handled through the module `Options`. They are explained briefly in `options.ml`. Here is their list in alphabetical order.

aspect (string) Determines the ways to view the object (specifically for entries). Options are

- "user" (user view)
- "system" (system view)

Default: `user`.

col (integer) the column width. Default 38.

compact (boolean) Determines whether or not we wish to to output elements to the user in the most compact way (labels are generated on the fly and repeated items are just linked to earlier ones). Default: `false`.

ddet (string) the way to render handlers (see `latex_dump.ml`);

- `dir` With that option, only the directionality of the handlers is displayed (left, right and open).
- `gen` With that option, the handler is ignored.
- `num` Shows the handler in full detail.

Default: `num`.

det (boolean) Whether or not the term is output in full detail; default: `false`.

dictdir (string) The directory, where dictionaries are stored and rertrieved from. Default: `dict`.

earg (boolean) If true it allows arguments to be of zero length (= empty). Default: `true`.

edet (string) the way to render exponents (see `latex_dump.ml`); Options are

- `dots` If that options is chosen, morpheme boundaries are displayed by a centered dot.
- bare If the option is chosen, morpheme boundaries are not displayed.

Default: `dots`.

expl (boolean) Whether terms should be issued at all. Default: `true`.

farbe (boolean) whether the output is coloured. Colour is used to distinguish various arguments (as they are crossreferenced in the semantics). Default: `true`.

form (string) Determines the basic format options. The choices are:

- `xml` (xml style format)
- `latex` (latex format for viewing)

Default: `latex`.

lang (string) the output language. Supported values:

- `en`
- `de`

Default: as given in the file `vars.lge`.

meth (string) Is either latex or pdflatex, but more options are conceivable. Default: `latex`.

osys (string) This names the operating system, normally recognised during installation; it is of restricted usefulness, since only the value `linux` is supported (so it is the default).

rdet (string) the way conditions ame rendered for glue strings; options are

- `plain` (do not show) and
- `fine` (show in all detail);

Default: `fine`.

showparse (boolean) whether or it shall produce an output to show the parse. Default: `true`.

showrank (boolean) Determines whether to show ranks; default: `true`.

simple (boolean) whether or not simplified style should be used (no gluestrings, no discontinuity). Is an output option. Default: `false`.

transsem (boolean) whether or not transformers are semantically empty; if true, the argument's semantics is suppressed under transform-style merge. Default: `false`.

view (string) the name of the viewer program; it can be evince, xpdf, or acroread. Notice the the string is passed on to the underlying system, so when you set the value you have to specify the shell command as you would normally type it. (Notice that if METH has the value `latex`, METH must be a viewer for DVI files, while is METH has the value `pdflatex` you must name a viewer for PDF files.) Default: `xdvi`.

wide (boolean) If true, parse returns all possible strings that are formed by the parse term when morph annotation is ignored. Default: `false`.

To take a look at the options, type

(1.37) `show options <enter>`

To change the value of some option *`opt`* to value *`val`* type

(1.38) `set` *`opt`* `to` *`val`* `<enter>`

Chapter 2

Exponents and Rules

IN this section we explain the syntactic machinery that we are using. Our approach is completely surface oriented: no rule ever changes or removes a symbol. In order to deal with the complexity of languages several important decision have been taken. The first is that constituents may be discontinuous. We allow them to have two or more parts, each of which are essentially strings. The next feature is that an entry consists of several morphs that can combine independently of each other. To account for surface combination restrictions and to keep the number of possibilities resaonably small we are working not with strings but with what we call glued strings: these are strings which can declare what surroundings they want to occur in.

2.1 Strings, Morphs and Morphemes

Language presents itself to us basically in the form of strings of sounds or strings of letters. It is in the latter form that we shall mostly deal with it here. This is no more and no less advantageous than dealing with sequences of phonemes. The only advantage is that we can use material that is readily available and can be understood even without knowing the phonology of a language or the phonetic alphabet. **Strings** are sequences of characters. We assume that a language has an alphabet A of characters and that strings are members of the set A^*. (So the strings

may be empty. This will be of significance later on.) If a language contains a string 'foot' we usually refer to the string using typewriter font and enclose it in slashes like this: /foot/. This will set it apart from other strings that might also be set into typewriter font (for example, computer code). The **concatenation** is denoted by $^\frown$, so that, for example, /foot/$^\frown$/print/ is the same as /footprint/ and *not* /foot print/, since we must account for the extra space (written here ␣). The **empty string** is denoted here by ε.

The present framework, like many other, assumes that sentences are composed from minimal units via some rules of combination. However, it is strictly **surface oriented** (see Hausser 1984 for one of the few frameworks that are similar in this respect). This means the following. A constituent or unit consists of one or several strings. These strings may neither be deleted nor changed by any rule. They can only be concatenated. It is also possible to duplicate strings, and strings may be empty. Thus, the regular English plural /dogs/ can be obtained through the concatenation of the strings /dog/ and /s/. The plural formation in Malay is by complete reduplication; for example, the singular /orang/ 'man' becomes /orang-orang/ 'men' (with a hyphen in between; in written language one often writes /orang2/). Both are perfectly straightforward for a surface based account. However, /men/ cannot be thought of as being composed from /man/ plus some 'umlaut'. Recall, namely, that umlaut really is a unary function that if applied to a vowel produces *another* vowel. Thus umlaut *changes* the input string and is therefore excluded. Thus, we are working within a paradigm of item-and-arrangement, not item-and-process (see Matthews 1978). On our part this restriction is more a matter of *convenience* than principle. If we did not assume such a rule then parsing strings would be much more difficult without revealing more about what we actually want to say.

In order to deal with the plural of /man/ we may choose a different route, however. We may simply list the two forms in the lexicon. Thus, while the word /dog/ will be entered into the lexicon as a noun root—which crucially lacks any number specification in a sense made precise below—, the words /man/ and /men/ are entered as nouns together with their respective singular and plural specification. For irregular forms this is the best way to proceed. For regular phenomena of alternation we have still other means. Consider the case suffixes of **Hungarian**. The inessive has two forms: /ban/ and /ben/. The first form is attached to all nouns that contain only the vowels /a/, /á/, /e/, /é/, /i/, /í/, /o/, /ó/, /u/, /ú/; the second form is attached to nouns that have the vowels /ä/, /e/, /é/, /i/, /í/, /ö/, /ő/, /ü/,

/ű/. (This is not an exact statement of the facts, there are some complications.) We group these nouns into two form classes, say [HARM : *front*] and [HARM : *back*] and declare in our rules that /ban/ will attach to nouns of class [HARM : *back*] and /ben/ to nouns of the class [HARM : *front*].

Furthermore, we shall enter into the lexicon two separate case entries: one for /ban/ the other for /ben/. However, this leaves something to be desired. We cannot explain, for example, why they function in the same way syntactically and semantically; nor can we explain why they exclude each other on nouns, or why it is that one can be added to a noun and the other cannot. We will be forced to say that there are two different cases, and certain nouns can appear in one of them while other can appear only in the other. This is unsatisfactory. Therefore we distinguish in this situation between a **morpheme** and a **morph**. The idea is that /ban/ and /ben/ share the semantics and are just alternate forms that are chosen by the context. Therefore we say that /ban/ and /ben/ belong to the same morpheme, or, alternatively, that they are **allomorphs** of each other. It is noted however that once again this treatment is by no means necessary; however, if we choose not to proceed this way there is an important regularity of the language that we would fail to capture.

In what is to follow, we assume that the dictionary is a collection of **signs**. We have some nonstandard idea about what a sign is. This is necssitated in part by the desire to maintain the unity of the morpheme in the theory.

Definition 2.1 *A **sign** is a triple $\sigma = \langle E, C, M \rangle$ where E is the **exponent**, C the **combinatorics** and M the **meaning** of σ.*

Most of this book will be consumed by investigating the middle part, the combinatorics. In this chapter, however, we shall look at the exponents. As we shall see, exponents are considered to be sets of morphs; morphs in turn are not just strings. We also consider them to have some complex nature. The reason for complicating morphs is that we want our approach to be basically **lexicalist**. This means that the rules of complex sign formation are kept universal and small in number. What varies from language to language is therefore only the lexicon. If two signs are to be composed, and they each contain a number of morphs we need to decide which morphs can be combined. Since there is no outside mechanism to tell us, the signs must themselves sort this out among themselves. They must have information in them that allows to determine which morph may be combined with which other morph.

2.2 Glued Strings

Ordinarily, we think of morphs as certain strings (either of letters or of sounds). These strings may contain punctuation marks or blanks (recall that the blank is rendered here /␣/). This is because we make no distinction between words and other parts. Thus, English has a plural morph /`s`/. This morph attaches itself to the noun. So, by attaching it to /`car`/ we get /`cars`/. Attaching here means concatenating either to the right or to the left. It is the plural morph that fixes the directionality: it affixes itself at the end, not at the beginning. That makes it a suffix, not a prefix. As it happens, though, not all nouns are set into the plural by adding an /`s`/. Some nouns have irregular forms (/`child`/, /`fish`/ and a few more). These must be dealt with separately. Their plural is not fruitfully analysed as arising from the addition of some string. Other nouns do have a regular plural, but it is not formed by the addition of /`s`/ but by the addition /`es`/, as in /`churches`/, /`clutches`/, and so on. The conditions that determine the choice between /`s`/ and /`es`/ are straightforward to formulate. If the noun ends in /`sh`/ or /`ch`/, then the plural is formed by adding /`es`/ rather than /`s`/; if the noun ends in /`s`/ the ending is /`ses`/. To capture this context dependency, we introduce the notion of a *glued string*. A glued string is a string that has two context conditions: one for its left context, another for its right context. These conditions specify properties of strings $\vec{u}$ and $\vec{v}$ such that $\vec{x}$ can appear in a string $\vec{u}\vec{x}\vec{v}$.

We allow strings to have associated with them two sets of conditions on either side: a *positive* condition, stating that the string on its left must have a certain suffix (or the string on its right a certain prefix); and a *negative* condition, stating that the string on its left must not end in a particular suffix (or the string on its right must not begin with a particular prefix).

Definition 2.2 (Glued String) *A **requirement** is a pair $(s, \vec{y})$, where $\vec{y}$ is a string and s is either $+$ or $-$. If s is $+$ we say the requirement is **positive**, and if s is $-$ the requirement is **negative**. A **glued string** is a triple $\gamma = \langle L, \vec{x}, R\rangle$, where $\vec{x}$ is a string, and L and R are sets of requirements. L is called the **left requirement** and R the **right requirement** of γ.*

A string $\vec{x}$ may be considered if necessary as the glued string $\langle \varnothing, \vec{x}, \varnothing\rangle$. Conversely, a glued string $\langle L, \vec{x}, R\rangle$ may be considered as the string $\vec{x}$, by stripping the context conditions.

The notion of a glued string is an auxiliary one. In a sense, we can only observe "unglued strings". The gluiness shows up in the context restrictions.

Definition 2.3 (Occurrence) *Let $\vec{x}$ and $\vec{y}$ be strings. An occurrence of $\vec{x}$ in $\vec{y}$ is a pair $o = \langle \vec{u}, \vec{v} \rangle$ of strings such that $\vec{y} = \vec{u}\vec{x}\vec{v}$. If $o_1 = \langle \vec{u}_1, \vec{v}_1 \rangle$ is an occurrence of $\vec{x}_1$ in $\vec{y}$, and $o_2 = \langle \vec{u}_2, \vec{v}_2 \rangle$ is an occurrence of $\vec{x}_2$ in $\vec{y}$, we say that o_1 is **to the left of** o_2 (and o_2 is **to the right of** o_1) if $\vec{u}_1\vec{x}_1$ is a prefix of $\vec{u}_2$; that o_1 is **immediately to the left of** or **left adjacent to** o_2 (and o_2 is **immediately to the right of** or **right adjacent to** o_1) if $\vec{u}_2 = \vec{u}_1\vec{x}_1$. o_1 and o_2 are said to be **contiguous** if o_1 is immediately to the right or immediately to the left of o_2. If o_1 is neither to the left or to the right of o_2, the two occurrences **overlap**.*

*Let $\gamma = (L, \vec{x}, R)$ be a glued string. A (string) **occurrence** of γ in $\vec{y}$ is a pair $\langle \vec{u}, \vec{v} \rangle$ such that*

1. *if L is not empty then for some $(+, \vec{w}) \in L$, $\vec{w}$ is a suffix of $\vec{u}$;*
2. *for no $(-, \vec{w}) \in L$, $\vec{w}$ is a suffix of $\vec{u}$;*
3. *if R then for some $(+, \vec{w}) \in R$, $\vec{w}$ is a prefix of $\vec{v}$;*
4. *for no $(-, \vec{w}) \in R$, $\vec{w}$ is a prefix of $\vec{v}$.*

We extend the definitions of precedence, immediate precedence, contiguity and overlap to occurrences of glued strings in the natural way. In this way, the requirements restrict the context of the glued string. Notice the asymmetry. If there is no positive requirement that actually means that no positive restriction applies. Likewise, if there is no negative restriction, then none applies. However, positive conditions are taken disjunctively: each of them is a separate option. Negative conditions on the other hand are taken conjunctively: all of them must be satisfied. Hence the case of an empty list of positive conditions must be taken separately.

This definition of occurrence is instrumental in fixing the concatenation of glued strings. Notice that the condition $(+, \varepsilon)$ is trivial; and that the condition $(-, \varepsilon)$ is unsatisfiable. Both cases are therefore tacitly excluded. They are not meaningless, but rather unhelpful.

Suppose now that we will combine two glued strings. The result shall be a glued string again. Intuitively, one might therefore adopt the following definition.

Given two glued strings, $\gamma_1 = \langle L_1, \vec{x}, R_1 \rangle$ and $\gamma_2 = \langle L_2, \vec{y}, R_2 \rangle$ the concatenation $\gamma_1 \circ \gamma_2$ is defined iff:

1. If R_1 is not empty then for some $(+, \vec{u}) \in R_1$: $\vec{u}$ is a prefix of $\vec{y}$;
2. for every $(-, \vec{u}) \in R_1$: $\vec{u}$ is a not prefix of $\vec{y}$;
3. if L_2 is not empty then for some $(+, \vec{v}) \in L_2$: $\vec{v}$ is a suffix of $\vec{x}$;
4. for every $(-, \vec{v}) \in L_1$: $\vec{v}$ is a not suffix of $\vec{x}$.

In case the product is defined we put

(2.1) $$\gamma_1 \circ \gamma_2 := \langle L_1, \vec{x}^\frown\vec{y}, R_2 \rangle$$

Yet this is problematic. The Hungarian instrumental has as one of its many forms /`bal`/, which can only appear after /`b`/. So we posit it to actually be the glued string $\gamma = \langle \{(+, \mathtt{b})\}, \mathtt{bal}, \varnothing \rangle$. Suppose we combine it with the empty string $\langle \varnothing, \varepsilon, \varnothing \rangle$. (There are cases where the instrumental must be applied in this way. The form /`vel`/ of the instrumental is actually also used with personal pronouns and gives, for example, /`velem`/ 'with me', so this story is not so far fetched.) Then, according to the previous definition, the concatenation is undefined. The situation is not uncommon. We shall say, for example, that Hungarian has an empty suffix for the singular. This suffix has no surface requirements for the host string. The instrumental suffix $\langle \{(+, \mathtt{b})\}, \mathtt{bal}, \varnothing \rangle$, however, requires the host string to contain the suffix /`b`/, and so cannot be added directly to the singular affix. This may be a desirable outcome in that the orthographic conventions block the application of rules. But there may be situations where this is inappropriate. One such situation is where there is an allomorph which can attach only to the empty string. Then it is this allomorph that is chosen when the singular combines with the instrumental but it will not be chosen if the instrumental is attached to the full noun. Notice that the contact requirements are in this definition not exactly requirements about the surface string; rather, they tell us about the restrictions at the point of combination, which is something that invites trickery on the part of a grammer writer. Thus, we adopt here the true surface variant of concatenation, which runs as follows. Let as before $\gamma_1 = \langle L_1, \vec{x}, R_1 \rangle$ and $\gamma_2 = \langle L_2, \vec{y}, R_2 \rangle$ be given. Then $\gamma_1 \curlywedge \gamma_2$ is defined iff

1. If R_1 is not empty then for some $(+, \vec{u}) \in R_1$: $\vec{u}$ is a prefix of $\vec{y}$ *or* $\vec{y}$ is a prefix of $\vec{u}$;

2. for every $(-, \vec{u}) \in R_1$: $\vec{u}$ is not a prefix of $\vec{y}$ *nor* is $\vec{y}$ a prefix of $\vec{u}$;
3. if L_2 is not empty then for some $(+, \vec{v}) \in L_2$: $\vec{v}$ is a suffix of $\vec{x}$ *or* $\vec{x}$ is a suffix of $\vec{v}$;
4. for every $(-, \vec{v}) \in L_1$: $\vec{v}$ is not a suffix of $\vec{x}$ *nor* is $\vec{x}$ a suffix of $\vec{v}$.

So, basically, we also take into account the case where the added string is shorter than the string in the requirement in which case it has to match as well as it can. If the concatenation is defined, the result will be

$$\gamma_1 \wedge \gamma_2 := \langle L_1 \cup L_2^\star, \vec{x}^\frown\vec{y}, R_2 \cup R_1^\star \rangle \tag{2.2}$$

The sets $L_2^\star$ and $R_1^\star$ are yet to be defined. Let $\vec{v}$ and $\vec{w}$ be strings. Then $\vec{v}/\vec{w}$ is defined iff $\vec{w}$ is a suffix of $\vec{v}$ and equals $\vec{z}$, where $\vec{v} = \vec{z}\vec{w}$. Similarly, $\vec{w}\backslash\vec{v}$ is defined iff $\vec{w}$ is a prefix of $\vec{v}$ and equals $\vec{z}$ where $\vec{v} = \vec{w}\vec{z}$.

$$\begin{aligned} L_2^\star &:= \{(s, \vec{u}/\vec{x}) : (s, \vec{u}) \in L_2\} \\ R_1^\star &:= \{(s, \vec{y}\backslash\vec{u}) : (s, \vec{u}) \in R_1\} \end{aligned} \tag{2.3}$$

To be exact, if $\vec{u}/\vec{x}$ is undefined, then the pair $(s, \vec{u}/\vec{x})$ does not exist and is thus not added to the set. It contains all the 'leftover' conditions that are not yet satisfied through the string $\vec{x}$ (since it is too short). Likewise, $R_1^\star$ contains the leftover prefix conditions that $\vec{y}$ did not fully satisfy because of its length. With these definitions in place it is easy to show the following.

Proposition 2.4 *Let γ_1 and γ_2 be glued strings and $\vec{y}$ a string. If $\vec{y}$ contains an occurrence of $\gamma_1 \wedge \gamma_2$ then $\vec{y}$ also contains an occurrence of γ_1 and a contiguous occurrence of γ_2. Conversely, if $\vec{y}$ contains an occurrence of γ_1 with a contiguous occurrence of γ_2, then it contains an occurrence of $\gamma_1 \wedge \gamma_2$.*

Example 1. Consider the instrumental in Hungarian. Its basic form is /`val`/ and /`vel`/, where the vowel is chosen according to vowel harmony, discussed in Section 2.1. Additionally, the /`v`/ appears only when the ending is suffixed to a stem ending in a vowel (/`adó`/ 'tax' → /`adóval`/ 'with tax'). If the stem ends in a consonant, the suffix changes to whatever that consonant is. Thus we have /`ház`/ 'house' → /`házzal`/ 'with (the) house' /`fal`/ 'wall' → /`fallal`/ 'with (the) wall'

and so on. Thus, we say that the form /bal/ actually is a glued string since it can only be suffixed to strings ending in /b/.

(2.4) $(\{(+, \mathtt{b})\}, /\mathtt{bal}/, \varnothing)$

Similarly we have

(2.5) $(\{(+, \mathtt{v}), (+, \mathtt{a}), (+, \mathtt{á}), (+, \mathtt{o}), (+, \mathtt{ó}), (+, \mathtt{u}), (+, \mathtt{ú}), (+, \mathtt{e}), (+, \mathtt{é}), (+, \mathtt{i}), (+, \mathtt{í})\}, /\mathtt{val}/, \varnothing)$

This allows to correctly predict the forms /taxival/, /ragúval/, /sávva/. However, for reasons of vowel harmony we also need the entry

(2.6) $(\{(+, \mathtt{b})\}, /\mathtt{bel}/, \varnothing)$

as well as this one for /vel/:

(2.7) $(\{(+, \mathtt{v}), (+, \mathtt{ö}), (+, \mathtt{ő}), (+, \mathtt{ü}), (+, \mathtt{ű}), (+, \mathtt{e}), (+, \mathtt{é}), (+, \mathtt{i}), (+, \mathtt{í}), (+, \mathtt{y})\}, /\mathtt{vel}/, \varnothing)$

It is to be noted, though, that we get two forms: /vízzel/ and /vízzal/. This cannot be improved upon. This is because the effect of vowel harmony is a long distance effect and thus not describable in terms of contact restrictions. In the next section we shall propose a way to deal with vowel choice. We may alternatively think of the ending to be just /al/ or /el/, and that a /v/ is added if the stem ends in a vowel, and otherwise the last consonant is reduplicated. From a combinatorial point of view, both are equivalent.

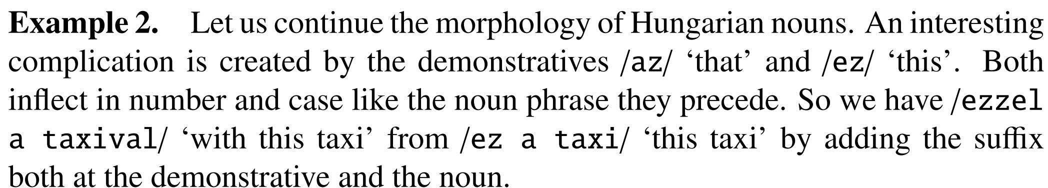
Let us pursue the first alternative. In the present system there is no way to insert abstract characters; once again this is a choice of convenience rather than principle. Thus, the instrumental has in our dictionary several dozen allomorphs. The list so far is this. (There actually are still more entries, but to explain their need will require more understanding of Hungarian morphology, which we will supply further down. See also the exercises to this section.)

(2.8)
/bal/, /bel/, /cal/, /cel/, /dal/, /del/, /fal/, /fel/,
/gal/, /gel/, /hal/, /hel/, /jal/, /jel/, /kal/, /kel/,
/lal/, /lel/, /mal/, /mel/, /nal/, /nel/, /pal/, /pel/,
/ral/, /rel/, /sal/, /sel/, /tal/, /tel/, /val/, /vel/,
/xal/, /xel/, /zal/, /zel/, /val/, /vel/.

Thus, for the form /bal/ we actually propose to analyse it as the glued string $\langle\{(+, \mathtt{b})\}, \mathtt{bal}, \varnothing\rangle$. This can be added to a nonempty string only if it ends in /b/. The condition is then fulfilled and may be removed. (The fact that it also requires the preceding part contains certain vowels is dealt with in another way.) It can be added to an empty word, but then the glue-requirement is passed on to the complex item.

Notice that positive specifications are disjunctive, negative ones conjunctive. This is because it makes no sense to require, for example, that some string has a suffix of the form $\vec{x}$ *and* a suffix of the form $\vec{y}$. Because if we did, then inevitably this can be satisfied only if either $\vec{x}$ is a suffix of $\vec{y}$, or $\vec{y}$ a suffix of $\vec{x}$. For example, suppose we have a left requirement of the form $\{(+, \mathtt{ex}), (+, \mathtt{ur})\}$, to be read in conjunction. Then the glued string must be a suffix to a string ending *both* in /ex/ and /ur/. But this can clearly not be the case. Similarly for right requirements.

If $\vec{y} = \mathtt{x}$, then it is $\vec{x}$ that is the stronger condition, and $(+, \vec{y})$ may be dropped. If $\vec{y} = \mathtt{hex}$, then $(+, \vec{x})$ is weaker and can be dropped. Notice that if the positive list is empty it is treated as no condition. This is just a convention to make life easier.



Example 2. Let us continue the morphology of Hungarian nouns. An interesting complication is created by the demonstratives /az/ 'that' and /ez/ 'this'. Both inflect in number and case like the noun phrase they precede. So we have /ezzel a taxival/ 'with this taxi' from /ez a taxi/ 'this taxi' by adding the suffix both at the demonstrative and the noun.

The demonstrative /ez/ also has many forms, depending on the first letter of the noun. Before a vowel, it is /ez/. However, before a consonant the /z/ assimilates. Thus, we get the forms /ebben/ 'in the house' (in place of */ezben/) and /ettől/ 'from this' (in place of */eztől/). When the demonstrative appears in the instrumental case, however, it is both the /z/ of the demonstrative that may assimilate, giving /evvel/ 'with this' or it may be the /v/ of the suffix that may assimilate, giving the (more popular) /ezzel/. However, the vowel never assimilates: /az/ 'that' is a different morpheme from /ez/ 'this'.

When we propose the morpheme to consist of glued strings just like the instrumental, we encounter a difficulty. The form /ebbel/ 'with this' is correct for

the glued strings, contrary to fact! This cannot be accounted for on the basis of glued strings, for we have no notion of **base**, or **neutral form**. ⚽

The previous example showed that there is no way around specifying a neutral (or default) form. For the glued strings leave too many options open in this case. It is unclear, though, to what extent there are general rules governing these cases of underdetermination. In the present case we can stipulate, for example, that one of the forms must be a neutral form.

The suffix also lengthens some of the vowels of the stem: /medve/ 'bear' → /medvével/ 'with [the] bear'. This lengthening is represented in the orthography by a different character. We shall return to this phenomenon in the next section.

There are other things to note about Hungarian in a truly surface oriented account. The language has a numbers of digraphs: /gy/ represents the sound [dj], /sz/ the sound [s], /cs/ the sound [tʃ], and /ly/ the sound [j]. When two identical digraphs are next to each other, the last letter from the first digraph is dropped: /gy/+/gy/ becomes /ggy/. Furthermore, there is no sequence of three identical digraphs. Any such sequence would be simplified to two occurrences. If we want to account for these facts we would have to assume that the word /hegy/ 'hill' has another form, /heg/; and that the instrumental chooses to be /gyel/ in this particular case. This is less than optimal. We think it is best to relegate such wrinkles of the orthography into the preprocessing of the string. Thus, we think that the actual string to deal with is /hegygyel/ and not /heggyel/. Likewise, we shall relegate the convention of choosing to begin a sentence with an upper case characters to the preprocessing. For the purpose of the next definition recall that a regular relation is one that is defined by a finite state transducer.

Definition 2.5 *There is a level of **deep orthography** at which every string operation is string concatenation. We assume that the relation between deep and surface orthography is regular.*

To pass from surface orthography to deep orthography we need to apply some function, called **preprocessing**.

Notes on this section. The relation between deep and surface orthography is not straightforward. One quite difficult problem is to decide when to replace a word by its upper case equivalent. Not all periods end a sentence, so the period is not a good diagnostic. In fact, only the sentence structure is. But the sentence structure

is not present in the string. Thus prima facie we seem to require a much more powerful machine to detect the end of a sentence. Probably the best approach is to simply disambiguate the period in the deep orthography.

Exercise 1. It is possible to generalize the notion of a glued string even further. For example, define a glued string to be a triple $(L, \vec{x}, R)$, such that L and R are regular languages. $\vec{x}$ has an occurrence in $\vec{y}$ if $\vec{y} = \vec{u}\vec{x}\vec{v}$ with $\vec{u} \in L$ and $\vec{v} \in R$. Specify the details of this extension. (The reason for not choosing this approach is that it eliminates the locality of the contact phenomena.)

Exercise 2. There is a way to deal with the reduplication that does not require an additional level. First, observe that nouns already ending in a reduplicated consonsnat like /`meggy`/ 'sour cherry' get the form /`meggyel`/, likewise /`vicc`/ 'joke' gets the form /`viccel`/. Formulate additional morphs in case the root already ends in a duplicate consonant. For in that case, no more consonant is added. Second, provide an alternate form for nouns such as /`hegy`/ that allows to combine only to get /`heggyel`/.

2.3 Morphological Classes

We have deferred the problem of how to deal with long distance effects and abstract word classes. As the exercises of the last section indicate, long distance effects could be treated in a different way. The fact remains, though, that they are not contact effects. Of course, glued strings will only represent true contact effects if the requirements only contain strings of length 1, but in view of graphematic complications (digraphs etc.) it would be counterproductive to implement such an extreme position. Thus, a phenomenon counts as a long distance effect only if an effective bound on the length of the influencing context cannot be given. So, long distance effects require a different strategy. In addition, many morphological rules cannot be described purely in terms of requirements on strings.

Example 3. There are two forms of the plural of the word /`Bank`/. One is /`Banken`/ and the other is /`Bänke`/. However, /`Bank`/ can mean both 'bank' in

the sense of a financial institution and 'porch'. If it means the first the plural is /Banken/, if it means the second the plural is /Bänke/. No string algorithm is able to handle these facts purely without taking meanings into account. ⚽

Example 4. In Hungarian, the vowels /e/, /é/, /i/ and /í/ are neutral, that is, they can cooccur with both series of harmonizing vowels. There are words that consist only of neutral vowels. For these words it is not clear which of the forms of a suffix should be taken. There is no available pattern. It is /a vízben/ 'in the water' but not /a vízban/; however, it is /az íjjal/ 'with the bow' and not /az íjjel/. (See also Lass 1984.)

Vowel harmony is not entirely phonological. It ends at the word boundary. This becomes important in compounding. The noun /vezérlőpult/ 'command console' is well-formed even though the two parts, /vezér/ 'commander' and /pult/ 'desk', have different harmony. Thus, not all morphemes have ingoing harmony value equal to outgoing harmony vowel. ⚽

Therefore, additional mechanisms are needed. One such mechanism is the introduction of **morphological classes**. Morphological classes are properties of individual morphs, not of morphemes. The classes control the behaviour of a morph under combination. We assume that when two morphs are combined one of them takes the role of the argument and the other is the functor that takes the other one to give a result. When a morph m_1 takes a morph m_2 as its argument they together produce a third morph, m_3. Hence, we have at least three classes to deal with: the class of m_1, the class of m_2 and the class of m_3. The way this is accounted for is as follows. We assume that m_1 actually has *two* classes associated with it. These are the **ingoing class** of m_1 and the **outgoing class** of m_1. The ingoing class of m_1 declares what class the argument must have in order to be combinable with m_1. The outgoing class specifies what class the combination of m_1 with its argument m_2 has.

Example 5. Vowel harmony in Hungarian is achieved by assuming two kinds of harmony values: F ("front") and B ("back"). There are phonological restrictions governing their distribution. As we have seen in Example 4, they do not exhaust the conditions. Hence, we associate with a root noun either of the classes F and B. Thus, /víz/ has class F, /íj/ has class B. The affixes always have ingoing class equal to outgoing class. In particular, /vel/ (and its alternative forms in /e/) have

ingoing (and outgoing) class F, the other forms have ingoing (and outgoing class B. This allows to control the distribution of the case forms of the instrumental. ⚽

The combinatorial properties of morphological classes can be rather complex. The number of such classes can be rather large. For example, to correctly determine the perfect form of a Latin verb we need various different classes (see Matthews 1978). Also, German nouns need plenty of annotation so that we know which of the suffixes will trigger umlaut, what infix to take when compounding them, and so on. Classes can serve multiple purposes. As the multitude of classes can be bewildering we need a mechanism that allows for specific control. One tool to manage the complexity is that of an **attribute value matrix** or **AVM**. It has the following form.

$$(2.9)\qquad \begin{bmatrix} \text{ATTRIBUTE}_1 : \mathit{valueset}_1 \\ \text{ATTRIBUTE}_2 : \mathit{valueset}_2 \\ \cdots \\ \text{ATTRIBUTE}_n : \mathit{valueset}_n \end{bmatrix}$$

If $n = 0$, the AVM is said to be **empty**. We write [ ] for the empty AVM. In (2.9), ATTRIBUTE_i is some arbitrary name, and $\mathit{valueset}_i$ is a set of names for values. For example, we may have $[\text{PERSON} : \{\mathit{1}, \mathit{2}\}]$. This says that the value of the person feature for this element is either *1* or *2*, whatever that may mean. We shall use also a different notation which reveals the logical character of the notation. And this is to use the disjunction $\vee$ and conjunction $\wedge$. Values as well as attribute value pairs can be combined with these symbols. In place of

$$(2.10)\qquad [\text{ATT} : \{v_1, \cdots, v_p\}]$$

We may write

$$(2.11)\qquad [\text{ATT} : v_1 \vee \cdots v_p]$$

The rules for disjunction are

$$(2.12)\qquad [\text{ATT} : p \vee q] \equiv [\text{ATT} : p] \vee [\text{ATT} : q]$$

where p and q stand for values and disjunctions thereof and $\equiv$ denotes logical equivalence. Thus, sets are shorthands for the disjunction of their elements. They describe the uncertainty. The more elements the less defined the entity is. However, if an attribute is absent, this means total lack of specification. There is an alternative notation and it is this:

$$(2.13)\qquad [\text{ATT} : \top]$$

Hence we have (by definition)

$$\left[\begin{array}{l}\text{ATT}_1{:}\alpha_1 \\ \text{ATT}_2{:}\ \top\end{array}\right] \equiv [\text{ATT}_1 : \alpha_1] \tag{2.14}$$

As in GPSG and later developments, each attribute *a* comes with a given set of values that it may take, called its **range**, denoted by rg(a). The range of an attribute is usually finite, and known in advance. The atttribute NUM may have values *singular* and *plural* in English, French and German, but *singular*, *dual*, *plural* in Sanskrit and Ancient Greek. It is part of the grammar to specify the range. It follows then that in English we have the following equation

$$[\text{NUM} : \top] \equiv [\text{NUM} : \textit{sing} \vee \textit{pl}] \tag{2.15}$$

In Sanskrit, however, we have instead

$$[\text{NUM} : \top] \equiv [\text{NUM} : \textit{sing} \vee \textit{pl} \vee \textit{dual}] \tag{2.16}$$

A special case needs to be considered, namely the empty set. The fact that an attribute receives the empty set as value means that we have an empty disjunction, which by definition is equivalent to falsum (notation $\bot$). So

$$[\text{ATT} : \varnothing] \equiv [\text{ATT} : \bot] \tag{2.17}$$

The brackets on the other hand represent conjunction. So

$$\left[\begin{array}{l}\text{ATT}_1{:}p \\ \text{ATT}_2{:}q\end{array}\right] = [\text{ATT}_1 : p] \wedge [\text{ATT}_2 : q] \tag{2.18}$$

The usual laws of logic apply, for example distribution. For example,

$$\begin{aligned}\left[\begin{array}{l}\text{ATT}_1{:}s \vee s' \\ \text{ATT}_2{:}\quad t\end{array}\right] &= [\text{ATT}_2 : t] \wedge ([\text{ATT}_1 : s] \vee \text{ATT}_2 : s']) \\ &= ([\text{ATT}_2 : t] \wedge [\text{ATT}_1 : s]) \vee ([\text{ATT}_2 : t] \wedge [\text{ATT}_1 : s']) \\ &= \left[\begin{array}{l}\text{ATT}_1{:}s \\ \text{ATT}_2{:}t\end{array}\right] \vee \left[\begin{array}{l}\text{ATT}_1{:}s' \\ \text{ATT}_2{:}\ t\end{array}\right]\end{aligned} \tag{2.19}$$

There is also a value $\top$ to denote 'any value'. It requires that the function f is defined. In the present implementation, however, you can do both. You have the symbol $\top$ but you may leave values unspecified.

We may think of the AVM (2.9) as a function f from attributes to value sets such that $f(a)$ is a subset of the value range of a, $\mathrm{rg}(a)$. If the AVM does not specify a value, we take this to mean that the value is $\mathrm{rg}(a)$. For later use we shall codify this in the following definitions.

Definition 2.6 (Feature Space) *A **feature space** is a triple $\mathcal{S} = \langle A, V, \mathrm{rg}\rangle$, where A is a finite set, the set of **attributes**, V a finite set, the set of **values**, and $\mathrm{rg} : A \to \wp(V)$ a function such that for all $a \in A$, $\mathrm{rg}(a) \neq \varnothing$. An $\mathcal{S}$-**matrix** is a partial function $f : A \hookrightarrow \wp(V)$ such that for all $a \in A$ $f(a) \subseteq \mathrm{rg}(a)$. If f and g are $\mathcal{S}$-matrices we write $f \leq g$ iff for all $a \in A$, either*

- *$f(a)$ and $g(a)$ are undefined or*
- *$f(a)$ is defined, $g(a)$ is undefined and $f(a) \neq \varnothing$ or*
- *$f(a)$ is undefined, $g(a)$ is defined and $g(a) = \mathrm{rg}(a)$ or*
- *$f(a)$ and $g(a)$ are both defined and*
  - *$f(a) = g(a) = \varnothing$ or*
  - *$f(a) \neq \varnothing \neq g(a)$ and $f(a) \subseteq g(a)$.*

*An $\mathcal{S}$-**atom** is an $\mathcal{S}$-matrix which is minimal with respect to $\leq$.*

Notice the slight change in terminology. An $\mathcal{S}$-matrix is an AVM, though it is an AVM that satisfies the restrictions of $\mathcal{S}$. We shall always assume that AVMs are drawn from a common feature space, though that assumption is in practice not necessary.

We have allowed f to be a partial function. This is a good idea for notational purposes, but when dealing with proofs and definitions it is actually cumbersome. We shall define the *completion* f^+ of an AVM f, and AVM that is not partial, and show that f^+ is equivalent to f in terms of the ordering $\leq$.

Definition 2.7 (Completion) *Let f be an $\mathcal{S}$-matrix. The **completion** of f is that $\mathcal{S}$-matrix g which is a function extending f such that if $f(a)$ is undefined, $g(a) = \mathrm{rg}(a)$. We write f^+ for the completion of f.*

In other words, for an attribute a, either f is defined on a and $f^+(a) = f(a)$ or f is not defined and $f^((a) = \mathrm{rg}(a)$. It is clear that the completion is uniquely defined. Two AVMs f and g are equivalent iff they have the same completion.

Lemma 2.8 *$f \leq g \leq f$ if and only if $f^+ = g^+$.*

This is not difficult to see. Assume $f \leq g$ and $g \leq f$. Pick an attribute a.

1. Both f and g are undefined on a. Then $f^+(a) = \mathrm{rg}(a)$ and $g^+(a) = \mathrm{rg}(a)$, hence $f^+(a) = g^+(a)$.

2. f is undefined on a while $g(a)$ is defined. Then $f^+(a) = \mathrm{rg}(a) \neq \varnothing$. Now, $f \leq g$ requires $g(a) = \mathrm{rg}(a)$, and so $g^+(a) = \mathrm{rg}(a)$ as well.

3. $f(a)$ is defined and g is undefined. Similar to the previous case.

4. Both f and g are defined. Then either they are both empty (and hence identical) or $f(a) \subseteq g(a)$ (from $f \leq g$) and $g(a) \subseteq f(a)$ (from $g \leq f$). Hence $f(a) = g(a)$, from which $f^+(a) = g^+(a)$.

Now assume conversely that $f^+ = g^+$. We have to show that $f \leq g$. (The case $g \leq f$ is similar.) Again, several cases arise.

1. f and g are undefined. That case is trivial.

2. f is undefined on a while $g(a)$ is defined. Then $f^+(a) = \mathrm{rg}(a) \neq \varnothing$. Since $f^+(a) = g^+(a)$ and $g(a) = g^+(a)$, we have $g(a) = \mathrm{rg}(a)$.

3. g is undefined on a while $f(a)$ is defined. Then $g^+(a) = \mathrm{rg}(a)$, and $f(a) \neq \varnothing$, for otherwise $f^+(a) = \varnothing \neq \mathrm{rg}(a)$.

4. Both f and g are defined. Then $f^+(a) = f(a)$ and $g^+(a) = g(a)$. Then either they are both empty (and hence identical) or $f(a) = g(a)$, from which $f(a) \subseteq g(a)$.

So, $f \leq g$ is satisfied.

The **unification** $f \wedge g$ of two complete AVMs f and g is defined as follows.

$$(2.20)\qquad (f \wedge g)(a) := \begin{cases} f(a) \cap g(a) & \text{if } f(a) \cap g(a) \neq \varnothing \\ \varnothing & \text{if } f(a) = g(a) = \varnothing \\ \text{undefined} & \text{otherwise} \end{cases}$$

Since the result is not always a function, it is not always defined. Hence $f \wedge g$ is defined iff for all a: either $f(a) = g(a) = \varnothing$ or $f(a) \cap g(a) \neq \varnothing$.

Logically, $[a : \bot]$ means that a has *no value*. Alternatively, we write $[a : \star]$, but it must be emphasised that $\star$ is not a value, but rather the empty set. Since a may either *have* a value or be undefined, we have the following:

$$(2.21)\qquad \top \equiv [a : \top] \vee [a : \bot]$$

This is not a contradiction. For example, a root noun does have no case, its case value is undefined. This is different from an unknown value. A noun has no case before a case suffix is added, so we use $\star$ to represent that. The combinatorics of $\star$ are unusual, because the empty set displays a special behaviour. It unifies only with $\star$, and it cannot cooccur with any other value (by definition!). So, $[\textsc{case} : \{\star, nom\}]$ is not appropriate. Essentially, $[\textsc{case} : \star]$ means that CASE is undefined on the item and is logically equivalent with $[\textsc{case} : \bot]$. Keep in mind, therefore, that not all attributes are defined.

Analyzing AVMs from a logical perspective, we understand a as a modal operator $\Box$. It satisfies the following axiom: $\Diamond p \rightarrow \Box p$. Equivalently, this logic (known as Alt_1) can be axiomatised by

$$(2.22)\qquad \Diamond p \wedge \Diamond q \leftrightarrow \Diamond(p \wedge q)$$

Additionally, since features cannot be stacked, we have the axiom $\Box\Box\bot$. (In the terminology of Gazdar et al. 1985, we assume here that there are no type 1 features. This is a choice of convenience rather than principle.) The following is also generally true of normal operators:

$$(2.23)\qquad \Diamond p \vee \Diamond q \leftrightarrow \Diamond(p \vee q)$$

Now, read $[a : \{v_1, \cdots, v_n\}]$ as $\Diamond(v_1 \vee v_2 \vee \cdots \vee v_n)$, where the v_i are propositional constants. When $n = 0$ we get the special case $\Diamond\bot$, which is the same as $\neg\Box\top$. Since $\Box\top \equiv \top$, $\neg\Box\top \equiv \neg\top \equiv \bot$, a contradiction. The translation of $[a : \star]$ is on the other hand $\neg\Diamond\top$, which says that there is no successor.

Assume for each attribute a a distinct modal operator $\langle a \rangle$ with logic Alt_1. Further, we have $\neg\langle a \rangle\langle a' \rangle\top$, $a, a' \in A$ (not necessarily distinct). Assume for each $u \in V$ a constant p_u such that $p_u \to \neg p_v$ for $u \neq v$. The feature space $\mathcal{S}$ translates into the following axiom.

$$\mathcal{S}^\tau := \bigwedge_{a \in A} \left(\left(\bigvee_{u \in \mathrm{rg}(a)} \langle a \rangle p_u \right) \wedge \bigwedge_{u \in V - \mathrm{rg}(a)} \neg\langle a \rangle p_u \right) \tag{2.24}$$

This defines a logic $L_\mathcal{S}$ of $\mathcal{S}$-matrices, see Kracht 1993). Translate an AVM as follows.

$$f^\tau := \bigwedge_{a \in A, f(a)\uparrow} \left(\left(\bigvee_{u \in \wp(f(a))} \langle a \rangle p_u \right) \wedge \bigwedge_{a \in V - \wp(f(a))} \neg\langle a \rangle p_u \right) \tag{2.25}$$

Now, the following is easy to show.

Proposition 2.9 *Let f and g be $\mathcal{S}$-matrices. Then $f \leq g$ iff $L_\mathcal{S} \vdash f^\tau \to g^\tau$.*

Although we have used logic to decompose the AVMs, it is the notation (2.9) that we shall use throughout. This is somewhat limiting in the way facts are represented. For example, the verbal suffix /en/ in German may signal various things: it signals the 1st or 3rd plural or the infinitive. This can only be written down as a disjunction where the first two can be grouped together. So, effectively it means that we need two different morphs, one for the person agreement features, and one for the infinitive.

Also, note that there is no AVM to represent $\bot$. This is actually not a bad idea, as is allows us to say that AVMs unify if and only if there is an AVM that they both subsume (see the exercises below). In Section 3.4 we shall be considering pairs of AVMs, where one of the two (or even both) may be absent. It may be deemed necessary to invoke an arbitrary empty AVM (perhaps $\bot$) to be put there. However, the mechanics of diacritics introduced in that section circumvent the need to do so.

Exercise 3. Let T be a finite set of attributes. Define $A \sqsubseteq B$ as follows. $A \sqsubseteq B$ if and only if for every α, if A contains $[\alpha : \sigma]$ then either B does not contain α or it contains $[\alpha : \sigma']$ and $\sigma \subseteq \sigma'$. Then A and B *unify* if and only if there is a C such that $C \sqsubseteq A$ and $C \sqsubseteq B$. Show that this is problematic. Specifically, the empty set creates problems.

Exercise 4. (Continuing the previous exercise.) This exercise explains the rationale for positing a value $\star$. Let $f(\alpha)$ be a finite set for each attribute α. Let $[A]$ be defined to be the product $\prod_{\alpha \in T} \pi_A(\alpha)$, where

1. if α is not contained in A, $\pi_A(\alpha) := f(\alpha)$;
2. if $\alpha : \varnothing$ is contained in A, then $\pi_A(\alpha) := \{\star\}$;
3. if $\alpha : \sigma$ is contained in A and $\sigma \neq \varnothing$ then $\pi_A(\alpha) := \sigma$.

Put $A \subseteq B$ if and only if $[A] \subseteq [B]$. Now say that A and B unify if $[A] \cap [B] \neq \varnothing$. Furthermore, the unification is that C for which $[C] = [A] \cap [B]$.

Exercise 5. Show that there are AVMs A and B such that no C exists for which $[C] = [A] \cup [B]$.

Exercise 6. Show that $f \wedge g$ as defined above is the greatest lower bound of f and g with respect to the order $\leq$.

Exercise 7. Show that $f \leq g$ iff every atom below f is below g.

2.4 Discontinuity

In the best of all worlds, morphs are affixes: some sequences of letters that are combined by placing them next to each other. However, there are many examples which show that this is not so.

Example 6. The perfect form of regular verbs in German is formed by prefixing the root with /`ge`/ and adding one of the following: /`t`/, /`en`/. Thus we have /`gefahren`/ ‘driven’, from /`fahr`/ ‘drive’ (the root form) by adding the prefix /`ge`/ *and* the suffix /`en`/. Similarly, we have /`gesucht`/ ‘searched’ from /`such`/ ‘look for, search’ (root form). The latter shows that the perfect stem is *not* derived from the infinitive, which is obtained by adding the suffix /`en`/ (/`fahren`/ ‘to drive’

and /suchen/ 'to search'). Moreover, many verbal roots allow prefixes. These originate from prepositions (/über/ 'above') or are altogether genuinely distinct (/ver/, /zer/, /be/). Prefixes of the second kind generally have no semantics of their own; the root together with its prefix are an idiom (compare /verwalten/ 'to administrate' and /walten/ 'to act', /versuchen/ 'to try' and /suchen/ 'to look for'). Prefixes come in two kinds: separable and inseparable. If the prefix is separable, it will stay in final position of the clause. We have /Johan las die Krümel auf./ 'Johan collected the crumbs.', using the verb /auflesen/ 'to collect (by picking up)'. If the prefix is inseparable, it moves with the root to second position: /Johann überging diesen Punkt./ 'Johan skipped that point.', using /übergehen/ 'to skip'. To form the perfect with these verbs, again the division into separable and inseparable plays a role. If the prefix is separable, the first part of the perfect morph, /ge/, is omitted: /übergangen/ 'to have skipped'. If the prefix is separable, the part /ge/ squeezes itself between the prefix and the root: /aufgelesen/. This latter form is bimorphemic but it consists of four parts: /auf/, /ge/, /les/ and /en/, of which the first and the third form the verb and the second and fourth the perfect morph. ⚽

Example 7. In Arabic, roots are typically triconsonantal. Let us look at Egyptian Arabic (examples are drawn from Fromkin 2000). For example, there is /ktb/ 'to write', and /drs/ 'to study'. From these roots, actual words are made by inserting vowels and some more consonants. (There are even rules doubling material.) Add two times /a/ and you get the the 3rd person singular past tense form: /katab/ 'he wrote'. Analogously, you get /daras/ 'he studied'. More forms: /baktib/ 'I write', /badris/ 'I study', /kaatib/ 'writer'; and /daaris/ 'student'. These forms are highly complex. What is constant is the root element. The other parts are typically not segmentable, although they sometime provide complex morphological information ('3rd singular active past tense indicative'). ⚽

This suggests that morphs may consist of several strings, or rather glued strings. This leads to the following definition.

Definition 2.10 (Fractured Glued String) *A* ***fractured glued string*** *is a sequences of glued strings. If $\gamma_1, \gamma_2, \ldots, \gamma_m$ are glued strings, we write $\gamma_1 \otimes \gamma_2 \otimes \cdots \otimes \gamma_m$ for the fractured glued string $\mathfrak{g}$ consisting of these glued strings in the order specified. γ_i is the ithe* ***section*** *of $\mathfrak{g}$. m is called the* ***dimension*** *of the sequence and written* $\dim \mathfrak{g}$. $\mathfrak{z}$ *('zero') denotes the (unique) fractured string of dimension 0.*

Clearly, '⊗' is fully associative and therefore all brackets can be dropped. Furthermore, we write

$$(2.26) \qquad \bigotimes_{i=1}^{m} \gamma_i := \gamma_1 \otimes \gamma_2 \otimes \cdots \otimes \gamma_m$$

Sometimes, dropping the index we simply write '$\bigotimes \gamma_i$' for a fractured (glued) string.

Definition 2.11 (String Content) *Let $\gamma = (L, \vec{x}, R)$ be a glued string. Then $c(\gamma) := \vec{x}$. Furthermore, $c(\bigotimes_{i<n} \gamma_i) = c(\gamma_0)^\frown c(\gamma_1)^\frown \cdots ^\frown c(\gamma_{n-1})$.*

In what is to follow in this section we shall not make use of the context requirements of glued strings, so they will simply be strings. Also, a fractured string of dimension 1 is not distinguished from the string it contains. This simplifies the notation. So we may now write the German verbal roots as /`auf`⊗`les`/ (with ⊗ inside the slashes), /`übergeh`/ and the perfect suffix morphs as /`ge`⊗`t`/ or /`ge`⊗`en`/, or as /`t`/, /`en`/. (There are no relevant context requirements anyway, so no information is lost anyway.) The Arabic roots are /`k` ⊗ `t` ⊗ `b`/ and /`d` ⊗ `r` ⊗ `s`/.

We need to specify what happens if two glued strings are combined. The way this is done here is by defining combinatorial functions called *handlers*.

Definition 2.12 (Handler) *A **handler** is a sequence $\mathbb{H}$ of sequences of pairs (i, b), where i is a natural number and b a boolean. The members of $\mathbb{H}$ are called its **sections**. A pair (i, b) is said to **occur** in $\mathbb{H}$, in symbols $(i, b) \in \mathbb{H}$, if there is a section of which (i, b) is some member. The pairs occurring in $\mathbb{H}$ are called its **parts**. Parts may have several occurrences. The result of applying $\mathbb{H}$ to the fractured glued strings $\mathfrak{g} = \gamma_1 \otimes \gamma_2 \otimes \cdots \gamma_m$ and $\mathfrak{h} = \eta_1 \otimes \eta_2 \otimes \cdots \otimes \eta_n$, is defined as follows. Put*

$$(2.27) \qquad (i, b)(\mathfrak{g}, \mathfrak{h}) := \begin{cases} \gamma_i & \text{if } b = \text{true} \\ \eta_i & \text{else} \end{cases}$$

Now, for the sequence $h_i = (i_0, b_0), (i_1, b_1), \cdots, (i_{p-1}, b_{p-1})$ we put

$$(2.28) \qquad h_i(\mathfrak{g}, \mathfrak{h}) := (i_0, b_0)(\mathfrak{g}, \mathfrak{h})^\frown (i_1, b_1)(\mathfrak{g}, \mathfrak{h})^\frown \cdots ^\frown (i_{p-1}, b_{p-1})(\mathfrak{g}, \mathfrak{h})$$

Finally, let $\mathbb{H} = (h_0, h_1, \cdots, h_{q-1})$ *have q sections. Then*

(2.29) $$\mathbb{H}(\mathfrak{g}, \mathfrak{h}) := h_0(\mathfrak{g}, \mathfrak{h}) \otimes h_1(\mathfrak{g}, \mathfrak{h}) \otimes \cdots \otimes h_{q-1}(\mathfrak{g}, \mathfrak{h})$$

The sections of a handler define the sections of the fractured glued string; each section therefore composes a glued strings by gluing together the glued strings picked out by its parts.

Notice that applying a specific handler to a fractured glued string may not be defined. For if $\mathbb{H}$ *contains* $(i, true)$*, the dimension of* $\mathfrak{g}$ *must be at least i, and if* $\mathbb{H}$ *contains* $(i, false)$*, the dimension of* $\mathfrak{h}$ *must at least be i.*

Here are some examples. Let

(2.30) $$\mathbb{F} := \langle\langle(0, true), (0, false)\rangle\rangle$$

Then

(2.31) $$\mathbb{F}(\vec{x}, \vec{y}) := \vec{x}^\frown\vec{y}$$

So, $\mathbb{F}$ is plain forward concatenation. Let's exchange *true* and *false*.

(2.32) $$\mathbb{B} := \langle\langle(0, false), (0, true)\rangle\rangle$$

Then

(2.33) $$\mathbb{B}(\vec{x}, \vec{y}) := \vec{y}^\frown\vec{x}$$

This is backward concatenation. A third example:

(2.34) $$\mathbb{W} := \langle\langle(0, true), (0, false), (1, true)\rangle\rangle$$

Then

(2.35) $$\mathbb{W}(\vec{x} \otimes \vec{v}, \vec{y}) := \vec{x}^\frown\vec{y}^\frown\vec{v}$$

This is forward wrapping. All these three handlers have just one section, so they result in a string. Now define the following.

(2.36) $$\mathbb{P} := \langle\langle(0, true)\rangle, \langle(0, false)\rangle\rangle$$

Then

$$(2.37) \qquad \mathbb{P}(\vec{x},\vec{y}) := \vec{x} \otimes \vec{y}$$

This is pairing.

$$(2.38) \qquad \mathbb{D} := \langle\langle(0, \mathit{true}), (0, \mathit{true})\rangle\rangle$$

Then

$$(2.39) \qquad \mathbb{D}(\vec{x}, \mathfrak{z}) := \vec{x}^\frown \vec{x}$$

This is doubling. Notice that doubling is considered a binary function, but it makes no use of its second argument. The second argument is not the empty string (in which case it *would* be a genuine argument) but the empty fractured string.

Variants of wrapping are the following.

$$(2.40) \qquad \begin{aligned} \mathbb{W}_0 &:= \langle\langle(0, \mathit{true})\rangle, \langle(0, \mathit{false})\rangle, \langle(1, \mathit{true})\rangle\rangle \\ \mathbb{W}_1 &:= \langle\langle(0, \mathit{true})\rangle, \langle(0, \mathit{false}), (1, \mathit{true})\rangle\rangle \\ \mathbb{W}_2 &:= \langle\langle(0, \mathit{true}), (0, \mathit{false})\rangle, \langle(1, \mathit{true})\rangle\rangle \end{aligned}$$

They are all slightly distinct in that the output is again a fractured string of dimension 2, containing so to speak a gap at some places, or of dimension 3.

We see that a handler may actually use some sections several times, as many times as the pair (i, b) occurs in it. The definition does *not* require that the handler uses all parts of the fractured glued string, though if it does not then material gets deleted. Thus we introduce the following definition.

Definition 2.13 (Proper Handler) *A handler $\mathbb{H}$ is* ***proper*** *if for all numbers i, j and booleans b, if $\mathbb{H}$ contains (i, b) and $j < i$ then $\mathbb{H}$ also contains (j, b). The* ***dimension*** *of a handler $\mathbb{H}$ is defined by*

$$(2.41) \qquad \dim \mathbb{H} = (\{i : (i, \mathit{true}) \in \mathbb{H}\}, \{(i, \mathit{false}) \in \mathbb{H}\})$$

If $\mathbb{H}$ is proper, $\dim \mathbb{H}$ *is a pair of numbers.*

To understand the last remark, notice that numbers are defined as follows. 0 is the empty set $\varnothing$, and

$$(2.42) \qquad n + 1 := \{0, 1, \cdots, n\}$$

It turns out that for a proper handler $\mathbb{H}$, $\mathbb{H}(\mathfrak{g}, \mathfrak{h})$ is defined if (and only if) $\dim \mathbb{H} = (k, \ell)$ and $k \leq \dim \mathfrak{g}$ as well as $\ell \leq \dim \mathfrak{h}$. This is because the above definition did not require the handlers to use up all parts of the arguments. However, from a linguistic perspective, handlers that omit material are actually detrimental. They should be banned.

> [No Deletion]
> In what is to follow, handlers are always required to be proper. Moreover, $\mathbb{H}(\mathfrak{g}, \mathfrak{h})$ is defined if and only if all sections of both $\mathfrak{g}$ and $\mathfrak{h}$ are used in $\mathbb{H}$. This is the case iff $\mathbb{H}$ is proper and $\dim \mathbb{H} = (\dim \mathfrak{g}, \dim \mathfrak{h})$.

A handler may not add further strings, it does not even contain empty strings. Any letter that is output by a handler must therefore originate in one of the arguments. This is a strict policy, but can be explained as follows. When thinking of, say, a root and an affix, we normally think of the affix as adding itself to the noun. Thus, the affix not only has a string associated with it but also a handler by which it knows whether it is a prefix or a suffix. Thus, the handler is not a third element in combining two strings, but it is part of one of them. This creates an asymmetry. There is one element, the functor, which has a handler associated with it, and another, that does not. This is theoretically an unsatisfactory affair. It disallows a functor to combine with an argument when the argument still needs to combine (as functor) with a third element. To remedy this, we define a morph to always contain a fractured glued string and a handler that is consistent with it. When two such pairs, $(\mathbb{G}, \mathfrak{g})$ and $(\mathbb{H}, \mathfrak{h})$ are combined, we will get an element $(\mathbb{G} \circ \mathbb{H}, \mathbb{G}(\mathfrak{g}, \mathfrak{h}))$, where $\circ$ is the product of the handlers.

Definition 2.14 (Product of Handlers) *Let $\mathbb{G}$ and $\mathbb{H}$ be handlers. The product, $\mathbb{G} \circ \mathbb{H}$, is defined as follows. It is defined only if* $\dim \mathbb{G} = (m, n)$ *and* $\dim \mathbb{H} = (n, k)$ *for some natural numbers m, n, k and yields a handler of dimension $(m + n, k)$. Suppose $\mathbb{H}$ has sections h_i, $i < m$. We first define h_i^{+r}. It is the sequence of all $(i, b)^{+r}$ in the order of appearance in h_i.*

$$(2.43) \qquad (i, b)^{+r} := \begin{cases} (i + r, b) & \text{if } b = \textit{true} \\ (i, b) & \text{otherwise} \end{cases}$$

This operation shifts the argument places.

$$(2.44) \qquad (i, b)^{\blacklozenge} := \begin{cases} \langle (i, b) \rangle & \text{if } b = \textit{true} \\ h_i^{+m} & \text{otherwise} \end{cases}$$

Then for each section g_i of $\mathbb{G}$ let $g_i^\blacklozenge$ be the concatenation of the $(i, b)^\blacklozenge$. $\mathbb{G} \circ \mathbb{H}$ is the sequence of all $g_i^\blacklozenge$.

This is a somewhat lengthy definition that does the following. If $\mathfrak{g}$ and $\mathfrak{h}$ may be combined, we may instead think of $\mathfrak{g} \otimes \mathfrak{h}$ as the functor which expects an argument. The dimension is now $m + n$. The argument that is expected by $\mathfrak{h}$ is still to be expected, and its dimension is retained.

Notice that we have implemented the following notational convention.

[Notational Convention]
In a merge of two constituents C and D, written $\boxslash(C, D)$, C is the head of the constituent, and D its argument in this construction.

This does not mean that heads universally precede their complements in surface syntax. All it does say is that in writing the term $\boxslash(C, D)$ it is always C which is the head, and D which is the complement. This is the canonical notation. Of course it depends on the handler of C what happens to D. If both are simply strings $\vec{x}$ and $\vec{y}$, then $\vec{x}$ may precede or follow $\vec{y}$, depending on what the handler of C says (the handler of D is irrelevant).

With discontinuous constituents the notion of directionality gets tricky. We define it as follows.

Definition 2.15 (Directionality) *A handler is **consumptive** if it contains an element of the form (k, false). A handler $\mathbb{H}$ is **rightward** if it is consumptive and the first element of the first nonempty sequence has the form (k, true) for some k; it if **leftward** if it is consumptive and the first element of the first nonempty sequence has the form $(k, \mathit{false}$ for some k. If a handler is neither leftward nor rightward, it is also called **neutral**.*

A consumptive handler is a handler such that $\mathbb{H}(\mathfrak{g}, \mathfrak{h})$ contains some fraction from $\mathfrak{h}$. Alternatively, for proper handlers this says that the dimension has the form (k, ℓ) for some $\ell > 0$.

The definition coincides with the standard one for handlers of dimension $(1, 1)$, which is of course the continuous case.

This product is somewhat clumsy. More intelligent versions could be defined, for example when the glued strings are affixes. However, we give an example to show that the previous definition cannot be simplified. Consider a prefix $\vec{x}$ and a suffix $\vec{y}$. When we combine them with $\vec{y}$ being the argument of $\vec{x}$, we expect the combination to be neither a prefix nor a suffix. It becomes a circumfix $\vec{x} \otimes \vec{y}$, which, when feeded a string $\vec{u}$, will give $\vec{x}\vec{u}\vec{y}$. Thus, the left dimension is 2 rather than 1. The same would be the case if $\vec{y}$ is the function, showing that the affix order cannot be recovered in this case.

The present definitions are in certain cases somewhat suboptimal. Consider the case of a fractured string $\vec{x} \otimes \vec{y}$, which is associated with the handler

$$\langle (0, \mathit{true}), (1, \mathit{true}) \rangle. \tag{2.45}$$

The dimension is $(2, 0)$, the second argument is empty. We have in total a one-argument function. So, we have one section, which consists of the concatenation of $\vec{x}$ and $\vec{y}$. This suggests that the fractured string is effectively behaving like a string. Thus, we can simply the fractured string to $\vec{x}^\frown\vec{y}$, with handler $\langle (0, \mathit{true}) \rangle$.

We have restricted the application of handlers to fractured strings of the exact dimension. There are times when this is too restrictive.

Definition 2.16 (Generalized Handlers) *A **generalized handler** is a set G of handlers such that for any two fractured strings $\mathfrak{g}$ and $\mathfrak{h}$ there is at most one $\mathbb{H}$ such that $\mathbb{H}(\mathfrak{g}, \mathfrak{h})$ is defined. We then write $G(\mathfrak{g}, \mathfrak{h})$ for the result of applying $\mathbb{H}$ to $\mathfrak{g}$ and $\mathfrak{h}$, if that is defined.*

This may be somewhat disingineous in allowing too much freedom. Indeed, let us note the following. As long as a generalised handler is finite, any morph containing it can be replaced by a finite list of morphemes, one for each member. So the generalization serves convenience and is of no theoretical significance.

However, with infinite G we may enter new territory. We shall however only use a handful of generalized handlers. The first two are the concatenator and the product. We describe their action first.

$$\Gamma(\bigotimes_{i=1}^{m} \gamma_i, \mathfrak{z}) := \gamma_1^\frown \gamma_2^\frown \cdots ^\frown \gamma_m \tag{2.46}$$

The concatenator Γ simply concatenates the glued strings. The concatenator is very useful when we want to finalise a sentence. Not knowing whether or not it

consists of several parts, we simply concenate everything and get a (glued) string. The second generalized handler is

$$(2.47) \qquad \Theta(\mathfrak{g}, \mathfrak{h}) := \mathfrak{g} \otimes \mathfrak{h}$$

This is the generalization of pairing. Finally, define "left glue"

$$(2.48) \qquad \Lambda(\vec{x}, \bigotimes \vec{y}_i) := \vec{x}^\frown \vec{y}_1 \otimes \vec{y}_2 \otimes \cdots \otimes \vec{y}_m$$

and "right glue":

$$(2.49) \qquad R(\vec{x}, \bigotimes \vec{y}_i) := \vec{y}_1 \otimes \vec{y}_2 \otimes \cdots \otimes \vec{y}_m{}^\frown \vec{x}$$

Exercise 8. Determine $\mathbb{F} \circ \mathbb{B}$ and $\mathbb{B} \circ \mathbb{F}$.

Exercise 9. In German verbs, even though the prefix is inseparable, strong roots inflect the same way when they are prefixed. The PPP of /`geh`/ 'walk' is /`gang`/ 'walked', whence the form /`übergangen`/, from the verb /`übergeh`/. Write an analysis in terms of fractured strings that allows to capture this regularity. (You might need to make use of morphological classes to implement this.)

Exercise 10. Supply the individual handlers for the four generalized handlers. In other words, define for eg the concatenator, what the handlers are that it contains.

2.5 Reduplication

The handlers of the previous section had one thing in common: they were linear in the sense of the following definition.

Definition 2.17 *A handler is **linear** if it is both proper and each part has only one occurrence.*

This means that if $\mathfrak{f} = \mathbb{H}(\mathfrak{g}, \mathfrak{h})$ then the material content (the occurrences of letters in $\mathfrak{f}$) derive from $\mathfrak{g}$ and $\mathfrak{h}$ in a unique fashion: each occurrence of a letter in $\mathfrak{f}$ can be traced uniquely to an occurrence in either $\mathfrak{g}$ or $\mathfrak{h}$, each occurrence of a letter in $\mathfrak{g}$ thus corresponds to a unique occurrence of a letter in $\mathfrak{f}$, and likewise for $\mathfrak{f}$. Consider by way of example the operation of wrapping. Recall that $\mathbb{W}_0 = \langle\langle(0, \mathit{true}, (0, \mathit{false}\rangle, \langle(1, \mathit{true}\rangle\rangle$. This handler is clearly linear. The functor must have two sections, represented here by $(0, \mathit{true})$, $(1, \mathit{true})$, and the argument has a single section, represented here by $(0, \mathit{false})$. Now,

$$(2.50) \qquad \mathbb{W}_0(\texttt{zog␣} \otimes \texttt{hervor.}, \texttt{den␣Apfel␣}) = \texttt{zog␣den␣Apfel␣} \otimes \texttt{hervor.}$$

Notice that this example operates on syntactic constituents (which can be seen from the fact that the strings contain blanks). The strings contain in total 4+7 = 11 letter tokens for $\mathfrak{g}$ and 10 in the case of $\mathfrak{h}$. The composed result contains 14+7 = 21 letter tokens, exactly the sum.

As we have discussed in Section 2.1, there are exceptions to this. Reduplication results in the duplication of certain substrings and so the result of applying a duplication operation creates new letter occurrences. Consider for example the plural morph of Malay, /-/, containing the handler

$$(2.51) \qquad \mathbb{D} := \langle\langle(0, \mathit{false}), (0, \mathit{true})\rangle, \langle(0, \mathit{false})\rangle\rangle$$

Apply this to /orang/ ‘man’.

$$(2.52) \qquad \mathbb{D}(\texttt{-}, \texttt{orang}) = \texttt{orang}\hat{}\texttt{-}\hat{}\texttt{orang}$$

Here, the token count is 1 (for the functor) and 5 for the argument. The total for the result is $2 \cdot 5 + 1 = 11$, however, since the argument string is produced twice.

Similarly, consider the instrumental in Hungarian, as discussed in Section 2.2. If a noun ends in a consonant, that consonant is doubled: /ember/ ‘man’ → /emberrel/. (Exceptions are nouns ending in a double consonant, like /vicc/ ‘joke’ which becomes /viccel/, and digraphs like /csúcs/ ‘summit’’, which becomes /csúccsal/, which we can attribute to an orthographic rule for writing the doubled digraph /cscs/.) To handle this, we shall assume that nouns have two parts, the last part consisting either of an empty string (when the noun ends in a vowel) or in a single letter (when the noun ends in a consonant). For example, we shall have /embe $\otimes$ r/ ‘man’ and /hajó $\otimes$ ε/ ‘ship’. The instrumental has the forms /al/ and /el/ (for consonantal nouns, depending on vowel harmony) and

/val/, /vel/ for nonconsonantal nouns, again depending on vowel harmony. It is combined with the following handler.

(2.53) $\mathbb{I} := \langle\langle(0, \mathit{false}), (1, \mathit{false}), (1, \mathit{false}), (0, \mathit{true}\rangle\rangle$

We have

(2.54) $\mathbb{I}(\mathtt{embe} \otimes \mathtt{r}, \mathtt{el}) = \mathtt{embe}^\frown\mathtt{r}^\frown\mathtt{r}^\frown\mathtt{el} = \mathtt{emberrel}$

Likewise, we have

(2.55) $\mathbb{I}(\mathtt{haj\acute{o}} \otimes \varepsilon, \mathtt{val}) = \mathtt{haj\acute{o}}^\frown\varepsilon^\frown\varepsilon^\frown\mathtt{val} = \mathtt{emberrel}$

There is another perspective on reduplication, which is worth explaining here since it is the one being used in the implementation—though for technical reasons. Instead of viewing the Malay plural as an operation creating two identical strings out of one, we may view it as an operation concatenating two strings *on condition that they are identical*. So, the plural morph attaches to two tokens of /orang/ 'man' to become /orang-orang/. This makes plural a binary operation. The execution of this idea must be deferred until we have dealt with sequences of arguments. It should be stressed that the semantics is nevertheless unary: the semantics of one of the input nouns is discarded. It is derived only "pro forma".

From a processing point of view this is less than optimal, to be sure. On the other hand it allows to generalize to cases where we do not have exact reduplication.

2.6 The Morph

So far we have looked at morphs that need a single other morph to form a complete unit. However, quite often morphs literally require more than one element. Basically, this is the case when it is syntactically required that a syntactic argument be given. Thus a verb that requires both subject and object to be present is a morph that needs not just one but two other morphs. (Remember that we do not distinguish lexical from sublexical morphs.) This deficit needs to be removed.

Before we begin, however, let us add one more detail. We have allowed morphs to be empty, where a morph is empty if its exponent does not contain

occurrences of letters. This includes the empty fractured string as well as fractured strings containing the empty string. However, consider two empty morphs, one being able to change a verb into a corresponding noun, and one that allows a noun to be made into a verb. With these two morphs it is possible to keep going indefinitely. This is not only a technical nuisance (parsers would run indefinitely). It is also not attested in languages. Pesetzky 1995 quotes Myers with the following law:

> *Myer's Generalization*
> Zero-derived words do not permit the affixation of further derivational morphemes.

To implement this, we associate with every morph a pair of natural numbers (i, j) such that $i > j$. These are the so-called *ranks*. The ranks can best be pictured as slots in a template morphology. An empty morph spans one or more of such slots, and so when it is added to some element it will inevitably reduce the rank. Of course, one could require adding such ranks everywhere, and so we could charge the rank function with the work that we decided to put on the morphological classes. However, when we deal with empty morphs, it becomes necessary to make use of such ranks again. The number i is the minimum rank of arguments that this morph takes, and j is the rank that it yields.

Definition 2.18 (Rank) *A **rank** is a pair of natural numbers. The first member of the pair is called the **in-rank** the second the **out-rank**. Two elements with rank (i, j) and (i', j') with the first being the function combine only if $j' = i$.*

We can put this in the form of an equation.

$$(2.56) \qquad (i, j) \cdot (i', j') := \begin{cases} (i', j) & \text{if } i = j' \\ \text{undefined} & \text{else} \end{cases}$$

The lexicon allows also elements to specify the in- or out-rank as "any". This element matches any other. Hence, $(\text{any}, j) \cdot (i', j')$ is always defined and yields (i', j). (The minimal rank is of course 0.)

Definition 2.19 (Proper Rank) *The rank of a glued fractured string is **proper** if either that string is nonempty or else the rank is (i, j) with $i > j$.*

Here is a convention.

> [Rank for Nonempty Elements]
> By default, non-empty morphs get the rank $(any, 0)$.

We are ready to complete the definition of what a morph is. First we shall group the morphological classes with the handlers. They together provide the selectional information and the levels of morphs.

Definition 2.20 (Selector) *A **selector** is a triple $\sigma = (M, N, \mathbb{H})$, where M and N are morphological classes and $\mathbb{H}$ a (generalized) handler. M is called the **in-class** of σ, N its **out-class**.*

The selector tells us what happens when the morph is applied to another morph. Suppose we have a functor $\sigma = (M, N, \mathbb{H})$ and an argument $\sigma' = (M', N', \mathbb{H}')$. The result of applying σ to σ' is defined (in first approximation) as follows.

(2.57) $$\sigma \cdot \sigma' := (M', N, \mathbb{H} \circ \mathbb{H}')$$

However, this is only defined if $N' = M$. But there is more. In actual fact, the way one should think about the selectors is as specifications of change. Hence they tell us how the particular input categories are changed by the functor. Say that a **fully specified morphological class** is an AVS where each attribute of the grammar receives a value (or $\star$). Then we are really thinking of a selector as a pair consisting of a handler $\mathbb{H}$ plus a function f from fully specified morphological classes to fully specified morphological classes. Then, with each selector specifying its own function, we would simply compose them as we did with the handlers. However, this is not only impractical but too general for the vast majority of purposes. However, in principle the underlying idea is kept. The excution of this idea reduced to just a few cases. We specify the function f separately for each attribute. And for each attribute ATT there are basically four cases.

- ATT is given a value a in M and a value b in N. Then f is defined on all nonempty values $a' \subseteq a$ and returns b.
- ATT is given no value in M but some value in N. Then f is defined on all nonempty values a and returns b.

- ATT is given a value a in M but no value in N. In that that case f is defined on all nonempty $a' \subseteq a$ and returns a'.
- ATT is neither given a value in M nor in N. Then f is defined on all a and returns a.

Notice that this is not a fact but a notational convention on the handling of AVSs. The pairs (M, N) are pairs of AVSs but they designate a function. The casewise definition specifies the product $(M, N) \cdot (M', N')$ by computing the values for each occurring attribute. (Nonoccurring attributes can be safely discarded.)

This product is quite tricky to define properly. We specialize on a given attribute, ATT. Rather than writing no value somewhere, we introduce a special value "idem" (written ✓) that represents the identity function. So we have to calculate the following product.

$$([\textsc{att} : a_1], [\textsc{att} : a_2]) \cdot ([\textsc{att} : a_3], [\textsc{att} : a_4]) \tag{2.58}$$

We have to distinguish various cases for the a_i. They could be proper values (some set of values), they could be $\star$, or even $\checkmark$. However, this latter choice is excluded for a_1 and a_3, for obvious reasons. The operation begins with attempting to match a_4 with a_1. The result is the intersection for standard values, if nonempty. Otherwise it is undefined. (The remaining case where the match is defined is where both of them is $\star$.) Call the new value b. If the values do not match, the operation fails. Finally, a_4 could also be $\checkmark$. In this case we need to match a_3 with a_1 instead.

We have spoken above about the need to allow for multiple arguments. To make room for these arguments, we associate a different selector for each argument. Thus we finally have the following definition.

Definition 2.21 (Morph) *A **morph** is a triple* $\mathfrak{m} = (\mathfrak{g}, \mathcal{A}, \rho)$*, where*

1. $\mathfrak{g}$ *is a fractured glued string;*
2. $\mathcal{A}$ *is a vector of selectors; and*
3. ρ *a proper rank.*

We call the length of $\mathcal{A}$ *the **dimension** of* $\mathfrak{m}$.

The mechanics of the vectors will be discussed later (see Page 91; the critical concept is that of a *diacritic*). Suffice it to say here that each selector that essentially takes an argument (i.e. each selector whose handler is of dimension (i, j) for some $j > 0$) must be some morph. In the simplest of all cases, the sequence of selectors $(\sigma_0, \sigma_1, \cdots, \sigma_n)$ projects n arguments, that must be fed from right to left (that is, starting with σ_n). When the argument has been fed, the corresponding selector is eliminated. Thus at the end we get a selector sequence (σ_0), where σ_0 does not need an argument. (We shall see in the next chapter that this is too simple. In short, some directives on argument handling are still needed.)

Notice that the rank is associated with the entire morph, not the particular arguments. This makes sense because of the following restriction.

> [No Empty Arguments]
> There is a prohibition against taking empty arguments.

One may think of particular counterexamples (empty pronominal elements are a case in point), but they are rather few in number and arcane. From the standpoint of implementing this grammar one is advised not to proliferate empty elements. Thus, when the morph is applied for the first time to one of its arguments, the resulting combination is non-zero. From that point on, the rank has lost its significance. Thus, it suffices to specify a single rank for a morph, as its use is restricted to the combination with its first argument.

We denote by $m \star m'$ the merge of the morphs m and m'. This is a partial operation. Now let $m = (\mathfrak{g}, \mathcal{A}, \rho)$ and $m' = (\mathfrak{h}, \mathcal{B}, \sigma)$. When is their merge defined and what is the result? Again, we shall defer a complete answer and treat only the simplest case, when $\mathcal{B}$ is a singleton (υ_0), requiring no more arguments. In that case, when $\mathcal{A} = (\tau_0, \cdots, \tau_n)$ we use the handler to determine the effective glued string $\mathbb{H}(\mathfrak{g}, \mathfrak{h})$, and multiply the ranks to determine the rank of the product. After that, τ_n and υ_0 are discarded. At this point the reader will ask why we have implemented the rather intricate system of in-classes and out-classes, when they get thrown away anyway. The answer is twofold. First, the classes are needed to restrict the combination of morphs. Second, there are important subcases (sequences of length 1) where the interdependency of in- and out-class actually does play an important role.

Definition 2.22 (Morpheme) *A **morpheme** is a set of morphs. If* $\mathfrak{m}$ *and* $\mathfrak{n}$ *are*

morphemes, the merge $\mathfrak{m} \star \mathfrak{n}$ *is the set of all* $m \star m'$ *such that* $m \in \mathfrak{m}$, $m' \in \mathfrak{n}$, *where* $m \bullet m'$ *is defined.*

Exercise 11. Suppose that a lexicon is given with empty morphs that are arguments. Can you think of a way to make them functors instead? Or more precisely, can you suggest a way to eliminate empty arguments?

2.7 Implementation Issues

In order to implement all this, various issues needed to be treated that do not come to mind at first sight. The biggest of them is that this implementation needed to be able to use the theory both in parsing text as well as producing it. These tasks are actually quite different. In parsing, you are given a string, that is, a sequence of letters $x_0 x_2 \cdots x_{n-1}$, and you want to provide all analysis terms t_0, t_1, and so on, that yield this sequence. In production, you are given an analysis term t and you wish to find all the strings that this term unfolds to. Here, analysis term is synonymous with structural analysis.

This is akin to the standard idea in Montague Grammar. There are lexical items and there are rules of formation. The items in the lexicon may be of various kinds, but the most interesting one is the **entry**. Entries combine a morpheme, that is, a set of morphs, with a semantics. The glue between them is the argument structure, to be discussed in the next chapter. Additionally, entries (and other kinds of items) have a so-called **identifier**. This is a character sequence by which it can be recalled from the lexicon. This is useful since the overt string may be either insufficient to uniquely identify an entry (in the case of homonyms, for example) and in addition may require special characters that are not easily available on a keyboard. The identifier is also called **major identifier**. For the entry consists of several morphs, each of which carries its own identifier, called **minor identifier**. A morph is addressed as a pair "$m : n$", where m is the major identifier, n the minor identifier. The user may give entries and morphs identifiers, but they are both optional. The system will always choose one if none is given. Identifiers must be unique. If they are not, then an error message is produced. An entry is displayed by showing the identifier and then the sequence of morphs. Below the entry, we find a list of so-called parse terms, which in the case of a lexical entry

are the pairs $m : n$.

Two entries e and e' may be combined using different functions or **modes**. In the present system the mode is mostly uniquely determined by e and e', but that need in general not be the case. Standard merge, the operation used here exclusively, is denoted by the symbol $\boxslash$. For example, when we use the standard merge it means that *each of the morphs of* e *is being applied to each morph of* e′ (via the operation discussed in the previous section). The resulting element consists of all products that are defined. For example, we put various plural morphs into the plural morpheme. Each of them is associated with, say, a particular morphological class. Then the product is defined only for those morphs that match the class of the noun to which the morpheme is attached. Likewise, the noun may have different stems depending on which suffix is being attached to, and so with another class attribute we can make sure that the correct form of the noun is chosen. Thus, the apparent multitude of results is actually often just a singleton in real language applications.

We may give the resulting entry a parse term. If the identifier of e is m and the identifier o e' is m', we choose $\boxslash(m, m')$.

> [Parse Terms]
> In a parse term, the function is always the first argument of a mode,
> the argument is second.

The system however displays the results differently. The entry consists once again of a sequence of morphs, each morph is associated with a term. The morph $m : n$ merged with $m' : n'$ is associated with the term $\boxslash(m : n, m' : n')$.

As explained above, there are two ways to use the lexicon. One is to build an entry, the other is to parse a string. The first performs the operations as defined in the term and executes them. We call this operation (and its result) the **unfolding** of the term. The unfolding operates on fractured glued strings, which are combined according to the specification by the handlers. Parsing proceeds from a given string. The procedure implemented here is a particular kind of chart parser, adapted to handle discontinuity. Technically, the parse table is a function from pairs (i, ρ) consisting of a length and a rank to sets of pairs (a, t), where a is a so-called short argument structure and t a parse term.

Given an initial string of length n, i may be aything between 0 and n. There

are finitely many ranks. The ranks are compiled from the dictionary. The parser only considers ranks that are listed somewhere in the dictionary. All others can be safely ignored. The parser starts by checking for each glued string where it has an occurrence. Occurrences are pairs of numbers (i, j) such that $i \leq j$. (Notice that even empty glued strings can have nontrivial conditions on the context, so they do not have an occurrence everywhere.) This means checking the left and right context conditions. When the occurrences are known, we associate with (i, ρ) all morphs of length i of rank ρ. This requires to check for each section of the morph whether it has an occurrence and to see whether these occurrences are pairwise disjoint. In the next step we try to add empty operators to all these elements. This requires going downwards through the ranks, however only once. This initialises the parse.

Then, with i going from 2 to n, we attempt to fill the entries (i, ρ) by combining entries of length j and $i - j$ $(0 < j < i)$ and then try to add empty operators. Combining occurrences is done on the pairs of numbers, it is not necessary to look at the strings again. An exception occurs with reduplication, however. In a reduplication, the handler looks like an ordinary handler but it is associated with a condition: that two elements are actually identical. In this case, combination must be accompanied by a check whether the identity conditions are satisfied.

Notice that the parser checks only occurrences of morphs. It does not look for morphemes. In actual fact, the morphemes are recorded in the parse term. When the parse is complete, we can recall the analysis together with the morphemes from the parse terms.

Chapter 3

Argument Structure

In this chapter we shall introduce Discourse Representation Structures (DRSs) and so-called Referent Systems, originally due to Kees Vermeulen. The two will be merged into a new semantics for natural language, which is based on variable sharing by overt agreement. However, various changes will be made to the referent systems to accommodate for several special features of language. After they are introduced, we shall derive some basic properties of this semantics. We shall show how to derive $\overline{X}$-syntax and alternate constituent orders.

3.1 Overview

In the previous chapter we have reviewed the morphological structures. In this chapter and the next we shall talk about the semantic structures. In actual fact, there are two different components we shall talk about: the first contains the semantic representation, and the second is an interface that controls the behaviour of the semantic structure under merge. This will also provide some missing details for the morphology.

The semantic structures will be plain Discourse Representation Structures. However, unlike standard semantic theory we will follow Albert Visser and Kees Vermeulen and provide an explicit device to handle the variables under constituent

formation. This device is not part of semantics proper; it is part of the language inasmuch as it determines in which way the different representations will be merged. Whether or not we shall in the end like to subsume under the label "semantics" also that part of the structure which determines the meaning of a constituent [X Y] on top of the truth conditions of X and Y will have to be seen. It is not our intention to discuss that question here. Rather, we shall provide a mechanism that will derive the truth conditional meanings of complex expressions on the basis of their representations.

To see the essential problem, consider a constituent X with associated semantics Δ, and another constituent Y with associated semantics Θ. What should be the meaning of the constituent [X Y]?

There are several answers to this. Montague, following Frege, suggested that the meaning is derived via function application. Thus, it would either be $\Theta(\Delta)$ or $\Delta(\Theta)$. However, he never managed to implement this idea systematically. As is well known, the rule of Quantifying-In uses means that go beyond mere function application. His approach has been thoroughly scrutinized over the last fourty years. One of the biggest problems turned out to be the assumption that the meanings of sentences are propositions. This means that identification of objects across sentences is difficult, to say the least. Discourse Representation Theory (DRT) provided an answer to these problems. In a discourse representation structure (DRS for short), we are allowed to have free variables. These variables can be bound retroactively. Moreover, their scope is extensible so that what looks like an existential quantifier is actually a more dynamic entity.

However, DRT also faced problems. Its systematic use of free variables exposed a problem that was latent in Montague Grammar as well. Namely, it was the problem of choosing a proper name for the variable. For if we want to use free variables, we obviously have to choose some names for them. The names are absolutely crucial, for under an influential proposal put forward by Henk Zeevat, DRSs that contain the same variables should be seen as pointing to the same object. Thus, the proposal was simply that [X Y] will be paired with the union $\Delta \cup \Theta$ of conditions. However, such a proposal is of limited scope. For it requires that every time we draw up some meaning we have to choose our variables very carefully so as to not get them in the way of others.

A solution has been proposed in Vermeulen 1995. The technical device is called a referent system (RS). Refererent systems are explicit devices to manage

the choice of variable names. In a referent system variables come equipped with names, one for input (= left context) and one for output (= right context). If a referent r of X has an output name that equals an input name of some referent r' of Y, then r and r' will be identified under merge. If r finds no match, it will not identified with any referent of Y.

The essential novelty of the present approach is that we treat the names as morphosyntactic properties expressed in terms of AVSs. That one would like to use morphosyntactic properties was already apparent to Visser and Vermeulen at that time. What was less clear was which form this incorporation of morphosyntax will have to take. We shall also do away with the distinction between left and right context and instead propose that the identification of referents is between functor and argument. The relative position may or may not play a decisive role in this. Seen this way, referent systems become more like *argument structures*. Recall that argument structures are the interfaces between syntax and semantics in generative grammar. They have been introduced to

1. provide for lexical properties to be entered into the syntax, and

2. to connect lexical properties with semantic behaviour.

An argument structure states what kinds of syntactic argument a head has. However, it has been argued (Haider 1993) that a purely syntactic argument structure is too impoverished to allow syntax and semantics to be properly linked.

In the present proposal, an argument structure is a sequence of variable identification statements of the form

(3.1) $\langle x : \delta : \Sigma \rangle$

where x is a variable, δ a so-called *diacritic* indicating the functor-argument properties of the variable and Σ some morphosyntactic property (basically, a pair of AVSs). For example, an adjective will have a pair of identical AVSs A associated with it which specify what agreement features it expects of the noun it modifies. The noun must provide these features, and then the two variables will be identified under merge. (In the next chapter we will extend these structures also with parameters.)

The diacritics play a crucial role. First, it is necessary that merge identifies *some* variable. This prevents an adjective to form a constituent with a non agree-

ing noun. Moreover, every variable identification statement is paired with a corresponding selector (see Section 2.6). This provides the bracket between morphology and argument structure. The selector determines the morphological shape of the formed constituent, while the identification statement determines the semantics. Second, we need to specify what happens to the triple $\langle x : \delta : \Sigma \rangle$ after merge has been performed. There are several possibilities, and they are determined by the diacritic. The basic distinction is whether or not the variable will be kept in the sense that it will still be given a morphological name and thus be available for further manipulation. In the case of adjectives it will be kept; in the case of nominal argments to a verb it will not. We have already seen in the chapter on morphology that argument sequences need a specification concerning the survival of the argument, a specification which they do not naturally provide themselves. It is the argument structure which does so. The diacritic is that specification. This diacritic is used also by the corresponding selector. This makes sure that variable identification statements are statements under morphosyntactic merge. They tell us what vector of selectors will be formed under constituent formation.

Here is now the structure of an entry in more detail. The argument structure is a sequence

(3.2) $$\langle x_1 : \delta_1 : \Sigma_1 \rangle, \langle x_2 : \delta_2 : \Sigma_2 \rangle, \cdots, \langle x_n : \delta_n : \Sigma_n \rangle$$

The semantics is a DRSs, in which the variables x_1 through x_n may occur free (but need not occur at all as is the case for example in subjectless sentences). And finally the morpheme is a set of morphs with the selectors

(3.3) $$\sigma_1, \sigma_2, \cdots, \sigma_n$$

The variable identification statement number i is associated with the selector σ_i. Every rearrangement of the above sequence is accompanied by a similar rearrangement of the selector sequence. The mechanics of this will be discussed in Section 3.5. The sections thereafter will discuss ramifications. In the next chapter we shall introduce an important addition, namely parameters. This concludes the exposition of the structures. After that, we shall have everything we need to do a full analysis.

3.2 Basic Semantic Concepts: DRT

As is well known, DRT was created in order to deal with certain problems of the interpretation of pronouns. Interestingly, DRT did away with functions and started to "display" variables instead. This is not as radical a departure from Montague Grammar as it sounds, since the latter always had been using free variables (recall the rule of Quantifying-In). We follow original DRT in dispensing completely with λ-calculus. Our representations will be DRSs enriched by a so-called argument structure. Let us therefore briefly rehearse the basic concepts of DRT.

The meaning of the word /man/, for example, will no longer be $\lambda x.\mathsf{man}'(x)$ (or, equivalently, man') but rather $\mathsf{man}'(x)$, where x is a variable. This is usually denoted in the form of a split box, also called a **Discourse Representation Structure** (DRS). (See Kamp and Reyle 1993 for an introduction to Discourse Representation Theory (DRT).)

(3.4) $\begin{array}{|l|}\hline \varnothing \\ \hline \mathsf{man}'(x) \\ \hline \end{array}$

We call the upper part the **head** and the lower part the **body** of a DRS. The body contains the restriction on the variables of the DRS. The head contains the variables that are existentially quantified over. In particular, the phrase /a man/ will be represented by

(3.5) $\begin{array}{|l|}\hline x \\ \hline \mathsf{man}'(x) \\ \hline \end{array}$

The presence of the variable x in the head-section makes all the difference: it effectively quantifies existentially over the variable.

Definition 3.1 *A **DRS** is a pair $[V : \Delta]$, where V is a finite set of variables and Δ a finite set of formulae or DRSs. The set of DRSs is constructed as follows.*

1. *If x is a variable then $[\{x\} : \varnothing]$, also written $[x : \varnothing]$, is a DRS.*
2. *If ϕ is a formula then $[\varnothing : \{\phi\}]$, also written $[\varnothing : \phi]$, is a DRS.*
3. *If $[V_1 : \Delta_1]$ and $[V_2 : \Delta_2]$ are DRSs then so are*

   (a) $[V_1 \cup V_2 : \Delta_2 \cup \Delta_2]$

(b) $\neg[V_1 : \Delta_1]$

(c) $[V_1 : \Delta_1] \Rightarrow [V_2 : \Delta_2]$

(d) $[V_1 : \Delta_1] \vee [V_2 : \Delta_2]$

We write $f \sim_V g$ if for all $y \notin V$ we have $f(y) = g(y)$. There are more constructors to form DRSs, but the ones above shall suffice for now.

Definition 3.2 *Let $\mathfrak{M} = \langle D, I\rangle$ be a first-order model, $f : Var \to D$ an assignment and δ a DRS. δ is **true in $\mathfrak{M}$ under the assignment** f, in symbols $\langle \mathfrak{M}, f\rangle \models \delta$, if the following holds.*

1. *$\delta = [V : \Delta]$ and there exists a $g \sim_V f$ such that $\langle \mathfrak{M}, g\rangle \models \gamma$ for all $\gamma \in \Delta$.*
2. *$\delta = \neg[V : \Delta]$ and for no $g \sim_V f$ we have $\langle \mathfrak{M}, g\rangle \models \gamma$ for all $\gamma \in \Delta$.*
3. *$\delta = [V_1 : \Delta_1] \vee [V_2 : \Delta_2]$ and either there exists a $g \sim_{V_1} f$ such that $\langle \mathfrak{M}, g\rangle \models \gamma$ for all $\gamma \in \Delta_1$ or there exists a $g \sim_{V_2} f$ such that $\langle \mathfrak{M}, g\rangle \models \gamma$ for all $\gamma \in \Delta_2$.*
4. *$\delta = [V_1 : \Delta_2] \Rightarrow [V_2 : \Delta_2]$ and for all $g \sim_{V_1} f$ such that $\langle \mathfrak{M}, g\rangle \models \gamma$ for all $\gamma \in \Delta_1$ there exists a $h \sim_{V_2} g$ such that $\langle \mathfrak{M}, h\rangle \models \gamma'$ for all $\gamma' \in \Delta_2$.*

We define now the notion of accessibility and boundedness. Let $\delta = [V : \Delta]$; then δ is **immediately accessible** to every $\gamma \in \Delta$. Furthermore, in $\delta' \Rightarrow \delta''$, δ' is immediately accessible to δ'', but δ'' is not immediately accessible to δ'. In $\delta' \vee \delta''$, neither is δ' immediately accessible to δ'' nor is δ'' immediately accessible to δ'. Accessibility is the reflexive and transitive closure of immediate accessibility: δ is **accessible** to δ' if $\delta = \delta'$ or there exist γ_i, $1 \leq i \leq n$, such that $\gamma_1 = \delta'$, $\gamma_n = \delta$, and for all $i < n$ the DRS γ_i is immediately accessible to γ_{i+1}. An occurrence of a variable in the body of δ is **bound** if there exists a DRS γ accessible to δ whose head contains x. An unbound occurrence is called **free**.

The constructors $\neg$, $\Rightarrow$, and $\vee$ correspond to negation, implication and disjunction. The operation in (3a) corresponds to the standard merge of the DRS. We will call it the **union**, since we will define a different merge on DRSs.

Definition 3.3 *Let $\delta_1 = [V_1 : \Delta_1]$ and $\delta_2 = [V_2 : \Delta_2]$ be two DRSs. The **union** of δ_1 and δ_2 is denoted by $\delta_1 \cup \delta_2$ and defined by*

$$\delta_1 \cup \delta_2 := [V_1 \cup V_2 : \Delta_1 \cup \Delta_2]$$

Let us show briefly how in DRS we can calculate the meaning of a simple phrase. Let us take the sentence

(3.6) `A tall man sees a small rose.`

The intended translation is the following DRS (modulo renaming of variables).

(3.7)

x y	
man′(x)	tall′(x)
rose′(y)	small′(y)
see′(x, y)	

For the DRS is true in a model iff there is an a and a b such that a is a tall man, b a small rose and a sees b. We assume that nouns and adjectives are given the same interpretation. For example, /`man`/ is translated by

(3.8)

$\emptyset$
man′(x)

The indefinite article is translated by

(3.9)

x
$\emptyset$

And, finally, the verb is translated by

(3.10)

$\emptyset$
see′(x, y)

We first choose a constituent analysis.

(3.11) ((`A` (`tall man`))(`sees` (`a` (`small rose`))))

When two parts of speech form a constituent, we form the union of the respective DRSs to get the associated semantics. Obviously, this will only result in a correct translation if we decide on the proper variables to be inserted into the DRS. For notice that the expression /`man`/ can also be translated by

(3.12)

$\emptyset$
man′(y)

Therefore, what we need is the following structure prior to translation into DRS-language.

(3.13) $((\mathtt{A}_x\,(\mathtt{tall}_x\,\mathtt{man}_x))(\mathtt{sees}_{x,y}\,(\mathtt{a}_y\,(\mathtt{small}_y\,\mathtt{rose}_y))))$

The indices shall guide the translation in the following way. If there is a single variable z in the DRS and the corresponding expression has index x, then the variable z in the DRS shall be replaced by x. If there are two variables in the DRS, z and z' and the corresponding expression has the indices x and y, then z and z' are replaced by x and y. (Notice that in order to be able to tell which variable is replaced by which other variable we would have to assume that the head of a DRS is not a set but a sequence.) The annotated expression $(\mathtt{a}_x\,(\mathtt{tall}_x\,\mathtt{man}_x))$ is therefore translated by

(3.14) $$\begin{array}{|l|}\hline x \\ \hline \varnothing \\ \hline\end{array} \cup \left(\begin{array}{|l|}\hline \varnothing \\ \hline \mathsf{tall}'(x) \\ \hline\end{array} \cup \begin{array}{|l|}\hline \varnothing \\ \hline \mathsf{man}'(x) \\ \hline\end{array}\right) = \begin{array}{|l|}\hline x \\ \hline \mathsf{tall}'(x) \\ \mathsf{man}'(x) \\ \hline\end{array}$$

Similarly, $(\mathtt{a}_y\,(\mathtt{small}_y\,\mathtt{rose}_y))$ is translated as

(3.15) $$\begin{array}{|l|}\hline y \\ \hline \mathsf{small}'(y) \\ \mathsf{rose}'(y) \\ \hline\end{array}$$

The reader is asked to check that we get the desired translation as the result of

(3.16) $$\begin{array}{|l|}\hline x \\ \hline \mathsf{tall}'(x) \\ \mathsf{man}'(x) \\ \hline\end{array} \cup \left(\begin{array}{|l|}\hline \varnothing \\ \hline \mathsf{sees}'(x,y) \\ \hline\end{array} \cup \begin{array}{|l|}\hline y \\ \hline \mathsf{small}'(y) \\ \mathsf{rose}'(y) \\ \hline\end{array}\right)$$

This algorithm has several drawbacks. First, most of the variable management that the λ-calculus was doing in Montague Grammar now has to be done "by hand". This is unsatisfactory. In Montague's original calculus, we would have to choose only a constituent structure and then a correct translation will be returned (however not for Quantifiying-In!). However, as we have seen earlier, even this is too much to be assumed. So, we would ideally like to assume no constituent structure at all. We want a calculus that just takes a string and returns a translation. For that, some of the information concerning the structure must be put into the semantics. This is roughly what we will do, though it is not semantics proper but a kind of interface between morphosyntax and semantics. Furthermore, we

need to reflect a little bit on the nature of the operation with which we translated the constituent juncture. We have hitherto assumed that it is the union. However, there are good arguments to show that the union is not a good choice.

We call an operation $\bullet$ a *merge* only if it has the following property.

$$\langle \mathfrak{M}, f\rangle \models \delta \bullet \delta' \qquad \Leftrightarrow \qquad \langle \mathfrak{M}, f\rangle \models \delta \quad \text{and} \quad \langle \mathfrak{M}, f\rangle \models \delta' \tag{3.17}$$

This means that $\delta \bullet \delta'$ is the true conjunction of the two DRSs δ and δ'. For we intend each of the DRSs to supply information about their respective variables. However, it is easy to see that the union fails to have this property. Namely, let α and β be unary predicates and $\mathfrak{M} := \langle \{a, b\}, I\rangle$ with $I(\alpha) := \{a\}$, $I(\beta) := \{b\}$. Let f be any assignment. Then

$$\langle \mathfrak{M}, f\rangle \models [x : \alpha(x)]; [x : \beta(x)]$$

(Note that we do not write the head and body in the usual set notation. Typically, we just write the items separated only by a semicolon; that is to say, we drop the set braces.) However, we do not have

$$\langle \mathfrak{M}, f\rangle \models [x : \alpha(x), \beta(x)]$$

A somewhat simpler example is $\delta := [x : \varnothing]$ and $\delta' := [\varnothing : \alpha(x)]$ and f any function such that $f : x \mapsto b$, where α is not true of b. Then $\langle \mathfrak{M}, f\rangle \models [x : \alpha(x)]; [x : \varnothing]$ but $\langle \mathfrak{M}, f\rangle \not\models [\varnothing : \alpha(x)]$. In this example we have a DRS which has a variable in the body that is unbound.

So, the union is not a good merge. The problem is that we take the *set theoretic* union of the heads rather than the *disjoint* union. Note namely that an occurrence of the variable x in the head of δ means *there is an x such that δ* and that likewise an x in the head of δ' means *there is an x such that δ'*. It surely does not follow that *there is an x such that δ and δ'*, because it might happen that the x satisfying δ is different from the x satisfying δ'. This is why we have to separate the sets of variables of δ and δ'. This we do as follows.

Definition 3.4 (Strong Substitution) *Let $s : V \to V$ be a map from variables to variables. Then the strong substitution resulting from s, also denoted by s, is defined as follows.*

1. $s(t(u_1, \cdots, u_n)) := t(s(u_1), \cdots, s(u_n))$*, t an n-ary function symbol;*

2. $s(R(t_1, \cdots, t_n)) := R(s(t_1), \cdots, s(t_n))$*, R an n-ary predicate symbol;*

3. $s([\varnothing : \varphi]) := [\varnothing : s(\varphi)]$*;*

4. $s(\delta_1 \cup \delta_2) := s(\delta_1) \cup s(\delta_2)$*;*

5. $s([x : \varnothing]) = [s(x) : \varnothing]$*;*

6. $s(\neg\delta_1) := \neg s(\delta_1)$*;*

7. $s(\delta_1 \Rightarrow \delta_2) := s(\delta_1) \Rightarrow s(\delta_2)$*;*

8. $s(\delta_1 \vee \delta_2) := s(\delta_1) \vee s(\delta_2)$.

Notice that strong substitution is *not* the same as substitution in ordinary predicate logic: it is entirely string based and substitutes free and bound occurrences alike. Of course, it is the job of the referent systems to take care that this is harmless. If $s(x) = y$, and $s(z) = z$ for $z \neq x$, then we write $[y/x]$ for the substitution induced by s.

Variables get superscripts consisting of sequences of 1s and 2s. These superscripts are finite, but can be arbitrarily long. Now, for a set V of variables we write

(3.18) $V^1 := \{x^1 : x \in V\}$

So, if $x = v^\alpha \in V$ then $x^1 := v^{\alpha 1} \in V^1$. The substitution so defined is denoted by ℓ_1. Likewise for the addition of '2'; this defines the substitution ℓ_2. If no confusion arises, we write Δ^1 in place of $\ell_1(\Delta)$ and Δ^2 in place of $\ell_2(\Delta)$.

Definition 3.5 *Let* $\delta = [V : \Gamma]$ *and* $\delta' = [W : \Delta]$ *be two DRSs such that no variable occurs free. Then the* ***merge*** *of* δ *with* δ'*, is defined by*

$$\delta \bullet \delta' := [V^1 \cup W^2 : \Gamma^1 \cup \Delta^2]$$

The reader may check that

(3.19) $\delta \bullet \eta = \delta^1 \cup \eta^2$

We shall show that this operation indeed is a merge. To that end, assume that

(3.20) $\langle \mathfrak{M}, f \rangle \models [V^1 \cup W^2 : \Gamma^1 \cup \Delta^2]$

Put $X := V^1 \cup W^2$. Then there exists a $g \sim_X f$ such that $\langle \mathfrak{M}, g \rangle \models \gamma$ for all $\gamma \in \Gamma^1 \cup \Delta^2$. Put $h_1(x) := g(x^1)$ for all $x \in V$, and $h_1(x) := f(x)$ otherwise. Likewise put $h_2(x) := g(x^2)$ for all $x \in W$ and $h_2(x) := f(x)$ otherwise. Now $h_1 \sim_V f$ and $h_2 \sim_W f$. It is an easy matter to verify that for every $\gamma \in \Gamma$

$$(3.21) \qquad \langle \mathfrak{M}, g \rangle \models \gamma^1 \quad \Leftrightarrow \quad \langle \mathfrak{M}, h_1 \rangle \models \gamma$$

and that for every $\delta \in \Delta$

$$(3.22) \qquad \langle \mathfrak{M}, g \rangle \models \delta^2 \quad \Leftrightarrow \quad \langle \mathfrak{M}, h_2 \rangle \models \delta$$

Hence, $\langle \mathfrak{M}, f \rangle \models [V : \Gamma]$ and $\langle \mathfrak{M}, f \rangle \models [W : \Delta]$. Conversely, let $\langle \mathfrak{M}, f \rangle \models [V : \Gamma]; [W : \Delta]$. Then $\langle \mathfrak{M}, f \rangle \models [V^1 : \Gamma^1]$ as well as $\langle \mathfrak{M}, f \rangle \models [W^2 : \Delta^2]$. (Here we need that every variable is bound.) So there exists an h^1 such that $\langle \mathfrak{M}, h^1 \rangle \models [V^1 : \Gamma^1]$ and an h^2 such that $\langle \mathfrak{M}, h^2 \rangle \models [W^2 : \Delta^2]$. Since V^1 and W^2 are disjoint, the following is well-defined: $g(x) := h^1(x)$ if $x \in V^1$, $g(x) := h^2(x)$ if $x \in W^2$ and $g(x) := f(x)$ else. Then $\langle \mathfrak{M}, g \rangle \models \Gamma^1$ and $\langle \mathfrak{M}, g \rangle \models \Delta^2$ and so $\langle \mathfrak{M}, g \rangle \models \Gamma^1 \cup \Delta^2$. Therefore, $\langle \mathfrak{M}, f \rangle \models [V^1 \cup W^2 : \Gamma^1 \cup \Delta^2]$.

Let us finally return to unbound variables. In a DRS $[\varnothing : \alpha(x)]$ the variable x occurs free. Likewise in $[\varnothing : \beta(x)]$. In this case, we do have

$$(3.23) \qquad \langle \mathfrak{M}, f \rangle \models [\varnothing : \alpha(x), \beta(x)] \quad \Leftrightarrow \quad \begin{array}{rl} & \langle \mathfrak{M}, f \rangle \models [\varnothing : \alpha(x)] \\ \textit{and} & \langle \mathfrak{M}, f \rangle \models [\varnothing : \beta(x)] \end{array}$$

Hence, free occurrences should in fact not be renamed. This will make the definition of the proper merge rather cumbersome, and we have therefore excluded that case. Notice that also that in our translation we cannot define the union simply by the merge as just defined, since we made crucial use of free variables. Rather, the whole machinery has to be changed. First of all, we do not allow any free variables. Therefore, /man/ and /see/ are translated by

(3.24)

x
$\mathsf{man}'(x)$

$x \quad y$
$\mathsf{see}'(x, y)$

By DRT interpretation, these translations mean *there is a man* and *something sees something*. Hence, the indefinite article has lost its function. This is not so tragic. Indeed, many languages do not even have an indefinite article; moreover, it is still not without function for syntactically it is often needed as a left boundary marker for a noun phrase. If we now translate the sentence using the merge, we would

of course get a totally wrong translation, which can be paraphrased as follows: *there is something, there is a tall thing, there is a man, something sees something,* .... We now have the opposite problem: variables are distinct even when they should be equal. As we shall see, this is a much more favourable position to be in. What we will now try to achieve is the following: we will assume that the words in addition to the DRSs also contain some information as for how the variables should be handled when the DRS is merged with another one; in particular, we need information as for which variables should in fact be the same after merge. So, by some means two DRSs that are merged should be able to communicate with each other the idea that certain of their variables are actually talking about the same individual. Exactly this information is hidden in the syntax and should be brought to light. This leads us directly to the next section.

Exercise 12. Show that the following DRSs are equivalent in the sense that they are true in the same models under the same (partial) valuations.

(3.25)

x, y
$\varphi(x, y); x \doteq y$

x
$\varphi(x, x)$

Exercise 13. In model theory it is a standard exercise to show that any language with functions can be replaced by a theory with only relations. For an n-ary function symbol f introduce an $n+1$-ary relation symbol R with the intention that $x_{n+1} = f(x_1, \cdots, x_n)$ is equivalent to $R(x_1, \cdots, x_n, x_{n+1})$. Given a theory T expressed using f alone. Write a theory T' with R in place of f that is equivalent to T in the sense that any model of T becomes a model of T' (and vice versa) replacing the interpretation of f by that of R (and vice versa).

Exercise 14. This exercise shows that we can do away with everything except propositions as meanings of constituents. In Montague Grammar, if X has the meaning f and Y has the meaning g then the meaning of $[X\ Y]$ is either $f(g)$ or $g(f)$, depending on types. Without loss of generality, let it be the first. Now rewrite the meaning of X as $x_2 = f(x_1)$ and the meaning of Y as $x_1 = g$. What are the types of these variables? What should be meaning of $[X\ Y]$? What operation can yield this?

3.3 A New Theory of Semantic Composition

In Visser and Vermeulen 1996 and Vermeulen 1995, Kees Vermeulen and Albert Visser have formulated a new theory of meaning. Its philosophy is that the mechanism for gluing meanings is not function application or union of DRSs but a rather articulated operation. The primary reason for this is that they wanted to create an interpretation mechanism that satisfies several conditions. First, any part carries meaning, and gluing certain parts together is basically the same as heaping up meanings. So, rather than determining the meaning by applying a function to an argument we simply take the conjunction of such meanings. This is reasonable because in many cases it is impossible to say which of the two items is a function and which one is the argument. Adjectives and adverbs are a case in point. Second, interpretation works strictly left to right, is fully associative, and allows for starting at any point in the discourse. The latter property is called the *break-in-principle*. It is motivated by the fact that discourse is linear, and the constituent structure which we use in Montague Grammar to assemble the meaning of a sentence has to be derived from the string. The information concerning the sentence structure is encoded into the linear order and the morphology of the words. The latter is very important for our purposes. We wish to bring to light exactly those parts of speech that are concerned with the composition of meaning.

In addition, as we have observed earlier, alternative formalisms such as DRT and Dynamic Montague Grammar (see Groenendijk and Stokhof 1990) all share the problem that the names of variables must be chosen with care to ensure the correctness of the interpretation at points where it should actually matter least. When inserting the meaning of an item into a structure—say $\mathsf{man}'(x)$—the choice of the variable should be immaterial, because any other variable is just as fine for the mere meaning of that item. (See Kracht 2011 for extensive discussion.) But at the point of insertion there might be an accidental capture of that variable, and this has to be prevented. In Montague's own system this does not arise in this particular form since we do not allow free variables. However, as soon as binding facts are to be accounted for, a notion of identity of bound variables is to be reintroduced, giving rise to the infamous rules of *Quantifying-In*. Now rather than stipulating this, Vermeulen and Visser let the merging operation itself take care of the variable management. Thus, while Montague would let the machinery of λ-calculus do the variable handling, here it is the semantic system itself that does it. Moreover, in some sense this is the only thing it is doing. So, we must be

interested in knowing how it does the job. If two chunks of meaning m_1 and m_2 are merged into $m_1 \bullet m_2$ (think of m_1 and m_2 as being ordinary formulae, or DRSs) then the merge will make all variables of m_2 distinct from those of m_1 before putting them into a single structure. This is the default case; if however m_1 and m_2 contain information to the effect that a variable is intended to be shared between them, then the merge will refrain from renaming that variable in m_2. (This is the problem of identifying "coordinated variables" in the sense of Fine 2007.) Of course, the immediate question is how m_1 and m_2 can make it clear that a variable is to be shared. The solution is quite simple: we introduce a vocabulary by which DRSs can communicate about the status of their variables, whether some of them should be identified and others not. This vocabulary will initially be rather simple but later on it will become more and more involved.

Definition 3.6 *Let N be a set. A **referent system over** N is a triple $\langle I, R, E\rangle$, where R is a finite set, called the **set of referents**, I a partial injective function from N to R, called the **import function** and E a partial injective function from R to N, called the **export function**. N is called the set of **names**. If $I(A) = x$, x is said to have **import name** A; and if $E(x) = A$ then x is said to have **export name** A.*

Definition 3.7 *Let N be a set. An **N-system** over N is a pair $[\mathfrak{R} : \Gamma]$, where $\mathfrak{R} = \langle I, R, E\rangle$ is a referent system over N and Γ a DRS over R. $\mathfrak{R}$ is called the **argument structure** of the N-system and Γ the **body**.*

Actually, it is also possible to define DRS-like structures by allowing Γ to be a set of formulae and N-systems, respectively. We will not make much use of these extended structures. Moreover, we will have to provide means of handling argument structures inside Γ. That case is therefore put aside here. (However, see Section 3.5.) N-systems are written vertically rather than horizontally. Since in a DRS the variable set is separated from the body by a horizontal line, we use a double line to separate the referent system from the DRS. Further, in order to denote the pair $\langle \vec{x}, [\mathfrak{R} : \Gamma]\rangle$, where $\vec{x}$ is a string of our language (or more generally an exponent) with denotation $[\mathfrak{R} : \Gamma]$, we usually put the exponent on top of the N-system. The following example illustrates this.

$\vec{x}$	`/man/`
$\mathfrak{R}$	$\langle \text{NOM}, x, \text{NOM}\rangle$
V	x
Δ	$\mathsf{man}'(x)$

The merge of two N-systems is defined in two stages. First, we show how referent systems are merged; the merge of N-systems is then rather straightforward. For the definition of the merge recall the merge of two DRSs. There we used the superscript notation. Here we will make this somewhat more precise. Notice first of all that Vermeulen 1995 uses the notion of a **referent**, which is distinct from a variable, whence the name **referent systems**. In what is to follow, the terms "referent" and the "variable" are used synomously, however. Referents can be identified with addresses of a memory cell. The particular address is unimportant as long as we can properly manage these addresses. (Think of the choice of variable names in Prolog.) Referents are featureless objects, they can be distinct or equal; nothing more is important. Our referents have the form $v^\frown\sigma$, where v is a basic identifier string (used to denote the type of the variable), and σ is an element of $\{\mathtt{1}, \mathtt{2}\}^*$, that is, a finite sequence of 1s and 2s. We use x, y and z as metavariables. If x is a variable we also write x^1 in place of $x^\frown\mathtt{1}$, and x^2 for $x^\frown\mathtt{2}$. Using these sequences is a good way to track occurrences of a variable. Now for the definition of the merge. We present some examples first. We write $\{A : x : B\}$ or simply $A : x : B$ to say that x is imported under the name A and exported under the name B. (So, $I(A) = x$ and $E(x) = B$.) If x has no import name we write $- : x : B$, if it has no export name we write $\{A : x : -\}$; we write $\{- : x : -\}$ if x has neither an import name nor an export name. A referent system is simply a set of triples $\{\alpha : x : \beta\}$ where $x \in R$ and $\alpha, \beta \in N \cup \{-\}$. (We use small Greek letters to denote elements of $N \cup \{-\}$ while upper case Roman letters continue to denote names, that is, elements of N.) Suppose we merge two referent systems $\{\alpha : x : \beta\}$ and $\{\gamma : y : \delta\}$. Then several cases may arise. First assume $\beta, \gamma \in N$.

1. $\beta \neq \gamma$. Then x and y are made distinct by using a superscript 1 and 2, and the resulting referent system is
   (a) $\{\alpha : x^1 : \beta, \gamma : y^2 : \delta\}$, if $\alpha \neq \gamma$ and $\beta \neq \delta$;
   (b) $\{\alpha : x^1 : \beta, - : y^2 : \delta\}$, if $\alpha = \gamma$ and $\beta \neq \delta$;
   (c) $\{\alpha : x^1 : -, \gamma : y^2 : \delta\}$, if $\alpha \neq \gamma$ and $\beta = \delta$;
   (d) $\{\alpha : x^1 : -, - : y^2 : \delta\}$, if $\alpha = \gamma$ and $\beta = \delta$.
2. $\beta = \gamma$. Then x and y are taken to be the same variable, which is x^1 (to make the definitions uniform). The resulting referent system is $\{\alpha : x^1 : \delta\}$.

The reason for treating all these subcases separately is that if $\alpha = \gamma$, then the new import function is not well-defined unless one of the two referent loses its import

name (or gets a completely different name, a possibility that we have ruled out). Similarly if $\beta = \delta$.

Remains the case where either β or γ are identical to $-$. In that case no identification is possible and we continue as in 1. above.

Thus, when we merge two refererent systems we need to make sure that the new system is also well defined. The problem is due to the fact that two referents can have the same import or export name. If that is so, they must come from different referent systems, of course. If in the merge x has the export name that y imports, we say that *x supervenes y*. If x and y compete for the same import name, *x I-preempts y*, and if they compete for the same export name, *y E-preempts x*. These situations can arise in all combinations.

Definition 3.8 *Let $\rho_1 = \langle I_1, R_1, E_1\rangle$ and $\rho_2 = \langle I_2, R_2, E_2\rangle$ be referent systems over N. Let $x \in R_1$ and $y \in R_2$. We say that x* ***supervenes*** *y if $I_2(E_1(x)) = y$. We say that x* ***I-preempts*** *y if there is a $A \in N$ such that $I_1(A) = x$ and $I_2(A) = y$. We say that y* ***E-preempts*** *x if $E_1(x) = E_2(y)$.*

(R_1 and R_2 need not be disjoint. Hence x and y may be the same variable. The definition makes it clear whether we talk of x as a variable in R_1 or of x as a variable of R_2.) Given $\rho_1 = \langle I_1, R_1, E_1\rangle$ and $\rho_2 = \langle I_2, R_2, E_2\rangle$ then $\rho_3 := \rho_1 \bullet \rho_2$ is formed as follows. First R_3 is defined. Let $R_1^1 := \{x^1 : x \in R_1\}$ and $R_2^2 := \{x^2 : x \in R_2\}$. Then let $S := \{y \in R_2 : (\exists x \in R_1)(I_1(y) = E_2(x))\}$ be the set of supervened referents and $R_3 := (R_1^1 \cup R_2^2) - S$. This construction ensures that the sum of the sets is disjoint. Next, we define two injections, $\iota_1 : R_1 \rightarrowtail R_3$ and $\iota_2 : R_2 \rightarrowtail R_3$ (where $\rightarrowtail$ indicates an injectice function), by

$$\begin{aligned} \iota_1(x) &:= x^1 \\ \iota_2(x) &:= \begin{cases} y^1 & \text{if } y \text{ supervenes } x \\ x^2 & \text{if } x \text{ is not supervened} \end{cases} \end{aligned} \tag{3.26}$$

The functions I_3 and E_3 are defined as follows (here $f(x) = \uparrow$ means that f is

Figure 3.1: Merge with nonidentical names

$$\begin{array}{|l|}\hline A : x : B \\ \hline x \\ \hline \phi(x) \\ \hline \end{array} \quad \bullet \quad \begin{array}{|l|}\hline C : x : D \\ \hline \varnothing \\ \hline \psi(x) \\ \hline \end{array} \quad = \quad \begin{array}{|l|}\hline A : x^1 : B \quad C : x^2 : D \\ \hline x^1 \\ \hline \phi(x^1) \\ \psi(x^2) \\ \hline \end{array}$$

Figure 3.2: Merge with identical names

$$\begin{array}{|l|}\hline A : x : B \\ \hline x \\ \hline \phi(x) \\ \hline \end{array} \quad \bullet \quad \begin{array}{|l|}\hline B : x : C \\ \hline \varnothing \\ \hline \psi(x) \\ \hline \end{array} \quad = \quad \begin{array}{|l|}\hline A : x^1 : C \\ \hline \phi(x^1) \\ \psi(x^1) \\ \hline \end{array}$$

undefined on x and $f(x) = \downarrow$ that f is defined on x).

$$I_3(A) \quad := \quad \begin{cases} I_1(A) & \text{if } I_1(A) = \downarrow \\ I_2(A) & \text{if } I_1(A) = \uparrow \text{ and } I_2(A) = \downarrow \\ \uparrow & \text{else} \end{cases}$$

(3.27)

$$E_3(u) \quad := \quad \begin{cases} E_2(I_2(E_1(x))) & \text{if } u = x^1 \text{ and } E_2(I_1(E_1(x))) = \downarrow \\ E_2(x) & \text{if } u = x^2 \text{ and } E_2(x) = \downarrow \\ E_1(x) & \text{if } u = x^1, E_1(x) = \downarrow \text{ and } x \\ & \qquad \text{is not E-preempted} \\ \uparrow & \text{else} \end{cases}$$

Definition 3.9 *Let $\nu_1 = [\rho_1 : \Gamma_1]$ and $\nu_2 = [\rho_2 : \Gamma_2]$ be two N-systems. The merge is defined as follows*

$$\nu_1 \bullet \nu_2 := [\rho_1 \bullet \rho_2 : \iota_1[\Gamma_1] \cup \iota_2[\Gamma_2]]$$

Here, $\iota_j[\Gamma_j]$ is the result of replacing every referent r occurring in a formula ϕ of Γ_j by the referent $\iota_j(r)$.

Let us now show how the N-systems solve our previous problem. We take again our sentence

(3.28) `A tall man sees a small rose.`

To get the desired translation we assume that there is exactly one name, †, so $N := \{\dagger\}$. Furthermore, determiners, adjectives and nouns get interpreted the same way:

(3.29)
$$\begin{array}{c}\texttt{/man/}\\ \boxed{\dagger : x : \dagger}\\ \begin{array}{|l|}\hline x\\ \hline \mathsf{man}'(x)\\ \hline\end{array}\end{array}\qquad \begin{array}{c}\texttt{/tall/}\\ \boxed{\dagger : x : \dagger}\\ \begin{array}{|l|}\hline x\\ \hline \mathsf{tall}'(x)\\ \hline\end{array}\end{array}\qquad \begin{array}{c}\texttt{/a/}\\ \boxed{\dagger : x : \dagger}\\ \begin{array}{|l|}\hline x\\ \hline x\\ \hline\end{array}\end{array}$$

The verb however has a more interesting N-system.

(3.30)
$$\begin{array}{c}\texttt{/sees/}\\ \boxed{\dagger : x : -, - : y : \dagger}\\ \begin{array}{|l|}\hline e, x, y\\ \hline \mathsf{see}'(e); \mathsf{act}'(e) \doteq x;\\ \mathsf{thm}'(e) \doteq y.\\ \hline\end{array}\end{array}$$

(This is a referent system even though the name † is used to identify two referents. Notice, namely, that $I(\dagger) = x$ and $E(x) = \uparrow$ as well as $E(y) = \dagger$. So, both E and I are partial injective functions.) First, let us translate /`a tall man`/. We get

(3.31)
$$\begin{array}{c}\texttt{/a/}\\ \boxed{\dagger : x : \dagger}\\ \begin{array}{|l|}\hline x\\ \hline x\\ \hline\end{array}\end{array} \bullet \left(\begin{array}{c}\texttt{/tall/}\\ \boxed{\dagger : x : \dagger}\\ \begin{array}{|l|}\hline x\\ \hline \mathsf{tall}'(x)\\ \hline\end{array}\end{array} \bullet \begin{array}{c}\texttt{/man/}\\ \boxed{\dagger : x : \dagger}\\ \begin{array}{|l|}\hline x\\ \hline \mathsf{man}'(x)\\ \hline\end{array}\end{array}\right)$$

$$= \begin{array}{c}\texttt{/a tall man/}\\ \boxed{\dagger : x^1 : \dagger}\\ \begin{array}{|l|}\hline x^1\\ \hline \mathsf{tall}'(x^1);\\ \mathsf{man}'(x^1)\\ \hline\end{array}\end{array}$$

Similarly, /`a small rose`/ will receive the translation

(3.32)
$$\begin{array}{c}\texttt{/a small rose/}\\ \boxed{\dagger : x^1 : \dagger}\\ \begin{array}{|l|}\hline x^1\\ \hline \mathsf{small}'(x^1);\\ \mathsf{rose}'(x^1)\\ \hline\end{array}\end{array}$$

(We will replace x^1 by x for readability.) Finally, if we combine these two with the verb, we get the following result.

(3.33)

$$\begin{array}{c}\texttt{/a tall}\\\texttt{man/}\\\begin{array}{|l|}\hline \dagger : x : \dagger \\\hline\hline x \\\hline \mathsf{tall}'(x); \\ \mathsf{man}'(x) \\\hline\end{array}\end{array} \bullet \left(\begin{array}{c}\texttt{/sees/}\\\begin{array}{|l|}\hline \dagger : x : -, - : y : \dagger \\\hline\hline e, x, y \\\hline \mathsf{see}'(e); \mathsf{act}'(e) \doteq x; \\ \mathsf{thm}'(e) \doteq y. \\\hline\end{array}\end{array} \bullet \begin{array}{c}\texttt{/a small}\\\texttt{rose/}\\\begin{array}{|l|}\hline \dagger : x : \dagger \\\hline\hline x \\\hline \mathsf{small}'(x); \\ \mathsf{rose}'(x) \\\hline\end{array}\end{array}\right)$$

$$= \begin{array}{c}/(3.28)/\\\begin{array}{|ll|}\hline \dagger : x^1 : -, & - : y^{12} : \dagger \\\hline\hline x^1, \quad y^{12}, \quad e^{12} & \\\hline \mathsf{tall}'(x^1); & \mathsf{small}'(y^{12}); \\ \mathsf{man}'(x^1); & \mathsf{rose}'(y^{12}); \\ \mathsf{see}'(e^{12}); & \mathsf{act}'(e^{12}) \doteq x^1; \\ \mathsf{thm}'(e^{12}) \doteq y^{12}. & \\\hline\end{array}\end{array}$$

The reader may check that in this example the merge is fully associative. Therefore, no constituent structure needs to be prescribed beforehand to arrive at the correct translation. This is, as was explained earlier, a welcome feature of the calculus. Nevertheless, it still suffers from various deficiencies. Notice that we have made no use of the names, only of the directionality of the system. So, a simple transitive sentence in an SVO language (or an OVS language, for that matter) will receive a correct translation simply because the verb can distinguish its arguments from the place they occupy with respect to it. The subject is to the left, the object to the right. In all other types, VSO, VOS, OSV and SOV, the verb cannot discriminate its arguments according to the direction. Some other means must be found. One possibility is morphological marking, and this is what we shall propose in a later section. At the moment, however, we shall pick up a rather delicate problem of the argument selection that is still unresolved in the present calculus.

A word on implementation. The actual implementation of variables is somewhat different. Variables are pairs of strings (encoding the type) and numbers. Only variables of identical type can ever be merged. Instead of adding a 1 or a 2 to the superscript, we apply the functions $n \mapsto 2n$ and $n \mapsto 2n + 1$. They too make the variable names disjoint. However, merge is done by default in such a way as to use an initial segment of the numbers for the variables. This is done in

order to keep the numbers small. Without compactification the numbers grow exponentially, while with compactification they only grow linearly. After ten merge operations we would thus end up with numbers in the thousands (without compactification), ten more operations would send these numbers into the millions. Thus, compactifying the names saves space (especially when displaying the variables) and it serves to keep them well below the threshold (natural numbers cannot be arbitrarily large, typically the limit is $2^{16} - 1$ or $2^{32} - 1$, depending on the machine you are running it on).

Exercise 15. Suppose we have a language of type SOV without case markers. For simplicity, pretend English to have that word order. Thus $N = \{\dagger\}$ as above. Rewrite the above lexicon to accomodate for this syntax. Calculate the interpretation of

(3.34) `a man a rose sees`

How does it differ from

(3.35) `a rose a man sees`

Does that conform to your intuition?

Exercise 16. Now try to develop a lexicon for the following language. The only grammatical sentences are /`he her likes`/, /`her he likes`/, /`she him likes`/, and /`him she likes`/. (Notice that gender serves as a marker for minimal meanings (male vs. female), while cases serve to identify grammatical function (subject vs. object).)

3.4 The Transmission of Referents

The previous section introduced referent systems and N-systems and showed how a basic English sentence gets the right translation. We have used referent systems

to combine semantics and syntax. The verb has an argument structure which requires the subject to be on the left hand side and the object on the right hand side. The original system has several drawbacks, however. One of them is that overt syntax determines the naming of referents. However, the order of constituents is already present in the morphology. For the handlers, introduced in Section 2.4, contain all necessary information on the formation of the constituent, hence also of directionality. Note that the left/right distinction is a rather crude diagnostic. It is unclear how we should deal with wrapping in terms of import and export names. Nevertheless, the notions of import and export name *do* play a role. Instead of using them to specify order, we shall define them with respect to the merge function: the import name is that name under which a referent is identified when the item is merged with another item, while the export name is that name under which that referent is made available after merge. Thus, import and export names are relevant at *abstract syntax*, also known as *tectogrammar*.

We now introduce a new notation. Rather than writing $\{A : x : B\}$ we put the variable first and write $\langle x : A \mapsto B\rangle$ to visualize that the name of x is transformed from A into B. Notice however that one of A and B can be missing. In that case we shall write $\langle x : A\rangle$, $\langle x : B\rangle$ or even $\langle x :\rangle$. However, since that notation would not reveal whether or not A or B are import or export names, we add a so-called diacritic. Writing $\langle x : \vartriangle : A\rangle$ means that x does not have an import name, only the export name A; and writing $\langle x : \triangledown : A\rangle$ we say that x as the import name A but no export name. Also, in place of $\langle x : A \mapsto B\rangle$ we write $\langle x : \Diamond : A \mapsto B\rangle$, or, if $A = B$, simply $\langle x : \Diamond : A\rangle$. The absence of names is coded in the diacritic. (We could have written instead $\langle x : - \mapsto A\rangle$, $\langle x : A \mapsto -\rangle$ or $\langle x : - \mapsto -\rangle$, but found it more visual to use diacritics. Also, it will turn out that the diacritic is one of the few pieces of information that must be shared across levels, in contrast to the actual names.)

Definition 3.10 (Diacritics) *A* ***(vertical) diacritic mark*** *is an element of* $V := \{\vartriangle, \triangledown\}$. *A* ***(vertical) diacritic*** *is a set of diacritic marks. We abbreviate the diacritics as follows. We write* $-$ *for* $\varnothing$, $\vartriangle$ *for* $\{\vartriangle\}$, $\triangledown$ *for* $\{\triangledown\}$ *and* $\Diamond$ *for* $\{\vartriangle, \triangledown\}$. *A triple* $\langle x : \partial : A \mapsto B\rangle$, *where A and B are names (or blank) is called an* ***argument identification statement (AIS)****. An AIS is said to* ***export*** *x (**under the name** B) if it contains* $\vartriangle$*; it is said to* ***import*** *x (**under the name** A) if it contains* $\triangledown$*. If one of A and B is absent or* $A = B$ *then* $\langle x : \partial : A \mapsto B\rangle$ *is abbreviated by* $\langle x : \partial : A\rangle$ *(or* $\langle x : \partial : B\rangle$*).*

Notice that "blank" stands for $-$. However, in practice it does not matter if that is

Figure 3.3: Merge with diacritics

$$\begin{aligned}
\langle x : \triangledown : A\rangle \bullet \langle x : \vartriangle : A\rangle &= \langle x : - : A\rangle \\
\langle x : \Diamond : A \mapsto B\rangle \bullet \langle x : \vartriangle : A\rangle &= \langle x : \vartriangle : B\rangle \\
\langle x : \triangledown : A\rangle \bullet \langle x : \Diamond : C \mapsto A\rangle &= \langle x : \triangledown : C\rangle \\
\langle x : \Diamond : A \mapsto B\rangle \bullet \langle x : \Diamond : C \mapsto A\rangle &= \langle x : \Diamond : C \mapsto B\rangle
\end{aligned}$$

a nonexistent name or something else. Thus we can also put an empty AVM there.

The diacritic $\triangledown$ means that the referent is *consumed* (so the argument structure in question is a functor with respect to that variable) and $\vartriangle$ means that the referent in question is *produced* (so the argument structure is an argument with respect to the referent). Instead of talking about consumption and production we may also think about passing the referent **down** ($\triangledown$) or **up** ($\vartriangle$). If we have the diacritic $\{\vartriangle, \triangledown\}$ then the referent is consumed and produced, or equivalently, it is passed up and down. A more standard terminology is the following. Let α be a set of AISs. We assume throughout that for every referent x there is at most one AIS that contains the referent x. If this is the AIS $\langle x : \partial : A \mapsto B\rangle$, then we say that the diacritic of x in α is ∂.

Definition 3.11 *Let α be a set of AISs. Then α is an x-**head** if the diacritic of x in α is $\triangledown$, an x-**argument** if the diacritic in α is $\vartriangle$, and an x-**adjunct** if the diacritic of x is $\Diamond$. α is an x-**carrier** if the diacritic of x is $-$.*

Translating the definition of merge into this new notation we obtain various cases listed in Figure 3.3. Recall that the functor is placed first. The precondition on merge is that the first diacritic ∂ contains $\triangledown$ and the second diacritic ∂' contains $\vartriangle$. In that case the middle names must match, and the outer names are retained. The resulting diacritic is $(\partial - \{\triangledown\}) \cup (\partial' - \{\vartriangle\})$. Notice the simplified notation $\langle x : \Diamond : A\rangle$. In case $A \neq B$ we speak of $A \mapsto B$ as a *transformation*.

Definition 3.12 *Let α and β be argument structures and x a referent. Suppose that x is shared in the merge $\alpha \bullet \beta$. Then α is called a **head** (**under merge**) relative to x (and β is called an **argument**), if the diacritic of x in α contains $\triangledown$, and if the diacritic of x in β contains $\vartriangle$.*

We will assume the following argument structures and representations for the En-

glish nouns, adjectives, determiners and transitive verbs:

$$(3.36)\qquad \begin{array}{c}\texttt{/man/}\\ \begin{array}{|l|}\hline \langle x : \vartriangle : \dagger\rangle \\ \hline x \\ \hline \mathsf{man}'(x) \\ \hline \end{array}\end{array} \quad \begin{array}{c}\texttt{/tall/}\\ \begin{array}{|l|}\hline \langle x : \lozenge : \dagger\rangle \\ \hline x \\ \hline \mathsf{tall}'(x) \\ \hline \end{array}\end{array} \quad \begin{array}{c}\texttt{/a/}\\ \begin{array}{|l|}\hline \langle x : \lozenge : \dagger\rangle \\ \hline x \\ \hline \varnothing \\ \hline \end{array}\end{array}$$

$$(3.37)\qquad \begin{array}{c}\texttt{/see/}\\ \begin{array}{|l|}\hline \langle e : \vartriangle : \dagger\rangle, \\ \langle x : \triangledown : \dagger\rangle, \\ \langle y : \triangledown : \dagger\rangle. \\ \hline e, x, y \\ \hline \mathsf{see}'(e); \mathsf{act}'(e) \doteq x; \\ \mathsf{thm}'(e) \doteq y. \\ \hline \end{array}\end{array}$$

The phrase /`a tall man`/ now receives the translation

$$(3.38)\qquad \begin{array}{c}\texttt{/a/}\\ \begin{array}{|l|}\hline \langle x : \lozenge : \dagger\rangle \\ \hline x \\ \hline \varnothing \\ \hline \end{array}\end{array} \bullet \left(\begin{array}{c}\texttt{/tall/}\\ \begin{array}{|l|}\hline \langle x : \lozenge : \dagger\rangle \\ \hline x \\ \hline \mathsf{tall}'(x) \\ \hline \end{array}\end{array} \bullet \begin{array}{c}\texttt{/man/}\\ \begin{array}{|l|}\hline \langle x : \vartriangle : \dagger\rangle \\ \hline x \\ \hline \mathsf{man}'(x) \\ \hline \end{array}\end{array} \right)$$

$$= \begin{array}{c}\texttt{/a tall man/}\\ \begin{array}{|l|}\hline \langle x^1 : \vartriangle : \dagger\rangle \\ \hline x^1 \\ \hline \mathsf{tall}'(x^1); \\ \mathsf{man}'(x^1). \\ \hline \end{array}\end{array}$$

The placement of the arguments (left or right) is dealt with by the morphosyntax. We use the annotation $\ogreaterthan$ if the argument is placed to the right, and $\olessthan$ if it is placed to the left. $\bigcirc$ is used if the variable is not imported. Hence we write

$$(3.39)\qquad \begin{array}{c}\texttt{/man/}\bigcirc\\ \begin{array}{|l|}\hline \langle x : \vartriangle : \dagger\rangle \\ \hline x \\ \hline \mathsf{man}'(x) \\ \hline \end{array}\end{array} \quad \begin{array}{c}\texttt{/tall/}\ogreaterthan\\ \begin{array}{|l|}\hline \langle x : \lozenge : \dagger\rangle \\ \hline x \\ \hline \mathsf{tall}'(x) \\ \hline \end{array}\end{array} \quad \begin{array}{c}\texttt{/a/}\ogreaterthan\\ \begin{array}{|l|}\hline \langle x : \lozenge : \dagger\rangle \\ \hline x \\ \hline \varnothing \\ \hline \end{array}\end{array}$$

Let us now turn to the verb. Here we discover a problem. If the morphosyntax tells us that the verb has two arguments, one subject and one object, when performing

the merge in semantics we do not know which one of the referents is subject and which one is object. To put it more precisely: we do not know which argument is consumed first, and which one last. Since morphosyntax and argument structure are independent, there must be a way to secure a link between them. This link is given by the fact that both are *sequences of identical length*. The morphosyntax tells us that the there is a subject (to the left) and an object (to the right) in that order. The argument structure of /`see`/ likewise orders x (subject) before y. In English, referents must be dealt with from right-to-left. Hence the object is the first to be identified under merge.

(3.40)

$$\begin{array}{c} /\texttt{see}/\bigcirc, \circledless, \circledgtr \\ \boxed{\begin{array}{l} \langle e : \vartriangle : \dagger \rangle, \\ \langle x : \triangledown : \dagger \rangle, \\ \langle y : \triangledown : \dagger \rangle. \end{array}} \\ \boxed{\begin{array}{l} e, x, y \\ \hline \mathsf{see}'(e); \mathsf{act}'(e) \doteq x; \\ \mathsf{thm}'(e) \doteq y. \end{array}} \end{array}$$

It is worth noting that the order is also necessary to account for default orders of arguments. The availability of an SV constituent is a consequence of the lack of order in the argument structure. There is no way to tell a verb in which way it needs to consume its arguments. In addition to the existence of a subject-verb constituent (which one might actually want to have), there are more problems which definitely call for a solution. These are the fact that focus projection in German can spread to the VP if the object has not been scrambled (an observation due to Tilman Höhle, see Haider 1993). In English we also need to account for order with ditransitive verbs. A ditransitive verb in English must be able to distinguish which of its objects is the first (direct) and which is the second (indirect) object.

(3.41) `They called him an idiot.`

(3.42) `He gave Albert the car.`

Notice namely that inverting the order of the objects results in sentences that are ungrammatical under the same reading as the corresponding (a) sentences.

(3.43) *`They called an idiot him.`

(3.44) *`He gave the car Albert.`

Hence, the verb is forbidden to compose with the indirect object first. However, syntactically, nothing distinguishes an indirect object from a direct object, not

even animacy. It is possible to say /`John gave the farmer the slave.`/ as well as /`John gave the slave the farmer.`/. Hence, if the argument structure is the same we must conclude that the verb can keep track of the order in which the relevant arguments appear.

Finally, let us turn to the mechanics of names. There are two changes we need to make. The first is the change to attribute value matrices, as defined in Section 2.3. This means the following. Names will be treated as mutually exclusive elements; AVMs denote sets of such names. An $\mathcal{S}$-atom is thus nothing but a name. We shall assume some feature space $\mathcal{S}$, whose precise nature hardly matters, however, except to calculate the set of names. Therefore, the dependency on $\mathcal{S}$ will be backgrounded in the sequel. The other change is a change in the way we read pairs of AVMs. In the previous chapter we thought of these pairs as containing two independent AVMs. However, this is not the most interesting way to understand them. This is because the essential nature of referent systems consists in the ability to change names. Let us take for example the situation where a variable is associated with two names. When we allows AVMs in place of A and B, we need to consider what happens with underspecification. If x is associated with the pair (f, g) this would prima facie mean that x is imported under any name falling under f and exported under any name falling under g. However, this is actually the least common situation. It is more common to understand that underspecified input names are passed on as they are in the concrete case. We shall explore that last option later however. First, we look at the general setting.

In full generality, we will say that in case of a transformation a variable is paired with a *relation* R from input names to output names. If x is paired with R as functor, and is identified with y as argument, where y is in turn paired with the relation S, then after merge x^1 will be paired with $S \circ R$, where $A\,(S \circ R)\,B$ if and only if there is a C such that $A\,S\,C\,R\,B$. So, the input name of x is required to be identical to the output name of y, namely C. The input name of x^1 is then the input name of y for output C, while the output name of x^1 is the output name of x for input C.

This is the generic case. When y has no import name or x no output name, we proceed in the analogous way. In that case the resulting variable is not associated with a relation but with a plain set of names. We end up with the constructs $S \upharpoonright C$ (x is not exported) and $R \upharpoonleft B$ (y is not imported).

$$
\begin{aligned}
S \upharpoonright C &:= \{(M, N) : (M, N) \in S, N \in C\} \\
R \upharpoonleft B &:= \{(M, N) : (M, N) \in R, M \in B\}
\end{aligned}
\tag{3.45}
$$

If both is the case and the merge succeeds, we have an empty AIS.

This defines merge of pairs in the most general case. However, we will not need such generality. By far the most important situation is where the relation is such that whatever value an attribute has in f, its value is retained in g. For this we introduce the special notation $[a : \checkmark]$, to be used in B only. It says that whatever value is chosen as input, this will be the output value as well. The mechanics of this value is explained below.

Definition 3.13 (Copy AVM) *Let $\mathcal{S} = \langle A, V, \mathrm{rg}\rangle$ be a feature space. An $\mathcal{S}$-**copy matrix** is a partial function $f : A \hookrightarrow \wp(V) \cup \{\checkmark\}$. A **relational AVM** is a pair (f, g), where f is a $\mathcal{S}$-matrix and g a $\mathcal{S}$-copy matrix.*

The difference between a matrix and a copy matrix is that a copy matrix is allowed to have the value $\checkmark$ in place of a subset of V. It is assumed that $\checkmark \notin \wp(V)$.

Definition 3.14 (Subsumption between Relation AVMs) *Let (f, g) and (f', g') be two relational AVMs. We have $(f', g') \leq (f, g)$ iff*

- *$f \leq f'$, and*
- *for all $a \in A$ one of the following applies.*
  - *$g(a)$ and $g'(a)$ are each either undefined or $= \checkmark$;*
  - *$g(a) = g'(a) = \varnothing$;*
  - *$g(a)$ and $g'(a)$ are both defined and do not equal $\varnothing$ or $\checkmark$ and $g'(a) \subseteq g(a)$; or*
  - *$g'(a)$ is undefined or $\checkmark$, $g(a)$ is defined, $f'(a) = \{v\}$ and $v \in g(a)$.*
  - *$g(a)$ is undefined or $\checkmark$, $g'(a)$ is defined, $f(a) = \{v\}$ and $g'(a) = \{v\}$.*

*If both $(f', g') \leq (f, g)$ and $(f, g) \leq (f', g')$, then (f, g) and (f', g') are said to be **equivalent**.*

This definition mixes two separate things: a convention, whereby an undefined attribute in the second AVM is to be identified with $\checkmark$ (and with $\top$ in the first AVM). And secondly, the meaning that $\checkmark$ has, namely that in particular when the

input values are reduced to just one, say v, then putting $g(a) = \{v\}$ is the same as putting $g(a) = \checkmark$. (The last two clauses deal with this second case.) Indeed, these two amount to exactly the same.

This definition effectively settles the question which relation between names is denoted by a relational AVM. However, below we shall return to this question in more detail. For now we complete the definition of argument structure.

Definition 3.15 (Argument Structure) *An* ***argument identification statement*** *or (**AIS**) is a triple $\langle x : \partial : (A, B)\rangle$, where x is a referent, ∂ a diacritic and (A, B) is a relational AVM such that $A = [\quad]$ whenever $\triangledown \notin \partial$ and $B = [\quad]$ whenever $\vartriangle \notin \partial$. If $\partial = -$ (and therefore $(A, B) = ([\quad], [\quad])$, the AIS is called* ***empty****. B may additionally contain the symbol $\checkmark$. An* ***argument structure*** *is a sequence $\alpha = \langle \mu_i : 1 \leq i \leq n\rangle$ of AISs such that if $n > 1$ then μ_1 is not empty.*

What we need to define next is the merge of two relational AVMs (f, g) and (h, k). The result is again a relational AVM. The merge can be defined for each attribute separately. Before we do so, however, we need to talk about equivalence.

Proposition 3.16 (Equivalence) *If in a specification of $P = (f, g)$, f is undefined on an attribute b then P is equivalent to $P' = (f', g)$, where f' is f augmented by $[b : \top]$. If g is undefined on b, P is equivalent to $P' = (f, g')$, where g' is g augmented by $[b : \checkmark]$. Further, if $f(b) = \{v\}$ and $g(b) = \{v\}$, then P is equivalent to $P' = (f, g')$, where g' is identical to g except that $g'(b) = \checkmark$.*

By this observation, (3.46) can be simplified to $([\textsc{case} : \star], [\textsc{case} : \mathit{dat}])$ (singletons are written without brackets). Notice that equivalence is defined for relational AVMs. If $P = (f, g)$, $P' = (f', g')$ where g and g' are complete and free of $\checkmark$, then P is equivalent to P' iff $f \equiv f'$ and $g \equiv g'$ in the usual sense of equivalence between AVMSs.

However, we are after a different notion here, since the pairs code functions from name sets to name sets. And here $\checkmark$ plays a special role. The notational convention is useful because it simplifies the matter as follows. We only have to deal with the case of an attribute that has a value in all four AVMs. (For if not, we can put one in using the Proposition 3.16.) So, we assume that f contains $[a : s_1]$, g contains $[a : s_2]$, h contains $[a : s_3]$ and k contains $[a : s_4]$, where the s_i may be sets of values or $\checkmark$ $(i = 2, 4)$. Special attention must be paid to the case of empty sets.

Definition 3.17 (Merge of AVM Pairs) *The merge of* $([a : s_1], [a : s_2])$ *and* $([a : s_3], [a : s_4])$ *is defined if and only if*

- $s_2 = \checkmark$ *and* $s_1 \cap s_3 \neq \varnothing$*; or*
- $s_2 \neq \checkmark$ *and* $s_2 \cap s_3 \neq \varnothing$.

If the merge is defined, its value is determined as follows.

- $s_2 = \checkmark$ *and* $s_4 = \checkmark$*:* $([a : s_1 \cap s_3], [a : \checkmark])$.
- $s_2 = \checkmark$ *and* $s_4 \neq \checkmark$*:* $([a : s_1 \cap s_3], [a : s_4])$.
- $s_2 \neq \checkmark$ *and* $s_4 = \checkmark$*:* $([a : s_1], [a : \checkmark])$.
- $s_2 \neq \checkmark$ *and* $s_4 \neq \checkmark$*:* $([a : s_1], [a : s_4])$.

Example 8. Here is an interesting contrast. Consider first the Latin adjective. The form /`parvum`/ could be neuter nominative or accusative (in addition to accusative for the masculine). Thus we have as values for CASE the set $\{nom, acc\}$. When it modifies a noun, say /`templum`/, it takes whatever case that noun has (nominative or accusative) and the resulting constituent has that same case. It turns out, though, that in the neuter there is no distinction in nouns either. In this case, we can use for adjectives the pair $([\textsc{case} : \{nom, acc\}], [\textsc{case} : \{nom, acc\}])$. Though it seems logically cleaner to use $([\textsc{case} : \{nom, acc\}], [\textsc{case} : \checkmark])$, the morphological facts do not decide the situation either way.

Consider on the other hand the dative case ending /`nek`/ in Hungarian. It can be added to nouns in the singular or plural. However, if the noun is in the singular, so is the corresponding noun in the dative (/`ember`/ vs. /`embernek`/), and if the noun is in the plural, so is the dative form (/`emberek`/ vs. /`embereknek`/). Therefore, in this case we say that the dative suffix contains the pair

$$\left(\begin{bmatrix} \textsc{case} : \star \\ \textsc{num} : \top \end{bmatrix}, \begin{bmatrix} \textsc{case} : \{dat\} \\ \textsc{num} : \checkmark \end{bmatrix}\right) \tag{3.46}$$

It is possible to write two entries instead (one for singular, one for plural). However, this would mean to miss an obvious generalisation. Moreover, there are more

attributes to be taken care of, and they require the same treatment, multiplying the number of entries even further. ⚽

We shall define equivalence of argument structures as follows.

Definition 3.18 *Two argument structures α and β are* ***equivalent*** *iff β can be obtained from α by adding or removing empty AISs. α is* ***reduced*** *iff it has no empty AISs.*

The rationale behind this definition is that empty AISs do not contribute to the merge (they might however block merge in certain cases). In fact, we assume that merge or fusion will always result in a reduced argument structure. We shall make the following assumption, which is a kind of well-formedness requirement.

Restriction 1 *In an $\langle \mu_i : 1 \leq i \leq n \rangle$, if some μ_i exports its referent, then also μ_1 exports its referent.*

Indeed, it is possible for there to be several exported referent, though that possibility is apparently used rather sparingly in language.

Notice that no requirement is made any more that the exported or the imported names be distinct for distinct referent. In fact, this is quite crucial for the way we handle the merge. The distinctness is anyway not needed since the structure is now ordered. We demand that in a lexical argument structure, it is the first identification statement that carries $\vartriangle$ or $\lozenge$.

How can the merge be defined? Clearly, by default, we assume that the order in the sequence matches the order in which the arguments can be taken. For example, the English verb /`give`/ will get (among other) the following semantic structure.

$$(3.47)\qquad \begin{array}{c} /\texttt{give}/\bigcirc, ⧀, ⧁, ⧁ \\ \begin{array}{|ll|} \hline \langle e : \vartriangle : \dagger \rangle, & \langle x : \triangledown : \dagger \rangle, \\ \langle y : \triangledown : \dagger \rangle, & \langle z : \triangledown : \dagger \rangle. \\ \hline \end{array} \\ \begin{array}{|ll|} \hline e, x, y, z & \\ \hline \mathsf{give}'(e); & \mathsf{act}'(e) \doteq x; \\ \mathsf{thm}'(e) \doteq z; & \mathsf{goal}'(e) \doteq y. \\ \hline \end{array} \end{array}$$

We assume that the list of arguments must be processed from right to left and bottom to top. So, the structure takes first the direct object, then the prepositional

phrase, and combines with the subject. The morphosyntax likewise allows for a sequence of argument and puts them such that the subject ends up on the left side, while the other arguments are on the right side of the verb. Notice also that an AIS for the event has been added. Previously, it was left out for the reason that the same name would be exported.

Exercise 17. This exercise provides some background theory. Take a look at (3.2). It was stated there that the veriables occurring in this sequence are x_1 through x_n, in that order. This requirement can be lifted entirely. This goes in two parts. First, as long as the names are different, the names are irrelevant. Second, we do not even need to require the names to be distinct. To that effect, take a look at these two structures.

(3.48)

$\sigma_1, \sigma_2, \cdots, \sigma_n$
V
$\Delta; x_i \doteq x_j$

$\sigma_1, \sigma_2, \cdots, [x_j/x_i]\sigma_i, \cdots, \sigma_n$
$[x_j/x_i]V$
$[x_j/x_i]\Delta$

These structures are identical for all intents and purposes.

Notation. As usual, $[y/x]U$ denotes the result of replacing free occurrences of x by y in U, where U can be a formula, DRS, a set of formulae or DRSs, or a set of variables.

Exercise 18. (Eliminating adjuncts.) Suppose we have an AIS $\langle x : \diamond : A \mapsto B\rangle$. Replace this with the sequence $\langle x : \vartriangle : B\rangle, \langle y : \triangledown : A\rangle$ (in this order). Show that this allows to eliminate adjuncts in argument structure (by chosing y in an appropriate way). How does the semantics have to be adapted to accommodate for this elimination? Given the previous exercise, show that we could also use $\langle x : \vartriangle : B\rangle, \langle x : \triangledown : A\rangle$ with no change in the semantics.

Exercise 19. There is a correlation between subsumption and the induced relational on names. Let $P = (f, g)$ be a relational AVM. Let

$$\widehat{P} := \{(f', g') : f', g' \ \mathcal{S}\text{-atoms and } (f', g') \leq (f, g)\} \qquad (3.49)$$

Now show that for $P' = (f', g')$ we have $P' \leq P$ if and only if $\widehat{P'} \subseteq \widehat{P}$.

Exercise 20. We can use $\vee$, $\wedge$ and $\neg$ on relational AVMs as well. The interpretation is the union, intersection and complement (within the feature space). Determine the following disjunctions (where $w \neq v$).

$$\begin{array}{l}([a:\{v\}],[a:\checkmark]) \vee ([a:\{v\}],[a:\{w\}]) \\ ([a:\{v\}],[a:\checkmark]) \vee ([a:\{w\}],[a:\checkmark])\end{array} \qquad (3.50)$$

3.5 Signs

We shall now turn to the definition of merge and show how basic syntactic facts follow directly from the design of the semantic structures. Before we begin, we need to clarify a few things. First, the referent systems are not to be identified with the head section of the DRS. Instead, we assume that a variable assumes its quantificational force directly from the place where it first occurs. So, we will stop writing a DRS like this: $[x : \mathsf{man}'(x)]$. Rather, we work with the implicit assumption that x occurs in the head section of the highest box that contains it. This is of course not a necessary assumption. It is feasible to assume that our structures are pairs consisting of a referent system, and a genuine DRS, which in turn consists of a head section and a body. Such stacked structures will be necessary to do binding, but for syntactic purposes we can dispense with them. Next we need to see how the merge is defined.

Our basic assumption is that every syntactic merge is accompanied by a semantic merge. So, we assume that whenever two structures $\mathfrak{S}_1$ and $\mathfrak{S}_2$ are merged, so is their meaning. Our syntactic structures are what is called *sign* in the literature. A sign consists in (a) a semantic unit, (b) a syntactic unit and (c) a morphological unit.

Definition 3.19 (Sign) *A **sign** is a triple $\mathfrak{S} = \langle m, \alpha, \Delta \rangle$, where m is a morpheme (= a set of morphs), α an argument structure and Δ a DRS such that every unbound referent of Δ occurs in α. Each morph of m is required to have the same dimension, which must be equal to the length of α.*

I remark here that the condition that unbound variables of Δ must have an occurrence in α has been added with reference to the problems of merge mentioned earlier. As long as we keep nameless referents in the argument structure this requirement is harmless; however, when we require nameless referents to be eliminated from the argument structure, we can only do so if the referent is added in the head section of an appropriate DRS. This is evidently a tricky issue, however one that I shall not pursue.

The merge of two signs is results in the following sign. Let $\mathfrak{S}_1 = \langle \mathfrak{m}, \alpha, \Delta \rangle$ and $\mathfrak{S}_2 = \langle \mathfrak{n}, \beta, \Theta \rangle$. Then we define the merge as follows.

(3.51) $$\mathfrak{S}_1 \oplus \mathfrak{S}_2 := \langle \mathfrak{m} \bullet_f \mathfrak{n}, \alpha \bullet \beta, s[\Delta^1 \cup \Theta^2] \rangle$$

In this merge, $\mathfrak{S}_1$ is the **functor** and $\mathfrak{S}_2$ **argument**. Notice that the functor is always written on the left. The substitution s is computed from the merge of the argument structures. Moreover, the merge of morphemes $\mathfrak{m} \bullet_f \mathfrak{n}$ is computed based on the occurring morphs. The definition of their merge however has remained incomplete. We have used the symbol $\star$ for this function in Definition 2.22. Its full identity will be revealed in Definition 3.27 below.

Thus, the computation is in three steps.

1. Compute $\alpha \bullet \beta$ first. This results in a substitution s and a so-called pairing function f, which depends on α and β.

2. Compute the DRS $s[\Delta^1 \cup \Theta^2]$.

3. Compute $\mathfrak{m} \bullet_f \mathfrak{n}$ based on f.

$\oplus$ is the *merge of representations*.

The definition of merge is split into several cases. First, we shall define the notion of *access*; there are two kinds of access restrictions, exemplified by English and German. Second, we distinguish *merge* from *fusion*; and finally, we distinguish between *monadic* and *polyadic* merge (and, similarly, *monadic* and *polyadic* fusion). Let us begin with the problem of access to individual AISs within an argument structure. In contrast to the original conception of referent systems we have argued that argument structure is not a *set* of AISs but a *sequence* thereof.

Example 9. Many tests show that word order in English is not free. (3.53) cannot mean the same as (3.52). Hence it is semantically odd, though it is syntactically well-formed.

(3.52) `John saw the book.`

(3.53) `?The book saw John.`

If we assume that the verb takes subject and object as its two arguments, placing the subject before and the object after, the oddness of (3.53) is already accounted for. What is not accounted for, however, is the fact that the verb forms a constituent exclusively with its object (though some contest that, see Steedman 1990).

However, the contrast between (3.41) and (3.43), repeated here as (3.54) and (3.55), still needs accounting for.

(3.54) `They called him an idiot.`

(3.55) `*They called an idiot him.`

In German, on the other hand, word order is basically free. The six permutations of the arguments can mean the same (that father gave the key to the director), though they do not all sound as natural.

(3.56) `Der Vater gibt dem Direktor den Schlüssel.`
the-NOM father gives the-DAT director the-ACC key

(3.57) `Der Vater gibt den Schlüssel dem Direktor.`

(3.58) `Dem Direktor gibt der Vater den Schlüssel.`

(3.59) `Dem Direktor gibt den Schlüssel der Vater.`

(3.60) `Den Schlüssel gibt der Vater dem Direktor.`

(3.61) `Den Schlüssel gibt dem Direktor der Vater.`

The fact that the verb occupies the second place needs accounting for. In subordinate clauses this is not so:

(3.62) `..., dass der Vater dem Direktor den Schlüssel gibt.`
... that the-NOM father the-DAT director the-ACC key gives

(3.63) `..., dass der Vater den Schlüssel dem Direktor gibt.`

(3.64) `..., dass dem Direktor der Vater den Schlüssel gibt.`

(3.65) `..., dass dem Direktor den Schlüssel der Vater gibt.`

(3.66) `..., dass den Schlüssel der Vater dem Direktor gibt.`

(3.67) `..., dass den Schlüssel dem Direktor der Vater gibt.`

All arguments are to the left, and the order is basically free. We shall see later that there nevertheless exist differences between them. The conclusion is that there is one argument structure, ordering the arguments as shown in (3.62), that is, first the subject, then the indirect object and then the direct object, see next section. However, this argument structure allows for alternative word orders as well. ⚽

In later chapters we shall develop a slightly more articulated view on that matter. The definition of access can be given two forms: either we talk about AISs or we talk about the variable that these AISs contain. We prefer the latter version. Notice however that while in a given argument structure different AISs have different variables, this need not be true across two AISs. For the purpose of the next definitions we assume that the variables of the first argument structure are x_i and the second structure has only one variable, y_1. (Thus, in the terminology of the definition below we are dealing with monadic merge.)

Let $\alpha = \langle \mu_i : 1 \leq i \leq m \rangle$ and $\beta = \langle \nu \rangle$ be argument structures. Let μ_i contain the variable x_i and ν the variable y. The merge $\alpha \bullet \beta$ is defined only if y accesses a variable x_k of α and $\mu_k \bullet \nu$ is defined. This is the centerpiece of the definition of merge (and fusion).

Let us explain this in some more detail. If $\mu_k \bullet \nu$ is defined, by our notational convention μ_k imports its referent and ν exports its referent and the names match. Now everything hinges on the notion of access. We assume that access is not uniform across languages. For example, English generally has strict access, while German for example has a more liberal access rule, allowing to jump over an AISs if the feature specifications do not match.

Definition 3.20 (Access) *Let $\alpha = \langle \mu_i : 1 \leq i \leq m \rangle$ be an argument structure. And let ν an argument identification statement.*

- *ν **E-accesses** μ_k iff $k = m$ and $\mu_m \bullet \nu$ succeeds.*
- *ν **G-accesses** μ_k iff k is the largest index such that $\mu_k \bullet \nu$ succeeds.*

Notice that access is defined without recourse to the variables. If μ_k contains the variable x_k and ν the variable y, we do however also say that y accesses x_k.

(Actually, there is one occasion where match is determined by the variable: we assume variables to be typed, and that type is represented in the non-numerical part, for example 'e' versus 'x' in (3.68). However, the type can in principle be coded into the names as well, so we are not dealing with it here.)

The idea behind these definitions is as follows. Suppose that α is the argument structure of a verb looking for the following arguments:

(3.68)
$$\begin{array}{ccccc} e & x_1 & x_2 & x_3 & x_4 \\ & \vdots & \vdots & \vdots & \vdots \\ & [c : \textsc{nom}] & [c : \textsc{dat}] & [c : \{\textsc{nom}, \textsc{acc}\}] & [c : \textsc{all}] \end{array}$$

The diacritics are as follows. e has $\vartriangle$, all others have $\triangledown$. Let the arguments all be to the right. Now, let β be the argument structure of an NP. So, it may be depicted by

(3.69)
$$\begin{array}{c} y \\ \vdots \\ [c : \gamma] \end{array}$$

Then, with E-access the merge $\alpha \bullet \beta$ where $\beta = \langle v \rangle$ will succeed only if CASE matches with $[c : \textsc{all}]$. For $[c : \gamma]$ must match the last entry for β, which is to say that $\textsc{all} \in \gamma$. If that is the case, y E-accesses x_4. If G-access is assumed, the situation is different. If γ contains ALL then y accesses x_4. If γ contains NOM or ACC but not ALL then y accesses x_3. If γ contains DAT but none of ALL, ACC or NOM, then y_1 accesses x_2. And so on. It is clear that if γ contains NOM then y does not access x_1, but x_3. (In the case of G-access, if names are sets, then it is just required that the intersection is not empty. So the first potential candidate for matching is taken.)

Example 10. Access need not be a global option for languages. While German verbs seem to prefer G-access, this is not so for adjective. Consider the following contrast.

(3.70) `der auf seine Schüler stolze Lehrer`

(3.71) *`der stolze Lehrer auf seine Schüler`

The German adjective /`stolz`/ ('proud') has the translation $\mathsf{proud}'(x, y)$. It is an adjunct with respect to x but has another argument y, the person or things that x

is proud of. If we merge it with a noun, then the adjective must be a head. Hence the referent x will be identified first, skipping y. The data suggests that this cannot be the case. So we must combine first with the phrase /`auf seine Schüler`/ ('of his pupils'). The phrase /`auf seine Schüler stolz`/ is an adjunct, as is the adjective /`stolz`/. ⚽

A way to solve the problem of nonglobal G-access is presented through the so-called *noskip* feature, see Section 4.6. Diacritics may contain a feature specifying that the particular variable cannot be skipped. Thus, access cannot look beyond them. E-access assumes noskip to be set everywhere, G-access assumes it to be set nowhere.

The syntactic restrictions should therefore be a consequence of the restrictions on combining argument structures. We will investigate this here with respect to basic syntax. First, as with argument structures, we take it that there exists a lexical and a functional merge. We assume that lexical elements can only lexically merge, but functional elements have the choice of merging functionally or lexically. First, let us put down the most important of all restrictions.

Restriction 2 *A* ***merge*** *of representations can take place only if at least one referent is identified.*

This condition holds for all types of merge and ensures that only those parts of speech are combined which share some common object about which they speak.

Definition 3.21 (Saturated Argument Structure) *An argument structure is* ***saturated*** *if none of its AISs imports any of its referents.*

Definition 3.22 $\alpha \bullet \beta$ *is a* ***merge*** *if* β *is saturated. Otherwise* $\alpha \bullet \beta$ *is an instance of* ***fusion****. A* $\alpha \bullet \beta$ *is* ***monadic*** *if* β *exports exactly one variable.*

We distinguish between monadic and polyadic merge depending on how many referents β exports. The standard case is the monadic merge. In a monadic merge we have $\alpha = \langle \mu_i : 1 \leq i \leq m \rangle$, and $\beta = \langle \nu_1 \rangle$, where ν_1 does not import y_1. The polyadic case is more involved as it potentially involves more variables to be identified in a single operation. However, we speak of merge in case β is saturated, so is not in need of arguments.

In fusion, both argument structures may be unsaturated. The availability of fusion has important consequences. If we only used merge then constituents have to be fully saturated in syntax, so no argument can ever be omitted. If the constituents are strings then that would entail that they have to be continuous. An adverbial could then not modify a verb unless the latter is fully saturated. This however is usually difficult to square with the word order facts. The option of discontinuity however makes arguing that case difficult.

In contrast to merge, fusion requires a number of additional decisions concerning the fate the arguments that the complement brings into the new structure. Suppose that α is fused with β and that α is in need of the arguments γ_1, γ_2, while β needs the argument δ. Then in which order does $\alpha \bullet \beta$ need its arguments? There are three choices that come to mind:

$$\begin{array}{lccc} & \delta & \gamma_1 & \gamma_2 \\ (3.72) & \gamma_1 & \delta & \gamma_2 \\ & \gamma_1 & \gamma_2 & \delta \end{array}$$

We choose the third option. That is to say we assume that fusion delays the consumption of arguments only temporarily.

Definition 3.23 (Monadic Merge and Fusion) *Let $\alpha = \langle\mu_1, \cdots, \mu_m\rangle$ be and $\beta = \langle\nu_1, \cdots, \nu_n$ be AISs that exports a single variable. Then $\alpha \bullet \beta$ succeeds iff ν_1 accesses some μ_i. In that case $\alpha\bullet\beta = \langle\mu_1, \cdots, \mu_{i-1}, \mu_i\bullet\nu_1, \mu_{i+1}, \cdots, \mu_m, \nu_2, \cdots, \nu_n\rangle$. If β is saturated, the operation $\alpha\bullet\beta$ is called **monadic merge**, otherwise a **monadic fusion**.*

Finally, we must consider a last possibility: that merge identifies several variables in one step. Such a merge will be called *polyadic*.

Definition 3.24 (Polyadic Merge) *Let $\alpha = \langle\mu_1, \ldots, \mu_m\rangle$ and $\beta = \langle\nu_1, \ldots, \nu_n\rangle$ be argument structures. Then the **polyadic merge** $\alpha\bullet\beta$ succeeds iff there is a sequence $i_1, \cdots, i_n$ such that*

① *$i_1, \ldots, i_p$ is strictly descending;*

② *$\mu_{i_k} \bullet \nu_k$ is a rightward merge for all $k \leq n$;*

③ *all and only the AISs of β that export a referent are in the list of ν_{j_k};*

④ *the monadic merge* $\alpha \bullet \langle \nu_1 \rangle$ *succeeds.*

In this case, the resulting argument structure is $\langle \xi_g : g \leq m \rangle$*, where* $\xi_g := \mu_{i_k} \bullet \nu_{j_k}$ *if* $g = i_k$ *for some* $k \leq n$*, and* $\xi_g := \alpha_g$ *else. The merge is called n-**adic**.*

(In this definition, some of the ξ_i will be empty, in which case they will be dropped. However, for the purpose of the definition it is easier to keep them first.)

This is the definition of merge; merge requires that the argument β does not take arguments of its own. If it is does, we speak of *fusion*. The definition is more involved.

Definition 3.25 (Polyadic Fusion) *Let* $\alpha = \langle \mu_1, \dots, \mu_m \rangle$ *and* $\beta = \langle \nu_1, \dots, \nu_n \rangle$ *be argument structures. Assume that the indices j for which* β_j *exports its referent are numbered* $j_1, \cdots, j_p$*,* $p > 0$*, in ascending order, with* $j_1 = 1$*. Let* $\gamma = \langle \kappa_1, \cdots, \kappa_{n-p} \rangle$ *be the sequence that results from* β *after removing the subsequence* $\langle \nu_{i_1}, \nu_{i_2}, \cdots, \nu_{i_p} \rangle$*. Then the **polyadic fusion*** $\alpha \bullet \beta$ *succeeds iff there is a sequence* $i_1, \cdots, i_p$ *and such that*

① $i_1, \dots, i_p$ *is strictly descending;*

② $\mu_{i_k} \bullet \nu_{j_k}$ *is a rightward merge for all* $k \leq p$*;*

③ *the monadic fusion* $\alpha \bullet \langle \nu_1 \rangle$ *succeeds.*

In this case, the resulting argument structure is $\langle \xi_g : g \leq m + n - p \rangle$*, where*

$$(3.73) \qquad \xi_g := \begin{cases} \mu_{i_k} \bullet \nu_{j_k} & \text{if } g = i_k \text{ for some } k \leq p, \\ \alpha_g & \text{else, if } g \leq m, \\ \kappa_{g-m} & \text{if } g > m. \end{cases}$$

*The fusion is said to be p-**adic**.*

The sequence $j_1, \cdots, j_p$ is constructed first. It contains the AISs of β that export a referent. Notice the special role of j_1, which is set to 1. By clause ③, the monadic fusion between α and $\langle \nu_1 \rangle$ must succeed. This means that the principal element to determine the fate of the fusion is α_{i_1}. Notice that the number i_1 is determined by the rules of access. Thus we have the following logical sequence.

1. Start with v_1 and determine the number i_1 according to the rules of access. This number is called the *pivot*.

2. Determine in sequence the other numbers i_k. This constitutes the *pairing function*; it determines essentially in which way the AISs of α are paired with the AISs of β.

Notice that polyadic fusion need not be defined by means of monadic merge; this is a rather indirect way. More directly, the specification is via access. We have opted for the indirect definition in order to start with a simpler case.

In a polyadic merge, the rules for choosing i_1 are as in the monadic merge. The rules are therefore those of access. The other i_k, $k > 1$, must form a decreasing sequence. In fact, we hardly need more than two simultaneous identifications, so only i_2 needs to be chosen. I discuss the choice of i_2, the general case is then clear enough. We require E-access only for the choice of i_1; when choosing i_2 we use G-access (see however the discussion on strictness in Section 4.6). That is to say, eliminate from α the member μ_{i_1} and every member following it. Then i_2 is the index such that x_{i_2} is G-accessed by y_{j_2} in this reduced structure.

Moreover, the exported variables of the argument must all be merged; none may be left out. If we attempt a merge between α and β then the choice of monadic versus polyadic merge is determined solely by the nature of the argument, not the functor. Every referent that the AISs of the argument export must be merged into the functor. One may wonder whether polyadic E-merge should actually be stricter and require that the ascending sequence be uninterrupted. This would be like an iterated E-merge, discharging one pair after the other. In practice, there is no situation where this makes a difference. (In Section 4.6 we discuss some diacritics. One of them is the “noskip” diacritic. It allows to specify arguments that cannot be skipped in polyadic fusion.)

Now that we have defined the merge of argument structure let us proceed to the other components.

Definition 3.26 (Pairing Function) *A* ***pairing function*** *of length n is a function* $f : \{1, \cdots, n\} \to \mathbb{N} \times \mathbb{N}$ *such that every number except 0 appears at most once as the first member of a pair and at most once as the second member of a pair. The* ***pivot*** *of* f *is the (unique) number* $k > 0$ *such that* $f(i) = (k, 1)$ *for some i.*

The definition of a pairing function ensure that the pivot is unique if it exists. For the number 1 occurs only once in a pair, so there is at most one k such that $(k, 1)$ is the value of f.

Let us be given argument structures α and β as in Definition 3.25. The pairing function $f(\alpha,\beta)$ is given as follows. It has length $m + n - p$, and

$$f(\alpha,\beta)(i) := \begin{cases} (i_k, j_k) & \text{if } g = i_k \text{ for some } k \leq p, \\ (i_k, 0) & \text{else if } g \leq m, \\ (0, j_k) & \text{else.} \end{cases} \tag{3.74}$$

As one can see the pairing function determines the sequence ξ. Indeed, this is main reason why we need it. The pairing function is now used to synchronise the semantics and morphology.

We define the following substitution s_f. We put $s_f(y) := x$ if for some i, $f(i) = (j, k)$, $y = u^2$, $x = v^1$, α_j contains v and β_k contains u. If no such i exists, we put $s_f(y) := y$. This completes the definition of s_f.

Finally, we turn to the morphology.

Definition 3.27 (Merge of Morphs) *Let* $\mathfrak{m} = (\mathfrak{g}, \mathcal{A}, \rho)$ *and* $\mathfrak{n} = (\mathfrak{h}, \mathcal{B}, \sigma)$ *be morphs of dimension* m *and* n*, respectively. In particular,* $\mathcal{A} = \langle \upsilon_1, \cdots, \upsilon_m \rangle$ *and* $\mathcal{B} = \langle \chi_1, \cdots, \chi_n \rangle$*. Let* f *be a pairing function. Then* $\mathfrak{m} \bullet_f \mathfrak{n} := (\mathfrak{i}, \mathcal{C}, \tau)$*, where* $\tau = \rho \cdot \sigma$*,*

$$\mathcal{C}(i) := \begin{cases} \upsilon_j & \text{if } f(i) = (j, 0); \\ \chi_j & \text{if } f(i) = (0, j); \\ \upsilon_j \cdot \chi_k & \text{if } f(i) = (j, k),\ j, k > 0. \end{cases} \tag{3.75}$$

And, finally, $\mathfrak{i} := \mathbb{H}(\mathfrak{g}, \mathfrak{h})$*. where* $\upsilon_i = (M, N, \mathbb{H})$*,* i *the pivot of* f*.*

So, in a polyadic merge or fusion, it is the pivot that decides in which way the glued strings are combined. For notice that by nature, polyadic merge identifies several variables at once. However, morphologically speaking only one combination of morphs (morphemes) takes place. Of course it could be any of the morphs. A decision must be made. And it is that the last AIS in α to combine with an AIS in β is to be taken.

We conclude the section with some definitions on equivalence. The idea behind equivalence is that from an empirical perspective there is often not much to choose between one analysis and another. The simplest case involes choice of variables, but later on we will meet other cases where it is impossible to choose between different analyses. To make this precise we need a definition of *congruence* of signs and sign systems.

Definition 3.28 (Congruence) *Two morphs* $m_1 = (\mathfrak{g}_1, \mathcal{A}_1, \rho_1)$ *and* $m_2 = (\mathfrak{g}_2, \mathcal{A}_2, \rho_2)$ *are **congruent**, in symbols* $m_1 \asymp m_2$*, if* $c(\mathfrak{g}_1) = c(\mathfrak{g}_2)$ *(identity in string content). Two morphemes* M_1 *and* M_2 *are **congruent**, in symbols* $M_1 \asymp M_2$*, if for every* $m \in M_1$ *there is an* $n \in M_2$ *such that* $m_1 \asymp n$*, and for every* $m_2 \in M_2$ *there is an* $n \in M_1$ *such that* $n \asymp m_2$*. Two signs* $\mathfrak{S}_1 = (m_1, \alpha_1, \Delta_1)$ *and* $\mathfrak{S}_2 = (m_2, \alpha_2, \Delta_2)$ *are **congruent**, in symbols* $\Sigma_1 \asymp \Sigma_2$*, if* $m_1 = m_2$ *(congruence) and* $\Delta_1 \equiv \Delta_2$ *(logical equivalence).*

Definition 3.29 (Congruent Sign Systems) *Let V and W be two sets of signs. V and W are called **congruent**, in symbols* $V \asymp W$*, if there is a bijection* $\zeta : V \to W$ *such that* $\mathfrak{S} \asymp \zeta(\mathfrak{S})$ *for all* $\mathfrak{S}$*, and such that* $\mathfrak{S}_1 \oplus \mathfrak{S}_2$ *is defined if and only if* $\zeta(\mathfrak{S}_1) \oplus \zeta(\mathfrak{S}_2)$ *is defined; and if one of the two is defined then*

$$\zeta(\mathfrak{S}_1 \oplus \mathfrak{S}_2) = \zeta(\mathfrak{S}_1) \oplus \zeta(\mathfrak{S}_2) \tag{3.76}$$

Congruent sign systems are the same not only for the signs that they involve in terms of the pairing between exponents and meanings; they also exhibit identical combinatorics in between them.

Notes. The pairing function is defined differently in the implementation. It uses the number -1 in place of 0, since counting starts at 0.

Exercise 21. Assume that an argument structure β does not contain unidentified variables and exports at least one variable. Show that it is saturated if and only if it is lexical. *Note.* This shows that if the merge $\alpha \bullet \beta$ succeeds and only one variable is identified, then β has length 1 (and the merge is monadic). Alternatively, it shows that 1-adic merge is monadic.

Exercise 22. Show that in general $\Sigma_1 \oplus (\Sigma_2 \oplus \Sigma_3) \neq (\Sigma_1 \oplus \Sigma_2) \oplus \Sigma_3$. This is because the operation may be undefind.

Exercise 23. A **partial semigroup** is a pair $(G, \cdot)$ such that G is a set and $\cdot$ a partial binary operation on G such that if $(a \cdot b) \cdot c$ and $a \cdot (b \cdot c)$ exist they are equal. Show that if S is a set of signs, then $(S, \oplus)$ is a weak semigroup.

Exercise 24. Let V be a sign system. Let W result from V by performing the following manipulation. If a given AIS contains $\langle x : \Diamond : A \mapsto B\rangle$ for some AVMs A and B, then choose a suitable variable y and replace that AIS by the sequence $\langle x : \vartriangle : B\rangle, \langle x : \triangledown : A\rangle$. Show that $V \asymp W$. *Remark.* Start with the case where A and B are names.

Exercise 25. (Continuing the previous exercise.) What happens if B is a copy AVM? Can it also be eliminated?

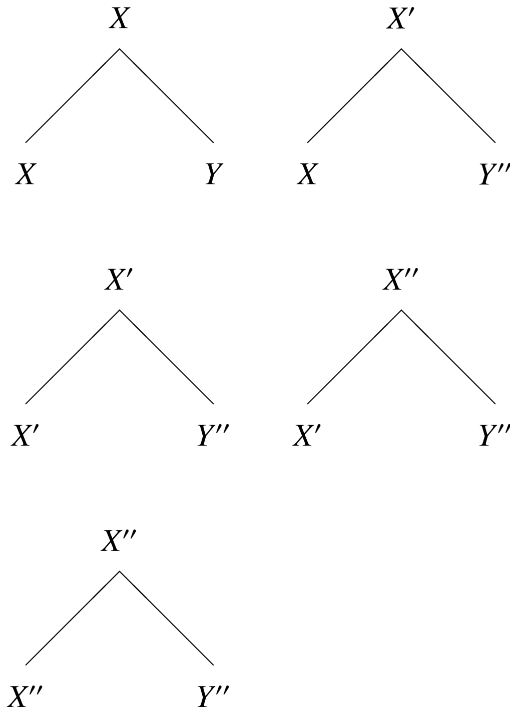
3.6 Basic Syntax

There is a simple four-fold classification of lexical argument structures given two basic objects, events and objects. In a lexical argument structure generally only one variable is exported, and this variable determines the semantic type of the result. Depending on whether this variable is also imported or not, the *category* of a syntactic object derives from its argument structure as follows.

	$\vartriangle$	$\Diamond$
event	V	Adv
thing	N	A

Let us explain this a little bit. Nouns export a referent denoting objects. There are nouns that import no other variable, such as /`man`/, whereas others do import one, such as /`destruction`/ or /`father`/. Nouns are prototypical arguments with respect to the referent that they export. Adjectives and adverbs are adjuncts with respect to the referent that they export. This explains why they can be accumulated in any number within a noun or verb phrase. They, too, may select arguments, such as /`proud`/. Verbs have an event referent, which is external, but may take a number of arguments.

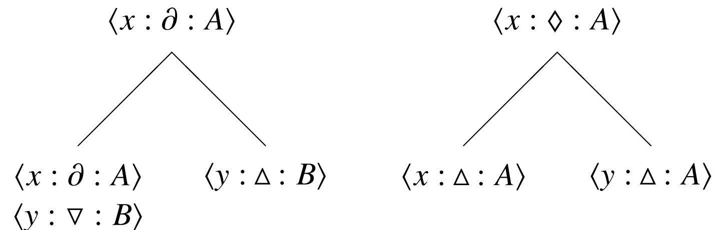
Figure 3.4: X-bar Syntax



Our classification looks different from the classical one. For example, there are no prepositions in this schema. The problem with classifying prepositions is however that PPs may be used to modify events as well as objects. Hence, PPs may function as adverbs as well as adjectives. What is more, PPs are often similar to case marked DPs, and the difference often seems to be rather a morphological accident than a syntactic reality. See Kracht 2003a for further arguments.

Now we turn to X-bar syntax. The basic property of X-bar syntax is that there is a head and it projects up to a phrase. In Government and Binding Theory, the types of syntactic junctures shown in Figure 3.4 are allowed (order irrelevant). Here, X and Y are variables over categories. The primes count the levels of projection. There are zero, one and two primes, hence up to three levels. The variable X denotes the head, since it is both the category of a daughter and of the mother.

Figure 3.5: Argument Discharge



(We may have $X = Y$.)

We have already seen how the constraints on lexical argument structure allow to define the category of a word or structure in the usual sense. Let us now see how the conditions on lexical merge allow to deduce the basic properties of X-bar syntax. First, an argument structure is phrasal if and only if it does not import any referent other than the one it exports. (This is not exactly saturatedness, since it includes adjuncts.) In X-bar terminology this means that it is of the third level, since two primes are now considered maximal, unlike in its original version of Jackendoff 1977. In our terminology, however, no levels are assigned to a phrase. Hence, there can be any number of levels in between a word and its corresponding phrase, although that number will rarely exceed three. For it directly corresponds to the number of arguments a word-level argument structure needs. If the highest number of arguments is three, our highest number of levels will be four. We have two basic types of merge: head-complement and head-adjunct. These are exemplified in Figure 3.5. To the left we have the combination head-complement. One referent, different from the one defining the category, is identified and "discharged" from the argument structure. The level increases, since there are less argument discharges needed to reach the phrasal level. To the right we have the head-adjunct juncture. The adjunct identifies the referent, but no change is made in the argument structure of the head. Notice that the referent that gets identified is not necessarily the head referent. One has to remember that our calculus allows for junctures that are not X-bar syntax proper. Therefore, no step is taken to ban such junctures. Finally, we have to discuss the head-head juncture in X-bar syntax. This is used differently in Government and Binding. Namely, this is not a phrasal combination but at word level, as we can see from the fact that the level is not increased. Moreover, it covers such cases as serial verbs, the verbal cluster

tense+aspect+verb and so on. Hence, it corresponds in our terminology not to the lexical merge but to the functional merge. Since the functional merge is rather involved we will not discuss it here.

We will exemplify the effect of these assumptions with Latin and German syntax.

Example 11. Recall the word order facts (3.62) – (3.67). In a subordinate clause the verb must be to the right of its arguments. Hence the semantic structure for the verb /geben/ 'give' is

(3.77)

/geben/○, ⦸, ⦸, ⦸

$\langle e : \triangle : - \rangle$,	$\langle x : \triangledown : \text{NOM} \rangle$,
$\langle z : \triangledown : \text{DAT} \rangle$.	$\langle y : \triangledown : \text{ACC} \rangle$,
e, x, y, z	
give′(e);	act′$(e) \doteq x$;
thm′$(e) \doteq z$;	ben′$(e) \doteq z$.

(3.78) SOIV, SIOV, OSIV, OISV, ISOV, ISOV

This is correct. Additionally, the order of the arguments is relevant. There is one word order, SIOV, which is obtained by E-merge only. All others require skipping arguments.

Moreover, assume that adverbs are preverbal in the subordinate clause, and that they allow for fusion. Then the theory predicts another fact, namely that adverbs (or adverbial phrases for that matter) may occur at any position between the arguments, that is, at any place marked by a star.

(3.79) $\star S \star O \star I \star V$

This is indeed the case (/D./ is short for /Direktor/).

(3.80) ..., dass morgen der Vater dem D. den Schlüssel gibt.
... that tomorrow the-NOM father the-DAT director the-ACC key gives

(3.81) ..., dass der Vater morgen dem D. den Schlüssel gibt.

(3.82) ..., dass der Vater dem D. morgen den Schlüssel gibt.

(3.83) ..., dass der Vater dem D. den Schlüssel morgen gibt.

For adverbs have the argument structure $[\langle e : \Diamond : \alpha\rangle, \ldots]$. Thus, basic word order facts can be accounted for, though only for the subordinate clause. ⚽

Example 12. In Latin, word order is even more free. This pertains foremost to the arguments of the verb. Consider as an example the following sentence.

(3.84) `Cicero consuli librum dat.`
'Cicero gives a/the book to the consul.'

This sentence is grammatical no matter in which of the 24 possible orders the words are being put. Assume the uninflected verb has the following argument structure.

(3.85)

/`dat`/○, ⧀, ⧀, ⧀

$\langle e : \triangle : -\rangle$,	$\langle x : \nabla : \text{NOM}\rangle$,
$\langle y : \nabla : \text{ACC}\rangle$,	$\langle z : \nabla : \text{DAT}\rangle$
e, x, y, z	
give′(e);	act′$(e) \doteq x$;
thm′$(e) \doteq y$;	ben′$(e) \doteq z$.

Let us assign the following structures to the remaining words:

(3.86)

/`Cicero`/○	/`librum`/○	/`consuli`/○
$\langle x : \triangle : \text{NOM}\rangle$	$\langle x : \triangle : \text{ACC}\rangle$	$\langle x : \triangle : \text{DAT}\rangle$
x	x	x
cicero′(x)	book′(x)	consul′(x)

Then 6 combinations out of 24 are acceptable and result in the following structure (modulo renaming of referents):

(3.87)

$\langle e : \triangle : -\rangle$	
$x, \quad y, \quad z, \quad e$	
give′(e);	act′$(e) \doteq x$;
thm′$(e) \doteq y$;	ben′$(e) \doteq z$;
cicero′(x);	book′(y);
consul′(z).	

One way to lift the restriction is to allow for each of the objects to either be on the left or on the right. This would require 8 different entries for the verb /`dat`/. ⚽

The basic structure of a subordinate clause is therefore accounted for without assuming movement of arguments. So, scrambling is not needed. We will discuss the implications of this later. The current proposal has its limitations, as is shown in the discussion of Latin. Here are two different ways to tackle the problem. The first is to assume that some morphological process is responsible to build up the argument structure. Initially, a verb exports (!) all its arguments, and diathesis and other processes lead to an installment of the arguments into the argument structure, where nominal arguments are no longer exported but actually imported. These processes require a limited set of morphemes, which can ask the argument to be realised to the right or to the left. In this way, when we want both options to be realised we do not need to duplicate the number of verbal roots; duplicating the argument installment morpheme is enough. More on that in Chapter 6.

The other option is to allow disjunction in the handlers. This however has the drawback that parse terms will not uniquely represent one exponent. For if we have a choice to put the argument either to the left or to the right under merge, every time we merge we must decide which option to take. This is the reason the second option has not been chosen.

Notes on this section. The system defined so far looks quite like categorial grammar. Yet there are noteworthy differences. First, we have defined liberal word orders not by using lexical rules (because the typing system generally is not flexible enough). But even if we were to consider only E-access, our system is different. Syntactically, the types that can be defined are more restricted. If order is disregarded, they are of the form

(3.88) $$\alpha_1 \multimap (\alpha_2 \multimap \ldots (\alpha_n \multimap \beta) \ldots)$$

where the α_i, $1 \leq i \leq n$, and β are basic types. In order for this to work, two things are necessary. First, the basic ontology must be rich enough to accommodate differences that are otherwise accounted for by higher types. Second, the semantics must be flexible enough to avoid the need for Geach's rule. For Geach's rule would be needed if one and the same adjunct can combine with different argument structures.

Exercise 26. For each argument of the verb /`dat`/ we can choose the directionality to be left or right. For each choice we make there are 6 word orders that are being accounted for. This would mean that we have 48 different word orders, while actually there are only 24. Resolve the mystery.

Exercise 27. The word order of the main clause in German still is unresolved. See Example 9 for the essential facts. Try to devise a solution.

Chapter 4

Features

While the previous chapters explained in some detail the mechanics of argument structures, we shall pause here and reflect on the relationship between the structure of the representations and linguistic analysis. This will give a better insight into why particular choices have been made.

4.1 Different Kinds of Features

Paradigms collect word forms of the same word. The various forms are said to possess different features. It is this idea of connecting forms with underlying features that is central to the argument structures.

Example 13. The nouns of Latin possess different forms, depending on number and case. Here are the forms of /`equus`/ 'horse'.

(4.1)

	singular	*plural*
nominative	`equus`	`equi`
genitive	`equi`	`equorum`
dative	`equo`	`equis`
accusative	`equum`	`equos`
ablative	`equo`	`equis`

The nominative singualar word form has the following structure.

$$\text{(4.2)} \qquad \begin{array}{c} \texttt{/equus/} \\ \begin{array}{|l|} \hline \langle x : \vartriangle : \left[\begin{array}{l} \textsc{num}: \ sg \\ \textsc{case}: nom \end{array}\right] \rangle \\ \hline x \\ \hline \mathsf{horse}'(x) \\ \hline \end{array} \end{array}$$

The feature space for Latin contains the feature NUM with values *sg* and *pl*, and the feature CASE with values *nom*, *gen*, *dat*, *acc* and *abl*. Thus, we expect ten different entries, for example

$$\text{(4.3)} \qquad \begin{array}{c} \texttt{/equorum/} \\ \begin{array}{|l|} \hline \langle x : \vartriangle : \left[\begin{array}{l} \textsc{num}: \ pl \\ \textsc{case}: gen \end{array}\right] \rangle \\ \hline x \\ \hline \mathsf{horse}'(x) \\ \hline \end{array} \end{array}$$

Not all forms are different, though. ⚽

An analytic question now comes up. In Latin, virtually all nouns ending in /us/ inflect like /equus/. Thus we can segment the word form into two parts, a *root* /equ/ and an *ending* /us/. Next we can think of root and ending as separate morphemes. If we do so, the paradigm is just the effect of combining the root with the various endings, and none of the word forms will have to be entered into the lexicon. We shall see shortly how that can be done. But first we need to clarify what we mean by "feature".

The notion of feature is surprisingly unclear. Corbette 2012 uses the term "feature" to refer to attributes. Often, the term feature refers to particular morphemes. Thus, Corbett speaks of a number feature where we speak of an attribute. But this is just a minor divergence. More irritating is the syntactic notion of a feature. In Minimalism, the feature-as-thing metaphor prevails. Features are intimately linked to the morph(eme)s that express them. In fact, features regularly get their own projection. Thus, one speaks of the number feature /s/, meaning that the feature "number" is expressed by /s/ (among other things), which is the head of the number phrase. This notion of feature focusses on the material side of things and is surprisingly nonabstract compared to GPSG, for example, which maintains the feature-as-property view. One may be forgiven to confuse the features in Minimalistm with morphemes. But there is after all a crucial difference in that not every morpheme qualifies as a feature.

We will *not* follow this usage, however. For us, a feature is an abstract entity. There are features related to morphological classes, and features related to syntactic classes. Both classes are instances of feature spaces, as defined in Definition 2.6. Nevertheless, an unclarity remains. In feature spaces, there is actually no feature as such. The feature "plural" corresponds to an attribute-value pair, say, [NUM : *pl*]. Again, one might think that the value *pl* of the attribute *is* the feature called "plural", but that is not a good way of seeing things. After all, a given value can be the value of many attributes, so the value alone does not provide enough information.

There also is another reason why values are not features. Consider the pair [NUM : $\star$] denotes the absence of a value. If no pair [NUM : α] is present then that counts as the same as [NUM : $\top$], wich in turn abbreviates a *set* of admissible value. This is underdeterminacy. However, [NUM : $\star$] means *by definition* that number has no value. Ideally, roots start out with features not present. Such roots are mostly not independent word forms. To become word forms, morphological processes must apply. These processes turn the root into a word form. In the present framework, a crucial aspect of this process is that it *installs the features*. Although we think of it as installing the feature, as if adding something, actually in the representation it is a transformation of the value of the attribute from $\star$ to whatever the associated value of the morpheme is.

Example 14. Let us continue the analysis of the Latin noun. We enter into the lexicon the following root.

(4.4)

/equ/
$\langle x : \vartriangle :$ [NUM : $\star$ / CASE : $\star$] $\rangle$
x
$\mathsf{horse}'(x)$

Case and number are now considered separate morphemes.

(4.5)

/us/$\otimes$
$\langle x : \Diamond :$ [NUM : $\star \mapsto sg$ / CASE : $\star \mapsto nom$] $\rangle$
x

Notice that there is no separate morpheme for singular and for nominative. /us/ is a single morph, signaling both singular and nominative.

Example 15. We assume that noun roots in English are unspecified for number.

(4.6)

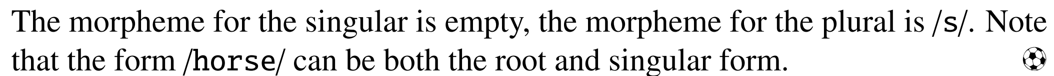
/horse/
$\langle x : \triangle : [\text{NUM} : \star] \rangle$
x
$\mathsf{horse}'(x)$

We assume two morphemes, one for each number.

(4.7)

//⊗	/s/⊗
$\langle x : \lozenge : [\text{NUM} : \star \mapsto sg] \rangle$	$\langle x : \lozenge : [\text{NUM} : \star \mapsto pl] \rangle$
x	x

The morpheme for the singular is empty, the morpheme for the plural is /s/. Note that the form /horse/ can be both the root and singular form.



When installing a feature we also need to pay attention to where the feature is added. An argument structure possesses several AISs, and each AIS may in turn have one or two AVMs. Thus, what is normally considered a simple agreement category like number and gender becomes ambiguous in view of double agreement. For example, many languages distinguish a subject agreement form from an object agreement form. Thus, while morphologically we speak of a feature like OBJ:pl, what happens in the argument structure is entirely different: the feature [NUM : *pl*] is being installed at the variable for the object rather than the subject.

Example 16. In Mordvin, a Uralic language, the verb shows agreement with the subject as well as the transitive object. Agreement is both in number and person. Moreover, there are different forms for the intransitive verb and its subject, as well as the transitive verb and its subject (called actor) and object (called undergoer). Table 4.1 shows the paradigm of a transitive verb, /sodams/ 'to know'. Table 4.2 shows the forms for an intransitive verb (the same verb again). The verb /sodams/ can be used transitively in the sense of knowing something and of knowing someone (see Keresztes 1990). Notice that certain forms are missing in this paradigm.

Table 4.1: Mordvin Double Agreement: sodams 'to know'

ν_A, π_A ↓	$\nu_U = sg, \pi_U$ → 1.SG	2.SG	3.SG
1.SG	–	sodatan	sodasa
2.SG	sodamasak	–	sodasak
3.SG	sodasamam	sodatanzat	sodasi(zé)
1.PL	–	sodatadiź	sodasińek
2.PL	sodasamiź	–	sodasink
3.PL	sodasamiź	sodatadiź	sodasiź
ν_A, π_A ↓	$\nu_U = pl, \pi_U$ → 1.PL	2.PL	3.PL
1.SG	–	sodatadiź	sodasiń
2.SG	sodamasiź	–	sodasit'
3.SG	sodasamiź	sodatadiź	sodasińźe
1.PL	–	sodatasiź	sodasińek
2.PL	sodasamiź	–	sodasink
3.PL	sodasamiź	sodatadiź	sodasiź

Table 4.2: Mordvin Single Agreement

NUM	PER	
SG	1	sodan
	2	sodas
	3	sodi
PL	1	sodatano
	2	sodatado
	3	sodit'

These correspond to the reflexive use of the verb, when part of the subject is also part of the object.

For the transitive verb we start with the following representation.

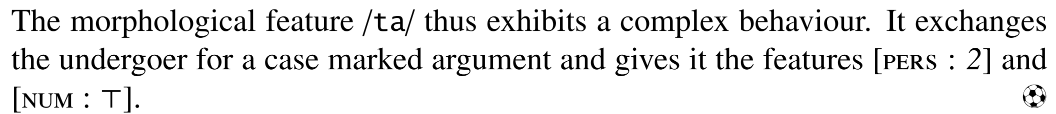
(4.8)

$$\begin{array}{|l|}\multicolumn{1}{c}{\texttt{/soda/}}\\\hline \langle e : \vartriangle : [\textsc{trans} : \mathit{true}]\rangle, \\ \langle x : \vartriangle : [\textsc{gf} : A]\rangle, \\ \langle y : \vartriangle : [\textsc{gf} : U]\rangle. \\\hline e, x, y \\\hline \mathsf{know}'(e); \quad \mathsf{exp}'(e) \doteq x; \\ \mathsf{thm}'(e) \doteq y. \\\hline \end{array}$$

The feature installment can be done in as many steps as the morphology allows for. Here is one possibility, the simultaneous installment of person and number for the undergoer.

(4.9)

$$\begin{array}{|l|}\multicolumn{1}{c}{\texttt{/ta/}}\\\hline \langle e : \Diamond : [\textsc{trans} : \mathit{true}]\rangle, \\ \langle x : \triangledown : \begin{bmatrix} \textsc{case}: \mathit{acc} \\ \textsc{pers}: \ 2 \\ \textsc{num}: \ \top \end{bmatrix}\rangle, \\ \langle y : \triangledown : [\textsc{gf} : U]\rangle. \\\hline e, x, y \\\hline x \doteq y. \\\hline \end{array}$$



The morphological feature /ta/ thus exhibits a complex behaviour. It exchanges the undergoer for a case marked argument and gives it the features [PERS : 2] and [NUM : ⊤]. ⚽

Features can be installed, modified, and removed. Features come into existence because a variable is created: in the case above, the undergoer is exchanged for an accusative object, accompanied by the introduction of a new variable, which is tied to an AIS. In that AIS, person and number are marked as absent. The agreement suffix is charged with installing the values. It should be stressed that there are two kinds of features: morphological feature and those occurring in an AIS. However, a morphological feature is linked — though indirectly — to a position in an AIS and hence ultimately depends on the presence of a variable. Thus, different forms are ultimately differentiated only because in a merge, functor and argument share a variable, and through this sharing determine each other's forms.

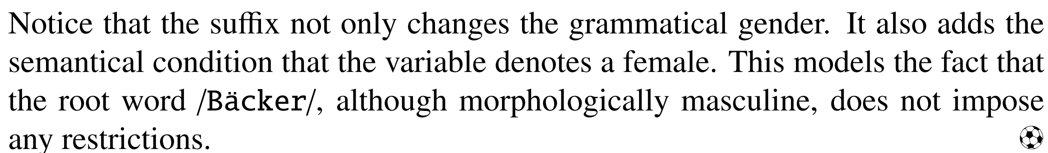
Values can also be changed. Here is an example.

Example 17. Nouns in German are marked for gender, like /`Auto`/ ‘car’ (neuter), /`Waage`/ ‘scale’ (feminine) and /`Baum`/ ‘tree’ (masculine). This gender is not inserted by means of a morphological process, nor is it changed by any such process. However, an exception exists for professions and properties of people in general. For example, there is the word /`Bäcker`/ ‘baker’, which has masculine gender. Adding the suffix /`in`/ creates a word of feminine gender: /`Bäckerin`/ ‘baker (woman)’. Similarly, /`Fahrer`/ vs. /`Fahrerin`/, /`Schüler`/ vs. /`Schülerin`/. Here we start with the following sign.

$$(4.10)\qquad \begin{array}{c} /\texttt{Bäcker}/\bigcirc \\ \hline \langle x : \vartriangle : \begin{bmatrix} \text{CASE} : \star \\ \text{GEN} : m \\ \text{NUM} : \star \end{bmatrix} \rangle \\ \hline x \\ \hline \mathsf{baker}'(x) \end{array}$$

The suffix has the following sign.

$$(4.11)\qquad \begin{array}{c} /\texttt{in}/\oslash \\ \hline \langle x : \Diamond : \begin{bmatrix} \text{CASE} : & \star \\ \text{GEN} : m \mapsto & f \\ \text{NUM} : & \star \end{bmatrix} \rangle \\ \hline x \\ \hline \mathsf{female}'(x) \end{array}$$



Notice that the suffix not only changes the grammatical gender. It also adds the semantical condition that the variable denotes a female. This models the fact that the root word /`Bäcker`/, although morphologically masculine, does not impose any restrictions. ⚽

It is technically also possible to remove features. Usually, this happens automatically when a variable disappears with its AIS. Technically, however, one may change from a certain value to no value at all, thus reversing the feature installment.

Agreement gets a simple explanation. Variables must be identified under merge. This can only happen when the corresponding names match. For that, both

functor and argument must have the appropriate form. In German, the determiner has different forms depending on gender, number and case. For example, we have /`das Auto`/, /`die Waage`/ and /`der Baum`/ (see Example 17). Consequently, we have /`der Bäcker`/ and /`die Bäcker`/. Agreement in form is a consequence of the requirement of matching names in merge.

The present framework thus connects two different mechanisms in language. One is the mechanism of argument selection, typically used in categorial grammar. A head is characterised by its potential to take certain arguments. We speak of the head selecting its arguments. If H selects for some argument C, then the complex formed by H and C together is a constituent. The selection requirement is cancelled. The other mechanism is that of agreement. The mechanism of agreement determines that H and C can be put together into a constituent if they agree in a certain way. The most elaborate theory based on agreement is HPSG. Here, any constituent simply puts constraints on its environment. H may thus declare that it is immediately to the left of some string which has a variable of a given type. C on the other hand may declare that it contains a variable of that type. If the constraints of H and C are consistent when attempting to form a constituent then that constituent is licit.

In the present mechanism, heads declare certain selectional restrictions. They do so by means of putting a variable into the argument structure together with a declaration of its name(s). The conditions on merge guarantee that the head can form a constituent with some constituent C only if the names of that variable match those of the principal variable exported by C. The match is a purely formal one, though some names may reflect semantic properties of the argument.

Notes on this section. Morphemes may install several features at once. Moreover, the morphology-syntax boundary is rather fluid. By contrast, many frameworks, for example LFG and HPSG, place great value on the so-called *Lexical Integrity Principle*, which states that morphological processes must precede syntactic processes. While at first this sounds like a good way to cut the cake, it should be noticed that the distinction between feature installment via morphology and feature installment via syntax is rather unpredictable. In the present system, morphology and syntax work hand in hand.

Exercise 28. Let us look again at Mordvin (Example 16). The sign for the verbal root is shown in (4.8). One peculiarity is that it exports all of its referents.

Alternatively, we could have started with the following root.

(4.12)

$$\texttt{/soda/}$$

$$\begin{array}{|l|}\hline \langle e : \triangle : [\textsc{trans} : \mathit{true}]\rangle, \\ \langle x : \triangledown : \begin{bmatrix} \textsc{case}{:}\mathit{acc} \\ \textsc{pers}{:}\ \star \\ \textsc{num}{:}\ \star \end{bmatrix} \rangle, \\ \langle y : \triangledown : \begin{bmatrix} \textsc{case}{:}\mathit{nom} \\ \textsc{pers}{:}\ \star \\ \textsc{num}{:}\ \star \end{bmatrix} \rangle. \\ \hline \end{array}$$

$$\begin{array}{|l|}\hline e, x, y \\ \hline \mathsf{know}'(e); \quad \mathsf{exp}'(e) \doteq x; \\ \mathsf{thm}'(e) \doteq y. \\ \hline \end{array}$$

Show that this is viable. What are the downsides of this?

Exercise 29. Latin nouns inflect differently depending on morphological class. Above in 13 we have seen the forms of o-class nouns. The following shows the corresponding forms for the a-class noun /`agricola`/ 'farmer'.

(4.13)

	singular	*plural*
nominative	`agricola`	`agricolae`
genitive	`agricolae`	`agricolarum`
dative	`agricolae`	`agricolis`
accusative	`agricolam`	`agricolas`
ablative	`agricola`	`agricolis`

Modify the entries for the nouns and endings so that there is a single morpheme for, say, genitive singular, with different morphs for the various classes.

4.2 Syncretism

The mapping between overt forms and underlying representations is far from ideal. The Latin noun /`equus`/ has 7 different forms, while there are 10 combinations of number and case.

Example 18. German has four cases, *nominative*, *genitive*, *dative* and accusative; three genders, *masculine*, *feminine* and *neuter*; and two numbers, *singular* and *plural*. This yields an array of 24 different AVMs (= names). Nouns, determiners and adjectives however possess only a fraction of these different forms. Here are the forms of the noun /Haus/ 'house'.

(4.14)

	singular	*plural*
nominative	Haus	Häuser
genitive	Hauses	Häuser
dative	Haus	Häusern
accusative	Haus	Häuser

And here are forms of the determiner /der/ 'the'.

(4.15)

	masc		*fem*		*neut*	
	sing	*plur*	*sign*	*plur*	*sign*	*plur*
nominative	der	die	die	die	das	die
genitive	des	der	der	der	des	der
dative	dem	den	der	den	dem	den
accusative	den	die	die	die	das	die

The determiner has only 6 different forms. ⚽

One may take advantage of this situation in various ways. For example, in the plural the determine does not have different forms for different genders. Also, in the plural (and the neuter) accusative and nominative are the same. We can thus reduce the entries. Rather than having 24 different entries for the determiner, we can do with less on condition of underspecification. We say that a paradigm is *syncretic* or *shows syncretism* if there are two names for which the paradigm contains the same form.

Analytically, we may even propose underlying features for which no overt reflex can be found. Such is the case for gender in Uralic languages. Hungarian morphology does not show gender. It is of course technically possible to propose an attribute *gen* and certain values, though it is not clear which ones they are (two, as in French, 3 as in Latin, or 10?). Morphological analysis will most likely yield that this feature has only one value, and that that value is always present. We may therefore propose the following princple.

[Minimise Syncretism]

> The morphological analysis must be done in such a way as to minimise the amount of syncretism.

We shall leave open exactly how this is evaluated.

Instead we shall look at particular cases where syncretism leads to problems that language needs to solve. The formal calculus shows us how particular structures are paired with particular meanings. The operations are acting on form and meaning at the same time. However, in actual communication, we are given only one of them, form or meaning, and want to obtain the other. We translate from meaning to form when we are the speaker and from form to meaning when we are the hearer. The translation is mostly not unique; the same meaning can be expressed in different ways, and the same expression can mean different things. The latter is a consequence of various confounding factors; the cause can be the structural as well as lexical. The latter means that certain exponents are ambiguous. Of particular importance is the situation where morphological marking is insufficient. This is quite a typical situation in language. One part of it is syncretism. Syncretism fails to give a one-to-one-mapping between word forms and names. The ambiguity can have a different effect depending on which way we go, from form to meaning or from meaning to form. We shall address a particular problem that arises primarily in connection with G-access.

Consider a language that uses the G-access. In such a language, a verb takes its arguments in any order, since it is allowed to take the last *matching element* rather than the last element simpliciter. Hence the verb takes its arguments in any order. This is borne out for German.

(4.16) `..., dass der Kater den Vater sieht.`
..., that the tomcat the father sees.

(4.17) `..., dass den Vater der Kater sieht.`
..., that the father the tomcat sees.
'..., that the tomcat sees the father.'

(4.18) `..., dass den Kater der Vater sieht.`
..., that the tomcat the father sees.

(4.19) `..., dass der Vater den Kater sieht.`
..., that the father the tomcat sees.
'..., that the father sees the tomcat.'

Here, (4.16) and (4.17) both mean the same, and (4.18) and (4.19) also mean the same, as shown in the translation. This is precisely as we have seen above in the case of Latin.

Additionally, German nouns also show a lot of case syncretism. For example there is no distinction between nominative and accusative except for masculine nouns in the singular. The distinction shows up in the determiner, however, not in the noun itself. The noun /`Kater`/ could also be singular or plural. The burden of disambiguation is entirely on the determiner. In the masculine singular the determiner shows clearly and unambiguously case and number.

We shall now look at cases when the determiner actually fails to exhibit the required contrast. The accusative and the nominative case forms are identical for all feminine and neuter noun phrases as well as all plural noun phrases. For example, /`die Katze`/ may be either nominative or accusative. (Note that the plural is /`die Katzen`/, so here it is the noun that exhibits the singular / plural contrast.) The sentence (4.20) is ambigous between two readings, which correspond in English to (4.21) and (4.22).

(4.20) ..., `dass die Katze die Mutter sieht.`

(4.21) '..., that the cat sees the mother.'

(4.22) '..., that the mother sees the cat.'

Let us see how we can account for this possibility.

Let A be the speaker of (4.20). A knows whether he wants to convey (4.21) or (4.22). He will use the proper case labels in his calculation. If A intends that /`die Katze`/ is the subject, he will give it the case name NOM, and if he assumes that /`die Katze`/ is the object, he will give it the case name ACC. Similarly with /`die Mutter`/. So, for A the situation is as follows for (4.21).

(4.23)

`die Katze`	`die Mutter`	`sieht`		
$\langle x \rangle$	$\langle y \rangle$	$\langle e,$	$v_1,$	$v_2 \rangle$
⋮	⋮		⋮	⋮
NOM	ACC		NOM	ACC

And for (4.22) it is like this:

$$\begin{array}{lccccc} & \texttt{die Katze} & \texttt{die Mutter} & & \texttt{sieht} & \\ & \langle x_1 \rangle & \langle x_2 \rangle & \langle e, & y_1, & y_2 \rangle \\ (4.24) & \vdots & \vdots & & \vdots & \vdots \\ & \textsc{acc} & \textsc{nom} & & \textsc{nom} & \textsc{acc} \end{array}$$

In both cases, the merge is well-defined with G-access and yields the desired translations.

For the hearer B the situation is different. B does not know which case labels to stick in. There are now two distinct ways in which B can handle the situation. The first choice is to interpret /`die Katze`/ as the exponent of two (homonymous) phrases, one in the nominative and the other in the accusative. The other is to interpret it as the exponent of a single underspecified structure. These options make different predictions concerning grammatical acceptability which I shall now turn to. Consider the first option. Since the calculus is blind to the actual form of the exponents this option predicts that all word orders are grammatical, as if all case endings were maximally distinct.

The second option is different. Under this option B will assume that (4.20) means what (4.21) means. Let us see why this is so. The elements involved are drawn from the lexicon (plus morphology, but see Chapter 5) by looking at their overt form—since this is what we are given. All noun phrases (/`die Katze`/, /`die Mutter`/) are ambiguous between nominative and accusative singular. B will therefore represent them as follows:

$$\begin{array}{lccccc} & \texttt{die Katze} & \texttt{die Mutter} & & \texttt{sieht} & \\ & \langle x_1 \rangle & \langle x_2 \rangle & \langle e, & y_1, & y_2 \rangle \\ (4.25) & \vdots & \vdots & & \vdots & \vdots \\ & \textsc{nom} \sqcup \textsc{acc} & \textsc{nom} \sqcup \textsc{acc} & & \textsc{nom} & \textsc{acc} \end{array}$$

Here, the merge yields only (4.21) as a result. Let us look at this a little bit closer. The structures to be inserted for the words are underspecified. For example, we insert the following structures for the words:

(4.26)

/`Katze`/○
$\langle x : \triangle : \begin{bmatrix} \textsc{num} : sg \\ \textsc{case} : \{nom, acc, dat\} \end{bmatrix} \rangle$
x
cat′(x)

/`Mutter`/○
$\langle x : \triangle : \begin{bmatrix} \textsc{num} : sg \\ \textsc{case} : \{nom, acc, dat\} \end{bmatrix} \rangle$
x
mother′(x)

Here, we ignore gender for simplicity. (All occurring items are feminine.) The determiner /`die`/ is also in many ways ambiguous:

$$\text{(4.27)}\qquad \begin{array}{l} \quad\texttt{/die/}⧁ \\ \hline \multicolumn{1}{|l|}{\langle x : \Diamond : \left[\begin{array}{l} \textsc{num}:\{sg, pl\} \\ \textsc{case}:\{nom, acc\} \end{array}\right] \rangle} \\ \hline \multicolumn{1}{|l|}{x} \\ \hline \multicolumn{1}{|l|}{\mathsf{unique}'(x)} \\ \hline \end{array}$$

(The semantics is still quite sketchy. We shall return to this issue.) If we merge the structures for /`die`/ and /`Katze`/ and /`die`/ and /`Mutter`/, respectively, we obtain for the first pair

$$\text{(4.28)}\qquad \begin{array}{l} \quad\texttt{/die Katze/}\bigcirc \\ \hline \multicolumn{1}{|l|}{\langle x : \triangle : \left[\begin{array}{l} \textsc{num}:sg \\ \textsc{case}:\{nom, acc\} \end{array}\right] \rangle} \\ \hline \multicolumn{1}{|l|}{x} \\ \hline \multicolumn{1}{|l|}{\mathsf{cat}'(x); \mathsf{unique}'(x).} \\ \hline \end{array}$$

Notice how the choices get reduced under merge. The noun signals that the complex is singular while the determiner eliminates the option of dative case. Now, if we merge /`die Mutter`/ and /`sieht`/, /`die Mutter`/ could in principle be either the subject or the object. However, in this merge it must inevitably be the object, since going from right to left in the argument structure, it matches the rightmost entry first, which corresponds to the object. So, it will end up being the object. After that, /`die Katze`/ merges, but the object argument has been cancelled, so it becomes the subject instead.

$$\text{(4.29)}\qquad \begin{array}{l} \quad\texttt{/sieht/}\bigcirc, ⧀, ⧀ \\ \hline \multicolumn{1}{|l|}{\langle e : \triangle : - \rangle,} \\ \multicolumn{1}{|l|}{\langle x : \triangledown : \left[\begin{array}{l} \textsc{num}:sg \\ \textsc{case}:nom \end{array}\right] \rangle} \\ \multicolumn{1}{|l|}{\langle y : \triangledown : [\textsc{case} : acc] \rangle.} \\ \hline \multicolumn{1}{|l|}{e, x, y} \\ \hline \multicolumn{1}{|l|}{\mathsf{see}'(e); \mathsf{act}'(e) \doteq x;} \\ \multicolumn{1}{|l|}{\mathsf{thm}'(e) \doteq y.} \\ \hline \end{array}$$

This means that although both word orders, (4.20) and (4.30), are allowed in German, (4.20) and (4.30) do not mean the same. In (4.30), the subject is /`die`

`Mutter`/ and /`die Katze`/ is object.

(4.30) `..., dass die Katze die Mutter sieht.`
..., that the cat the mother sees.

(4.31) `..., dass die Mutter die Katze sieht.`
..., that the mother the cat sees.

This suggests that hearers do not follow the second option. We can take this as an indication that the underspecification is not found in the AVM itself; rather we have to posit different lexical entries, each with a specific AVM.

Matters are different still in main clauses. Here the directionality is different for subjects and objects. Since both word orders are licit, however, we must assume that the verb always has two morphs associated with it, one for SVO and another for OVS word orders. (Actually, this is only the case if we work with concatenation only. A full account of German word order using discontinuity is however quite tricky and beyond the scope of this section.) This means in turn that syncretism between nominative and accusative never reduces the choices. For each verb now comes as two morphs. The sentence (4.32) now has two meanings while (4.33) does not.

(4.32) `Die Katze sieht die Mutter.`
'The cat sees the mother.'
'The mother sees the cat.'

(4.33) `Den Kater sieht die Mutter.`
*'The tomcat sees the mother.'
'The mother sees the tomcat.'

This lends support to the claim in Mel'cuk 1993 – 2000 that case is global. That is to say, even if the phrase /`die Mutter`/ could be an object to the verb in (4.33) that does not mean it has to. The fact that /`den Kater`/ is accusative means that its only option is to be object. Unlike English, in German we may have OVS word order regardless of syncretism. The associations between NPs and argument status are computed in this case globally.

We may approach word order variation in the subordinate clause similarly, simply awarding every transitive verb two morphs. Again, syncretism then increases the number of global options. This phenomenon is treated in the same

way by creating different lexical entries rather than leaving the indeterminacy in the AVM.

However, we briefly note that matters are more complex. Not all word orders are equally good or likely. Within the present system the differences between the word orders cannot be brought to light. An example has been noted in Müller 1999:

(4.34) `Maria mischt Wasser Wein bei.`
'Maria mixes wine into water.'

(4.35) `Maria mischt Wein Wasser bei.`
'Maria mixes water into wine.'

The NPs /`Wasser`/ and /`Wein`/ do not show the case distinction between dative and accusative. These two sentences indeed are not synonymous, as the glosses indicate. If we add the determiners, however, this effect disappears:

(4.36) `Maria mischt das Wasser dem Wein bei.`

(4.37) `Maria mischt dem Wein das Wasser bei.`
'Maria mixes the water into the wine.'

(4.38) `Maria mischt den Wein dem Wasser bei.`

(4.39) `Maria mischt dem Wasser den Wein bei.`
'Maria mixes the water into the wine.'

If entries are multiplied as discussed above, we have different entries for accusative, dative and nominative case markers, which all happen to be zero. If that were the case, (4.34) could have the same meaning as (4.35). The only difference between them would be the fact that one reading is obtained by skipping an argument while the other would be using plain E-access. All this points to the conclusion that there seem to be different "penalties" for the various operations involved for the parser. But this is not mirrored in the present system. It cannot differentiate easy from difficult readings.

Let us summarize the possibilities that we have with argument structures. First, languages can use E-access or G-access. Suppose a language uses E-access. Then this language is fully structural. The argument structure of the head is projected uniquely into the syntax. The only parameters left are the word order parameters. If we set them uniformly right or left, we get SOV, OSV, VOS and

VSO languages. Notice however that in VSO and OSV languages, the verb forms a constituent with its subject. (This argument requires continuity in constituent structure. The entire discussion makes this background assumption. I have not looked into the further possibility of allowing discontinuity. It will certainly help in eliminating some of the problems, but in turn needs arguing for in each particular case.) Although it goes against many currently accepted analyses, it has been claimed for languages like Berber and Toba-Batak by Keenan 1988 that these languages are VSO and that VS is a constituent. For OVS and SVO we simply let the verb pick different directionality for subject and object. However, we may also leave the directionality unspecified either partially or with both arguments. This generates a few patterns that are to our knowledge not attested (for example, if the subject is on either side but the object to the right, this language will allow for SVO/VOS patterns). However, just in case both subject and object are not directionally fixed, we get a language that specifies only immediate dominance but not linear precedence. Staal has claimed exactly this for Sanskrit word order patterns, see Gillon 1996. This means that if the verb forms a constituent only with its object (as is generally assumed) we get the following alternative word orders:

(4.40) [S [V O]], [S [O V]], [[V O] S], [[O V] S]

If G-access is allowed, we get languages that differ from the previous languages only in that they allow for scrambling. If the verb is at the right periphery, that is, all arguments are to the left, then we get German type subordinate clause structure. With all arguments needed to the right we get the mirror image of German. If directionality is unspecified, we get Latin, as discussed above. It is worthwile pointing out that the present model, although allowing free (or freer) word order, nevertheless has a notion of canonical word order. This is so since the argument structure of the verb is a sequence, not a set. And these languages allow for alternate word orders independently of syncretism (under the first option). Case syncretism increases pressure to conform to default word order but it is difficult to state in exact terms how stringent this requirement is. For example notice the fact that since in German the nominative and accusative are morphologically distinct only in the masculine singular, we find that there is a general bias against OSV constructions. Yet, number agreement may fill the gap. In the following sentence the singular agreement /`hat`/ in combination with the fact that /`die größten Kritiker`/ is definitely plural makes it clear right away that the subject is yet to come, and that /`die größten Kritiker`/ is actually the object.

(4.41) `Die größten Kritiker hat der Papst zu Hause.`

the biggest critics has the pope at home
'The pope's biggest critics are in his home country.'

Furthermore, in languages with rich morphology the word order freedom tends to be used to encode other features, in particular topic and focus. If we include prosodically marked discourse relations into the feature system of the DPs, then we can account for the fact that in German scrambled elements must be marked for discourse relations. Furthermore, we may restrict freedom of access in such a way that it is sensitive to certain features and not others. We shall not explore this further, however.

Notes on this section. Word order can be freed up even more if we allow fusion. Then we get languages that look more like Australian languages. Dixon 2002 remarks on Page 78: *In no Australian language is the syntactic function shown by the order of phrasal constituents within a clause (what is often, but misleadingly, called 'word order').* It is said, though, that many Australian languages do not even have a DP constituent. However, fusion is quite powerful and one must carefully look into the facts here.

We also note that if full fusion is allowed in syntax then even under the assumption of G-access we do not get fully free word order. Here is an example, which is reminiscent of the Golden Line in Latin verse. Suppose that a head H looks for two complements, C_1 and C_2, and that each complement is again looking for a complement. That is to say, C_1 has D_1 as complement and C_2 has D_2 as its complement. Then $HC_2C_1D_1D_2$ is accepted, while $C_1D_2HD_1C_2$ is not. The reason is that H cannot combine with either D_1 or D_2 even under fusion. But neither can C_1 combine with D_2 or C_2 with D_1. (This argument must be carefully constructed. If H selects its arguments under the name α and β, while C_1 selects its complement under γ and C_2 selects D_2 under δ and if α, β, γ and δ are pairwise distinct, then no parse exists. Otherwise, if for example $\gamma = \delta$, then C_1D_2 and C_2D_1 are constituents. What we are saying in this case is that $HC_2C_1D_1D_2$ cannot be parsed to mean the same as $(H(C_1D_1))(C_2D_2)$.) This means that the theory remains restrictive with respect to word order. Whether or not these word orders turn out to be the right ones remains to be seen.

Exercise 30. Using discontinuity show how to define a single entry for the German verb in the subordinate clause so that both SOV and OSV are admissible word orders. How would this solution fare with the examples (4.34) and (4.35)?

Exercise 31. Construct a solution of the last problem (Golden Line) using discontinuous constituents.

Exercise 32. Take the signs in (4.26) and (4.27) and eliminate the disjunction in the AVMs. Verify the claims made above concerning the readings so obtained.

Exercise 33. An opposition between two elements is *equipollent* if it reflects two different values of an attribute, for example singular:plural. An opposition is called *privative* if it reflects a contrast between the presence or absence of a feature. Explain how an attribute with only one admissible value may be nonredundant, and what opposition it gives rise to. Discuss which kind of opposition is manifested by the opoosition between singular and plural in Hungarian, and by case marking in Hindi. Hindi has two cases, **direct** (glossed DCT) and oblique (glossed OBL).

(4.42)

	singular	plural
DCT	`kamrā`	`kamre`
OBL	`kamre`	`kamroṁ`

4.3 Agreement

By design, the present theory also intends to cover agreement morphology. The leading idea is that if a variable is shared in merge, this triggers agreement between functor and argument.

Example 19. The Latin adjective agrees in number, gender and case with the head noun. The noun /`poeta`/ 'poet' is morphologically from the a-class (which

contains mostly feminine nouns) but controls masculine agreement.

(4.43)

	'laureate poet'	
nom sg	`poeta`	`laureatus`
dat sg	`poetae`	`laureato`
acc sg	`poetam`	`laureatum`
nom pl	`poetae`	`laureati`
dat pl	`poetis`	`laureatis`
acc pl	`poetas`	`laureatos`

Consider the following sign for /`poetam`/.

(4.44)

/`poetam`/

$\langle x : \vartriangle : \begin{bmatrix} \textsc{num}: & sg \\ \textsc{case}: & acc \\ \textsc{gen} & :masc \end{bmatrix} \rangle$

x
poet$'(x)$

Here are two forms of the adjective.

(4.45)

/`laureatus`/$\oslash$

$\langle x : \lozenge : \begin{bmatrix} \textsc{num}: & sg \\ \textsc{case}: & nom \\ \textsc{gen} & :masc \end{bmatrix} \rangle$

x
laureate$'(x)$

/`laureatum`/$\oslash$

$\langle x : \lozenge : \begin{bmatrix} \textsc{num}: & sg \\ \textsc{case}: & acc \\ \textsc{gen} & :masc \end{bmatrix} \rangle$

x
laureate$'(x)$

Merge of /`poetam`/ with /`laureatum`/ is successful since the AVMs are the same. However, /`poetam laureatus`/ would be ungrammatical, since /`laureatus`/ is in the nominative. ⚽

From the perspective of the theory, Latin presents an ideal case. However, it depends very much on the analysis whether or not agreement is to be expected.

Example 20. Hungarian nouns inflect for number and case. Adjectives do not.

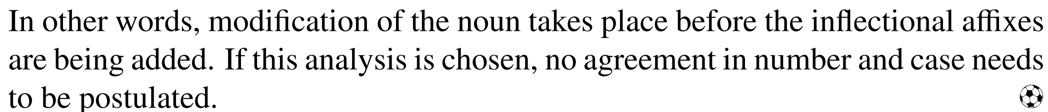
(4.46)

	‘the big mouse’
nom sg	`nagy egér`
dat sg	`nagy egérnek`
all sg	`nagy egérhez`
nom pl	`nagy egerek`
dat pl	`nagy egereknek`
all pl	`nagy egerekhez`

The adjective has an argument; we expect it to agree with the head noun. One possibility is to assume total syncretism in the adjective. In other words, the structure is the same as in Latin, but the adjective basically has only one form. However, there is a radically different analysis of this structure, one that makes the number and case affixes phrasal affixes.

(4.47) ((`nagy eger`)`ek`)`hez`

In other words, modification of the noun takes place before the inflectional affixes are being added. If this analysis is chosen, no agreement in number and case needs to be postulated.



Thus, how many different agreeing forms we have to expect depends not only on syncretism but also on the underlying structure. The structure determines the space of values. Obviously, there cannot be more forms as there are underlying values. The idea of total syncretism is consistent but not very attractive for Hungarian.

Comparing the analysis of Latin with that of Hungarian we notice that in principle both solutions work for both languages. For example, we may try the Hungarian analysis on Latin NPs. We merge the adjectival root with the noun and add the agreement suffix later.

(4.48) (`poet laureat`)`us`

The reason why this constituent structure is illicit is because the adjectival root has no gender, while the noun root does. As soon as we admit a masculine-morpheme for the adjective this changes the situation. Also, for two adjectives in a row we can successfully claim such a structure even under the current analysis.

(4.49) (`poet a`) (`laureat famos`)`us`

(Whether or not we have to put /et/ ‘and’ in between the adjectives does not change much.) The problem is that while agreement suffixes *can* be added, there is often no reason why they *have* to be added. One reason is that the lexicon does not inform us about words. We have argued against lexical integrity; however, we are now left without the notion of a word. Dually, Hungarian adjectives need not be inflected, but at this point there is no reason why they *cannot* be inflected.

There are several ways to regain it. Basically, they boil down to this. The adjective /laureat/ requires of its argument that it is a morphological word.

The lexical entries of the agreement suffixes have been left empty. But surely there are forms that have a particular meaning; one striking example is the plural. Generally, an NP in the plural means something like “several of that kind”. Now consider by way of example the following German NP:

(4.50)

den	schwarzen	Katern
the-DAT.PL	black-DAT.PL	tomcat-DAT.PL

The morphological paradigms of determiners, adjectives and nouns in German are quite distinct, as we have seen. In the singular, the noun shows no case distinction, but in the plural the dative takes /n/. Thus, we may assume that all three have been formed from a root that has no case and no number by adding successively a number and a case morpheme. (Determiners and adjectives also get a gender morpheme.) Let us concentrate on number. If we assume that the plural morpheme contributes its meaning every time it is added to a root word we face several problems. The first, harmless one, is that we keep iterating the same meaning. The second, less harmless one, is that in case an agreement marker is ambiguous, this ambiguity multiplies.

A third problem arises because the meaning we get under the standard analysis are not correct. Neither does the plural attach directly to the adjectival root, not does apply to the adjective-NP complex. To make this concrete, here is an entry for the agreement suffix /e/ on the adjective:

(4.51)

$$\begin{array}{c} /\mathtt{e}/\oslash \\ \hline \langle p : \triangle : \begin{bmatrix} \text{NUM}: & pl \\ \text{CASE}: & nom \\ \text{GEN}: & masc \end{bmatrix}\rangle\langle x : \nabla : \begin{bmatrix} \text{NUM}: \star \\ \text{CASE}: \star \\ \text{GEN}: \star \end{bmatrix}\rangle \\ \hline x, p \\ \hline p = \{x : p(x)\} \end{array}$$

This results in a meaning where plural on the adjective A means "several things that are A". The introduction of two variables is necessary, though it is awkward that we need two AISs for them. The next chapter provides an alternative solution.

This is incorrect. The addition of plural to /`schwarz`/ 'black' will result in an adjective that denotes a group of black things, while the addition to /`Kater`/ 'tomcat' will result in a noun denoting a group of tomcats. In sum, the above will then say that we are dealing with a group of things that is a group of black things in addition to being a group of tomcats. This does not work when the adjective is not intersective. For example, a group of big mice is not necessarily a group of big things and a group of mice. Thus the iteration of the plural meaning creates misleading if not downright incorrect meanings. We should establish in detail where exactly the plural meaning comes into being.

Now, however, we face a bracketing paradox: while the plural morpheme is attached to the words, they are semantically not construed as acting on the meanings of the roots directly. Similar problems arise with other morphological categories, for example case. There is a good reason why Mel'cuk 1993 – 2000 distinguishes case on adjectives from case on nouns. They are not only morphologically distinct; when looking at it from a semantic angle, we expect case to be added only once, to the phrase. However, we find it spelled out many times for agreement purposes. To solve this, we distinguish case that is added to the head from case that is added to its dependents. The element that primarily receives this case is morphologically speaking the noun, though from a semantic point of view it must be the noun phrase. This however only deals with nominal modification.

Agreement results in a tension between morphology and semantics. From a semantic viewpoint, a certain semantic fact needs to be expressed only once. Agreement however expresses it over and over. Let us call this phenomenon **overexposure**. Something is overexposed if it is expressed twice. Notice though that agreement is not what is overexposed: if anything it is the accompanying meaning that is overexposed. Typically, in semantics agreement is either ignored or it is simply assumed to have the same meaning wherever it occurs. Agreement is however only borderline semantic. This can be seen from its history. Affixes usually derive from independent words, which have lost their original meaning over time. Notice that this story, which is well documented, is in need of an explanation: as independent words tend not to be repeated over and over we are in need of an explanation how it is that affixes suddenly get repeated. Suffixaufnahme is one pathway (see Plank 1995). Another, suggested in Dixon 2002, is analogi-

cal formation based on a different language. The latter explains case agreement as spreading from one language to another. (It still leaves open how the source language developed its case marking pattern to begin with.) Agreement in other words is a scandal, something that shouldn't have happened theoretically, but did happen quite frequently.

We need to solve the problems that this poses. The solution will consist in treating most of the occurrences of an agreement morpheme as purely formal: they will not contribute their meaning. In principle this can be done by treating the same morpheme as occurring in two different roles, see Kracht 2002a. Or by assuming that what appears to be one morpheme is really two different morphemes: one formal morpheme, adding agremeent features, and a semantic one, adding the corresponding meaning. This latter is in line with the proposal by Mel'cuk, which we shall follow here.

In the life of an agreement feature we can distinguish three different types of locations: the word which consumes the feature (by selecting that feature as part of its argument), the words which transmit the feature (for example adjuncts) and finally the word which carries the feature as part of its argument structure of a variable that is neither head nor adjunct. For example, the Latin sentence contains four occurrences of the plural feature, each associated with the same object.

	`laborant`	`tres`	`fortes`	`agricolae`
(4.52)	work-3.PL	three-PL	strong-PL	farmer-PL
	▽	◊	◊	△

The noun creates the variable and with it the feature "plural", and the adjective and the numeral each transmit the feature until it is consumed by the verb. All these four occurrences must be considered as part of one meaning expressed (we also say **exposed**) four times.

The really crucial points are the beginning and the end of the chain. In fact, at both ends interesting things happen. As a rule of thumb, the selecting head decides what the meaning of the feature is going to be. In the case above this is the verb. For it turns out that some elements select a feature without there being an independent meaning associated with it. The clearest example is that of case selection. The accusative by itself can mean a few things, for example a stretch of time. On the other hand, if a verb selects accusative this meaning is clearly absent in the construction. In fact it is safe to say that there is no meaning associated with the accusative whatsoever. The feature has become purely formal. Yet, number is

unlike case in that it typically is not selected for. We speak of case selection, but not of number selection. The way to account for this difference is rather subtle, however.

Let us now look at the problematic issues. First, there is a case of nouns that are morphologically plural but lack plural meaning.

Example 21. English /scissors/ is a plural noun in singular meaning. It controls plural agreement.

(4.53) The scissors are on the table.

Suppose we posit a root */scissor/ '(scissor) blade', applying the plural would result in something like 'a group of blades'. This is not the correct meaning. Furthermore, we would expect there to be a singular form */scissor/. But there is none. Thus, we arrive at the conclusion that this is a primitive sign. ⚽

I digress here into the problem of blocking. In many morphological accounts it is postulated that irregular forms block regular forms by their existence. For example, English verbs do not distinguish a form for the first person singular; one says /I run/ just as one says /I like to run/. However, we do say /I am/ rather than */I be/. The reason for this is said to be the existence of the special form /am/. This is called blocking. Blocking is a global phenomenon: to know why some form is not licit you have to look at the entire set of forms. The existence of a special form blocks the general form. Thus we do not need to stipulate that the form /be/ is incorrect, it will simply never be produced in the presence of the specific form /am/. However, blocking seems to be whimsical. Blocking predicts that we cannot have at the same time regular plurals (like /formulas/) and irregular ones (like /formulae/). The irregular form will by its existence rule out the regular form. It so turns out that blocking is not a global mechanism. Sometimes special forms overrule regular formations, sometimes not.

In the present theory, there is no way to use blocking. This is because any form that can be produced is well-formed. The only way to block the existence of regular forms is not to postulate all of the elements from which it could be formed. In the case of /scissors/ we simply assume that the root form does not exist. In the case of /formulae/ we either postulate it (and then the regular exists as well) or we do not. In the latter case, however, we must posit an addition singular form /formula/ (not to be confused with the root). Blocking is an instance of what

is known as the *elsewhere principle*, or *default*. The idea is that a more special process overrides the general one. It does not matter therefore that blocking is actually mostly used in connection with derivation. For the abstract logical problem is the same with inflection. (See also the discussion of blocking in Bauer 2003.)

Let us finally look at the other end. When a feature is consumed it is up to the head to decide whether or not it has a meaning and if so what it is. It turns out, however, that matters are a little different. With respect to case it is clearly so that the meaning is decided by the selecting head. On the other hand, with number matters seems to be different. There are no idiosyncrasies I know of that require an argument to be singular and that fix of that singular in advance.

We shall return to the problem of agreement in more detail in **agr4-5** Principally, the solution lies in the following. We assume that the fate of an agreement morpheme is decided at the end when it is consumed as part of an argument. Until then it is considered purely formal, as an agreement device. This means in practice that even if the morpheme has meaning, the following mergers will not result in any addition of meaning: adding the feature to an adjunct variable or adding the feature to an argument variable.

This can be achieved by splitting the agreement morpheme into at least two morphemes, only one of which carries meaning. There will be a plural agreement suffix for adjectives, one for nouns, and one for verbs. It will turn out that in part this split is justified. In languages where the verb overtly agrees with two of its arguments we need to distinguish subject plural agreement from object plural agreement. However, the distinction between plural agreement on the adjective and plural agreement on the noun is moot; for they are often even formally similar and this unity needs to be explained.

Given that there is *overexposure* we expect there to be *underexposure* as well. Logically, however, these are not the same. We do not say, for example, that gender is underexposed in Hungarian. For gender is simply not marked. Thus, underexposure can only exist in virtue of the fact that a language requires exposure, at least in general.

Example 22. German nouns and pronouns nouns have three different genders, see also Example 17. It this way, the word /`Bäcker`/ is masculine but may in principle also refer to a woman. There have been various attempts to choose forms

that do not expose masculine gender (irrespective of their semantics). The gerund has recently been a much favoured solution. Thus one speaks of /`Studierende`/ with literal meaning 'those who study' in place of /`Studentinnen`/ '(female) students' or /`Studenten`/ 'students', thus obviating the need to choose either form. This solution capitalises on the fact that in the plural no distinction between masculine and feminine gender exists. In the singular we do however have two forms: /`Studierender`/ '(male) who studies' and /`Studierende`/ '(female) who studies'. Notice that the latter is identical to the plural form. ⚽

Cases of underexposure are thus somewhat more difficult to diagnose. Consider however a language in which plural is either optional or in some cases obligatorily absent. The second case is perhaps clearer. In certain languages, plural is not marked on all noun types. Typically, there is a split: the more animate, the more likely plural is marked on the noun. Languages differ in where that split point is. When they are not marked for number, they may nevertheless command plural agreement on the verb. The *animacy hierarchy* is as follows.

(4.54) 1 > 2 > 3 > kin > animate > inanimate

The rule is as follows.

> [Agreement and Animacy]
> If a noun controls number agreement, then so do all nouns equal or higher in the hierarchy.

Example 23. One such case is the language Muna, Corbette 2000, page 92-3. Pronouns control number agreement.

	`ihintu-umu`	`o-kala-amu`
(4.55)	2-PL	2-go-PL
	'you go'	

Inanimates show singular agreement regardless of their number.

	`bara-hi-no`	`no-hali`
(4.56)	good-PL-his	3.SG.REAL-expensive
	'his goods are expensive'	

Non human animates may or may not be accompanied by a plural marker on the verb.

	o	kadadi-hi	no-rato-mo	/do-rato-mo
(4.57)	ART	animal-PL	3.SG.REAL-arrive-PFV	/3.PL.REAL-arrive-PFV
	'The animals have arrived.'			

Thus, only human animates exert obligatory control. ⚽

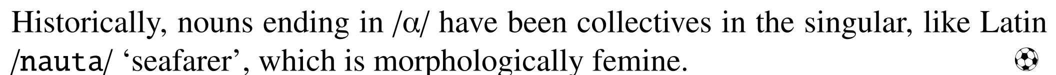
As these examples show, number marking may or may not exist on the verb. But the nouns are required to show singular or plural marking depending on the actual plurality. In these cases, the actual plurality is **underexposed** on the verb. The idea is that the inanimates simply do not show a singular/plural distinction, not that there are two empty suffixes for them. In the given context, non-human animates show an interesting pattern. They may behave like humans, in which case plurality must be marked, or they may be classed as inanimates, in which case the verb is in the singular.

Often, languages exhibit certain irregularities concerning agreement, whose source is not always apparent.

Example 24. In Ancient Greek number marking is obligatory on the nouns, and the verb shows agreement. However, neuter nouns in the plural ending in /α/ trigger singular agreement on the verb. Specifically, the rule is that while the verb is generally singular, the predicative noun is nevertheless in the plural.

(4.58) `Ta megala dora tes tuches ekhei phobon.`
The big gifts of luck give(sg) fear.

(4.59) `Panta ta dikaia kala estin.`
All that which is honorable is(sg) good(pl).


Historically, nouns ending in /α/ have been collectives in the singular, like Latin /`nauta`/ 'seafarer', which is morphologically femine. ⚽

Exercise 34. Spell out the details of the proposal above concerning words in Latin. Create entries for nouns, adjectives and agreement suffixes such that adjective can only merge with fully inflected nouns. The status of a word can be signalled in different ways. One is by an alphabetic symbol (for example the blank,

as is used in this text) and context requirements in glued strings or by means of a particular morphological feature (realised by an attribute WORD with values *true* and *false*).

Exercise 35. Similarly, provide an analysis for Hungarian NPs where the adjectives are always uninflected.

Exercise 36. Write a lexicon that can deal with the facts of Muna as shown in Example 23.

Exercise 37. Write a lexicon to account for the Greek data in Example 24.

4.4 Infinitives and Complex Predicates

Infinitives provide interesting evidence that fusion and polyadicity are necessary for a proper account of the data. We shall see that polyadicity is necessitated by control, while fusion is needed among other for agreement. Throughout this section we do not make use of discontinuity. We shall return to discontinuity at the end of the section.

Recall that three very closely related languages, English, Dutch and German, behave very differently with respect to embedded infinitives. We give an example. (4.60) can be translated by (4.61) into German and by (4.62) into Dutch. (There are—at least in German—many different ways to express (4.60), of which (4.61) is one.)

(4.60) `I said that Karl saw Peter let Mary teach the children to swim.`

(4.61) `Ich sagte, dass Karl Peter Maria die Kinder schwimmen lehren lassen sah.`

(4.62) `Ik zei dat Karl Peter Maria de kinderen zag laten leren zwemmen.`

The patterns are as follows.

(4.63)

English:	$\ldots \mathrm{NP}_1\ \mathrm{V}_1\ \mathrm{NP}_2\ \mathrm{V}_2\ \mathrm{NP}_3\ \mathrm{V}_3\ \mathrm{NP}_4\ \mathrm{V}_4$
German:	$\ldots \mathrm{NP}_1\ \mathrm{NP}_2\ \mathrm{NP}_3\ \mathrm{NP}_4\ \mathrm{V}_4\ \mathrm{V}_3\ \mathrm{V}_2\ \mathrm{V}_1$
Dutch:	$\ldots \mathrm{NP}_1\ \mathrm{NP}_2\ \mathrm{NP}_3\ \mathrm{NP}_4\ \mathrm{V}_1\ \mathrm{V}_2\ \mathrm{V}_3\ \mathrm{V}_4$

Here, NP_i is the subject of the infinitive V_i (and the object of V_{i-1} for $i > 1$). If we are only interested in the strings generated then all of these languages are context free. However, the semantic dependencies introduce trans-context freeness in the case of German and Dutch. (Recall that we do not consider discontinuity.) This means that a standard context free grammar backbone cannot generate the correct semantics for these structures. Now, it is not hard to show that the fusion free calculus is context free. For it is equivalent to a fragment of the bidirectional AB-Calculus, which is known to be context free (see Kracht 2003b). This provides an abstract argument why fusion is needed. We shall show in this section that it can actually also provide a correct analysis of complex predicates in all three languages.

First of all note that infinitives differ from finite verbs in that they do not assign case to their subject argument. This means that the subject of the infinitive must be expressed in a higher clause, since in these languages overt NPs require case. So, while /`John swims`/ is a well-formed sentence, since /`John`/ actually has case and so can be the argument of swims, in /`John to swim`/ this is not the case and the sentence is ungrammatical. In the sentence /`Mary asked John to swim`/ the constituent /`John`/ actually has accusative case, which we can demonstrate by exchanging it for a pronoun. We have

(4.64) `He swims.`

(4.65) *`Mary asked he to swim.`

(4.66) `Mary asked him to swim.`

Therefore, /`John`/ occupies the object position of the verb /`asked`/ and not the subject position of the verb /`to swim`/. In the analysis below infinitives therefore do not assign any case, not even nominative, which is typically unmarked.

We start with the English construction. There are two types of verbs, basic verbs such as /`swim`/ and serial verbs such as /`let`/. /`let`/ takes an NP and an infinitive as a complement and requires that the object NP is the subject of the embedded infinitive. Therefore the semantics of /`swim`/, /`Mary`/ and /`let`/ is as

follows. (We have simplified the representation of the AVSs.)

$$\begin{array}{c}\texttt{/swim/}\bigcirc\\ \begin{array}{|l|}\hline \langle e :\!\vartriangle\!: [\textsc{agr} : \mathit{inf}]\rangle \\ \hline \hline e \\ \hline \mathsf{swim}'(e); \\ \mathsf{act}'(e) \doteq x. \\ \hline \end{array}\end{array} \qquad \begin{array}{c}\texttt{/Mary/}\bigcirc\\ \begin{array}{|l|}\hline \langle x :\!\vartriangle\!: [\textsc{case} : \mathit{acc}]\rangle \\ \hline \hline x \\ \hline x \doteq m. \\ \hline \end{array}\end{array}$$

(4.67)

$$\begin{array}{c}\texttt{/let/}\bigcirc, \ogreaterthan, \ogreaterthan\\ \begin{array}{|ll|}\hline \langle e :\!\vartriangle\!: [\textsc{agr} : \mathit{inf}]\rangle, & \\ \langle f : \triangledown : [\textsc{agr} : \mathit{inf}]\rangle, & \langle y : \triangledown : [\textsc{case} : \mathit{acc}]\rangle \\ \hline \hline e, f, y & \\ \hline \mathsf{let}'(e); & \mathsf{act}'(e) \doteq x; \\ \mathsf{thm}'(e) \doteq f; & \mathsf{ben}'(e) \doteq y; \\ \mathsf{act}'(f) \doteq y. & \\ \hline \end{array}\end{array}$$

So, there is an event e of letting, and its theme (that which is let to be the case) is f. One may ask whether it is necessary to assume that verbs selecting an event actually also involve existential quantification over that event. The word /`let`/ might only mean that there is an event of letting but not necessarily that there is an event that is being let to be the case. For example, if I let someone enter my room, he can still decide not to enter at all. There is however still the event of me letting him enter my room. So, we might decide not to put f into the upper box but rather into some embedded box. Similarly with /`persuade`/, where the secondary event might actually be in the future as in /`persuade to go to London`/. Intensional verbs are still different. Eventually, this must be resolved by appeal to parameters, see Chapter 5. Nothing of substance hinges on the semantic analysis given here, however.

The beneficiary of the letting is y; y is also the actor of f. So, y is doing double duty: it is the beneficiary of the letting event but the actor of the embedded event. This is desirable since it allows to incorporate the distinction between subject control and object control of infinitives. We exemplify this with the verbs

/`promise`/ and /`persuade`/.

(4.68)

$$/\texttt{promise}/\bigcirc, \ominus, \ominus$$

$\langle e :\triangle: [\textsc{agr} : \mathit{inf}]\rangle,$ $\langle f : \triangledown : [\textsc{agr} : \mathit{inf}]\rangle, \quad \langle y : \triangledown : [\textsc{case} : \mathit{acc}]\rangle.$	
e, f, x, y	
$\mathsf{promise}'(e);$	$\mathsf{act}'(e) \doteq x;$
$\mathsf{thm}'(e) \doteq f;$	$\mathsf{ben}'(e) \doteq y;$
$\mathsf{act}'(f) \doteq x.$	

(4.69)

$$/\texttt{persuade}/\bigcirc, \ominus, \ominus$$

$\langle e :\triangle: [\textsc{agr} : \mathit{inf}]\rangle,$ $\langle f : \triangledown : [\textsc{agr} : \mathit{inf}]\rangle, \quad \langle y : \triangledown : [\textsc{case} : \mathit{acc}]\rangle.$	
e, f, x, y	
$\mathsf{persuade}'(e);$	$\mathsf{act}'(e) \doteq x;$
$\mathsf{thm}'(e) \doteq f;$	$\mathsf{pat}'(e) \doteq y;$
$\mathsf{act}'(f) \doteq y.$	

/`promise`/ has two arguments besides the subject (x), namely the beneficiary, y, and the infinitive. /`persuade`/ differs only in the thematic role of y; here it is a patient, but this is insignificant for the present purposes. Now, while the actor of the complement f is x in the case of /`promise`/, it is y in the case of /`persuade`/. Therefore, with this semantics, for (4.70) it turns out that it is Albert who will do the theorem proving and that in (4.71) it is Jan. This is as it should be.

(4.70) `Albert promises Jan to prove new theorems.`

(4.71) `Albert persuades Jan to prove new theorems.`

A problematic aspect of the present analysis is the fact that it assumes that the subject of the lower infinitive is the actor; it must do so in order to identify the

subject. There is an alternative solution which makes use of polyadic merge.

(4.72)

$$
\begin{array}{|l|}
\multicolumn{1}{c}{\texttt{/swim/}\bigcirc,\bigcirc} \\
\hline
\langle e :\vartriangle: [\textsc{agr} : \mathit{inf}]\rangle, \\
\langle x :\vartriangle: [\textsc{case} : \star]\rangle \\
\hline
e, x \\
\hline
\mathsf{swim}'(e); \\
\mathsf{act}'(e) \doteq x. \\
\hline
\end{array}
\qquad
\begin{array}{|ll|}
\multicolumn{2}{c}{\texttt{/let/}\bigcirc,\bigcirc,\oslash,\oslash,\oslash} \\
\hline
\langle e :\vartriangle: [\textsc{agr} : \mathit{inf}]\rangle, & \\
\langle x :\vartriangle: [\textsc{case} : \star]\rangle, & \\
\langle y : \triangledown : [\textsc{case} : \star]\rangle, & \\
\langle f : \triangledown : [\textsc{agr} : \mathit{inf}]\rangle, & \\
\langle z : \triangledown : [\textsc{case} : \mathit{acc}]\rangle. & \\
\hline
e, f, x, y, z & \\
\hline
\mathsf{let}'(e); & \mathsf{act}'(e) \doteq x; \\
\mathsf{thm}'(e) \doteq f; & \mathsf{ben}'(e) \doteq z; \\
y \doteq z. & \\
\hline
\end{array}
$$

Here, /`let`/ can be merged with /`Mary`/ if the latter carries accusative case. The result is as follows.

(4.73)

$$
\begin{array}{|ll|}
\multicolumn{2}{c}{\texttt{/let Mary/}\bigcirc,\bigcirc,\oslash,\oslash} \\
\hline
\langle e :\vartriangle: [\textsc{agr} : \mathit{inf}\rangle, & \\
\langle x :\vartriangle: [\textsc{case} : \star]\rangle, & \\
\langle y : \triangledown : [\textsc{case} : \star]\rangle, & \\
\langle f : \triangledown : [\textsc{agr} : \mathit{inf}]\rangle, & \\
\hline
e, f, x, y, z & \\
\hline
\mathsf{let}'(e); & \mathsf{act}'(e) \doteq x; \\
\mathsf{thm}'(e) \doteq f; & \mathsf{ben}'(e) \doteq z; \\
z \doteq m; & y \doteq z. \\
\hline
\end{array}
$$

Next we perform polyadic merge with /`swim`/ and we get

(4.74)

$$
\begin{array}{|ll|}
\multicolumn{2}{c}{\texttt{/let Mary swim/}\bigcirc,\bigcirc} \\
\hline
\langle e :\vartriangle: [\textsc{agr} : \mathit{inf}]\rangle, & \\
\langle x :\vartriangle: [\textsc{case} : \star]\rangle, & \\
\hline
e, \quad f & \\
\hline
\mathsf{let}'(e); & \mathsf{act}'(e) \doteq x; \\
\mathsf{thm}'(e) \doteq f; & \mathsf{ben}'(e) \doteq z; \\
z \doteq m; & \mathsf{swim}'(f); \\
\mathsf{act}'(f) \doteq y. & y \doteq z. \\
\hline
\end{array}
$$

This is the desired result. One may think that importing the same argument (though under different variables) twice is unsound. However, in the current calculus this is a perfectly legal option.

Now notice the following. The verb /`to let`/ demands three arguments, an infinitival complement, an object noun phrase and a subject noun phrase. If we allow the arguments to be accessed freely then the object NP and the infinitive can be freely permuted. This means that the sentence (4.75) would be grammatical, contrary to fact.

(4.75) *`I said that John lets swim Mary.`

Fortunately, the conditions of E-access rule this out. If there is an argument that is prior to the sentential complement, then that argument must be merged away first. The only grammatical constructions are therefore those in which the serial verbs take first a complement NP to the right, and then an infinitival complement. These are exactly the facts of English.

Now we turn to German. The major difference between German and English is the directionality of the selection. The verbs in German select both the noun phrase and the infinitive to their left. An additional difference is that the infinitive is selected first. The representations for /`schwimmen`/ and /`lassen`/ are therefore as follows.

(4.76)

/`schwimmen`/$\bigcirc, \bigcirc$

$\langle e : \vartriangle : [\textsc{agr} : \mathit{inf}] \rangle .$ $\langle x : \vartriangle : [\textsc{case} : \star] \rangle$
e, x
$\mathsf{swim}'(e);$ $\mathsf{act}'(e) \doteq x.$

(4.77)

/`lassen`/$\bigcirc, \bigcirc, \oslash, \oslash, \oslash$

$\langle e : \vartriangle : [\textsc{agr} : \mathit{inf}] \rangle ,$	$\langle z : \vartriangle : [\textsc{case} : \star] \rangle ,$
$\langle y : \triangledown : [\textsc{case} : \mathit{acc}], \rangle ,$	$\langle x : \triangledown : [\textsc{case} : \star] \rangle ,$
$\langle f : \triangledown : [\textsc{agr} : \mathit{inf}] \rangle .$	
e, f, x, y, z	
$\mathsf{let}'(e);$	$\mathsf{act}'(e) \doteq z;$
$\mathsf{thm}'(e) \doteq f;$	$\mathsf{ben}'(e) \doteq y;$
$x \doteq y.$	

Notice that German /`lassen`/ takes one more argument than English /`let`/. This is necessitated by the different syntax. The German infinitive first merges polyadically with the infinitive; at this stage the variable x is identified. Only after that the accusative object is taken.

For German, too, we must posit a restriction on access, this time however for the accusative object. We prohibit skipping of the event referents. Otherwise (4.61) would be grammatical just like (4.79).

(4.78) *`Ich sagte, dass Karl schwimmen Peter ließ.`
I said that Karl swim Peter let.

(4.79) `Ich sagte, dass Karl Peter schwimmen ließ.`
I said that Karl Peter swim lets.

The syntactic data of German are complicated by many factors, one being that auxiliaries and verbs do not let an infinitive appear on the right hand side, but many other verbs do. One such verb is /`helfen`/. Another complication is that while the infinitive appears on the left hand side, the finite clause complements are typically on the right side. However, these additional facts can be incorporated by suitable changes in the argument structure. Before we can address that we shall discuss the case of Dutch.

In Dutch, the facts get rather involved. First of all, from abstract arguments we know that merge alone would not allow to generate the Dutch data. This is so because Dutch is not strongly context free. If we allow only the merge, then we have only finitely many rules, each of which are context free. Moreover, the syntactic relations are mirrored by the semantic relations in a rather straightforward way. The subject and object of a verb must be within the extended projection of the verb. Hence, we must allow for fusion of argument structures. Specifically, we allow two verbs to fuse their argument structures. This generates a structure that is quite similar to the ones found in the literature (GB and LFG). The verbs join into a big cluster and only after that the NP arguments are discharged, one after the other. To account for these facts we do the following. We introduce two kinds of vertical diacritics, one for merge and another for fusion. So, if an argument is selected for, the functor can choose whether or not it selects through fusion. We write $\blacktriangledown$ and $\blacklozenge$ if fusion is required and $\triangledown$ and $\lozenge$ otherwise. We could also introduce a third kind for those heads that allow both merge and fusion for their arguments (as is the case with German verbs) but this can also be accounted for by using different lexical entries. Thus we refrain from introducing more symbolism here.

The semantics of the verbs /`zwemmen`/ and /`laten`/ are now as follows.

(4.80)

$$
\begin{array}{c}
/\texttt{zwemmen}/\bigcirc,\bigcirc \\
\boxed{\begin{array}{l}
\langle e :\triangle: [\textsc{agr} : \mathit{inf}]\rangle. \\
\langle x :\triangle: [\textsc{case} : \bigstar]\rangle.
\end{array}} \\
\boxed{\begin{array}{l}
e, x \\ \hline
\mathsf{swim}'(e); \\
\mathsf{act}'(e) \doteq x.
\end{array}}
\end{array}
$$

(4.81)

$$
\begin{array}{c}
/\texttt{laten}/\bigcirc,\bigcirc,\olessthan,\bigcirc,\ogreaterthan \\
\boxed{\begin{array}{l}
\langle e :\triangle: [\textsc{agr} : \mathit{inf}]\rangle, \\
\langle x :\triangle: [\textsc{case} : \bigstar]\rangle, \\
\langle y : \triangledown : [\textsc{case} : \mathit{acc}]\rangle, \\
\langle z : \triangledown : [\textsc{case} : \bigstar]\rangle, \\
\langle f : \blacktriangledown : [\textsc{agr} : \mathit{inf}]\rangle.
\end{array}} \\
\boxed{\begin{array}{ll}
e, f, x, y, z & \\ \hline
\mathsf{let}'(e); & \mathsf{act}'(e) \doteq x; \\
\mathsf{thm}'(e) \doteq f; & \mathsf{ben}'(e) \doteq y; \\
\mathsf{act}'(f) \doteq y; & y \doteq z.
\end{array}}
\end{array}
$$

Now the verb is looking first for an event referent to the right, and then for an object referent to its left. The rule for fusion is as follows. The entire argument structure minus the first entry of the argument is inserted in place of the referent that it identifies with; the identified referent is cancelled. In polyadic fusion all mergers are performed before the remainder is added to the list. (The precise details of the serialisation do not matter throughout this book.) The serial verb /`zag laten`/ is generated through fusion.

(4.82)

$$
\begin{array}{c}
/\texttt{zag laten}/\bigcirc,\bigcirc,\olessthan,\olessthan,\ogreaterthan \\
\boxed{\begin{array}{ll}
\langle e :\triangle: [\textsc{agr} : \mathit{past}\rangle, & \langle x :\triangle: [\textsc{case} : \bigstar]\rangle, \\
\langle y : \triangledown : [\textsc{case} : \mathit{acc}]\rangle, & \langle z : \triangledown : [\textsc{agr} : \mathit{acc}]\rangle, \\
\langle f : \blacktriangledown : [\textsc{agr} : \mathit{inf}]\rangle. &
\end{array}} \\
\boxed{\begin{array}{ll}
e, f, x, y, z & \\ \hline
\mathsf{see}'(e); & \mathsf{act}'(e) \doteq x; \\
\mathsf{thm}'(e) \doteq f; & \mathsf{ben}'(e) \doteq y; \\
\mathsf{act}'(f) \doteq y; & \mathsf{let}'(f); \\
\mathsf{thm}'(f) \doteq f; & \mathsf{act}'(f) \doteq z; \\
y \doteq z. &
\end{array}}
\end{array}
$$

Now notice that the lexemes for Dutch are different from the German ones in that they select the infinitive to the right, and they are also different from the English ones because the nominal arguments are consistently to the left. If Dutch has the extra option of fusion then not only would those sentences be grammatical which use fusion but also those which can be obtained through standard merge. The following sentences would be grammatical (with the meaning being that of (4.62)).

(4.83) *`Ik zei dat Karl Peter Maria [zag laten] de kinderen [leren zwemmen]`.

(4.84) *`Ik zei dat Karl Peter zag [Maria de kinderen [laten leren zwemmen]]`.

(4.85) *`Ik zei dat Karl Peter zag [Maria laten [de kinderen [leren zwemmen]]]`.

Namely, /`leren zwemmen`/ is a one-place predicate taking an argument to the left (the one who is being taught). This argument is consumed to the left giving rise to a zero-place predicate /`de kinderen leren zwemmen`/. This shows why (4.85) is generated by the calculus. Likewise, /`laten leren zwemmen`/ will be a two-place predicate taking as first argument to the left the one who is being taught and secondly the one who is being let to do the teaching. This explains the sentence (4.84). (4.83) is generated as follows. /`zag laten`/ is a three-place predicate taking an infinitival complement, in this case /`de kinderen leren zwemmen`/. We shall stress that fusion is not a global option for Dutch. So, it is not generally the case that Dutch allows fusion in contrast to English. Rather, it is specific arguments that allow for fusion in contrast to others.

It has been argued that German verbs too trigger fusion (even though you cannot see that by looking at our examples). So, the lexical entry for /`lassen`/ is now as follows.

(4.86)

$$\texttt{/lassen/}\bigcirc, \bigcirc, \olessthan, \olessthan, \olessthan$$

$$\begin{array}{|ll|}\hline \langle e :\triangle: [\textsc{case} : \mathit{inf}]\rangle, & \langle x :\triangle: [\textsc{case} : \bigstar]\rangle, \\ \langle y : \triangledown : [\textsc{case} : \mathit{acc}]\rangle, & \langle z : \triangledown : [\textsc{case} : \bigstar]\rangle, \\ \langle f : \blacktriangledown : [\textsc{agr} : \mathit{inf}]\rangle. & \\ \hline \end{array}$$

$$\begin{array}{|ll|}\hline e, f, x, y, z & \\ \hline \mathsf{let}'(e); & \mathsf{act}'(e) \doteq x; \\ \mathsf{thm}'(e) \doteq f; & \mathsf{ben}'(e) \doteq y; \\ \mathsf{act}'(f) \doteq y. & y \doteq z. \\ \hline \end{array}$$

We shall stipulate the following.

> [Fusion First]
> If an argument is selected through fusion, it must be selected first.

(Moreover, there shall be in general at most one argument that can be selected through fusion.) This generalizes the restriction we have made with respect to access in the German verb.

So, on what grounds are (4.83) – (4.83) excluded? We shall say in addition that fusion is restricted (in Dutch) to words and moreover it produces only words. So, /`laten`/ selects only words through fusion, and when /`laten`/ and /`zwemmen`/ merge, the result is again a word. Then the examples (4.83) – (4.85) are excluded. To see this, look at the argument structure of /`zag laten`/ in (4.82). The f^2 argument is inherited from /`laten`/. Since /`laten`/ is in turn a raising verb, this argument is identified through fusion. This means that it must be a word and it must be the first that is identified. Hence, it can neither combine with /`de kinderen`/, since this is not an event, nor with /`de kinderen leren zwemmen`/, since that is not a word. The same arguments work for (4.84) and (4.85).

Therefore, this analysis gets at least the basic syntactic structure right. Let us now turn to word order variation in Dutch and German. In Dutch there is next to no morphological variation, and so the arguments may not be permuted. Therefore, (4.87) and (4.88) cannot be taken to mean the same as (4.62).

(4.87) `Ik zei dat Peter Karl Maria de kinderen zag laten leren zwemmen.`

(4.88) `Ik zei dat Karl de kinderen Peter Maria zag laten leren zwemmen.`

The same holds for German. However, in those cases where there is a morphological differentiation, alternative word orders are allowed. So, (4.89) – (4.91) all mean the same as (4.92).

(4.89) `Ich sagte Karl, dass ich ihr den Kühlschrank zu reparieren versprochen hatte.`

(4.90) `..., dass den Kühlschrank ich ihr zu reparieren versprochen hatte.`

(4.91) ..., `dass ihr den Kühlschrank ich zu reparieren versprochen hatte.`
'I told Karl, that I promised her to repair the refrigerator.'

These examples show that we are really dealing with a complex predicate here (or, following traditional usage, we have a phenomenon of clause union). For the arguments can be serialized differently exactly when they exhibit clear morphological differentiation with respect to the names in question.

We close by noting that German allows even freer word order than permitted by the present system. Notably, infinitives are allowed to consume their case marked arguments before they fuse into a complex predicate.

(4.92) ..., `dass ich` [`den Kühlschrank zu reparieren`] [`ihr versprochen hatte`].

(4.93) ..., `dass` [`den Kühlschrank zu reparieren`] `ich ihr versprochen hatte.`

However notice that we are dealing here with another infinitive, namely the zu-infinitive, which might be responsible for this additional freedom. We shall not discuss this further. Notice that this is a feature of German. In Dutch, this phenomenon is absent. Lack of case marking would result in too much ambiguity. For then the highest raising verb can alternatively take the last NP as its object, rather than the first. Namely, in that case (4.62) can alternatively be rendered as (4.95).

(4.94) `Ik zei dat Karl Peter Maria de kinderen` [`zag` [`laten leren zwemmen`]].

(4.95) `Ik zei dat Peter Maria de kinderen` [[`Karl zag`] `laten leren zwemmen`].

Another alternative for Dutch is to use the analysis given in Calcagno 1995 for Swiss German. It assumes that Dutch infinitives are discontinuous. It is enough to use two parts $\vec{x} \otimes \vec{y}$, where $\vec{x}$ collects the string of NPs and $\vec{y}$ the string of Vs.

Notes on this section. The idea that the complex verbs of German and Dutch form a cluster which functions as a single word, shows up in many other syntactic theories. In GB, the verbs are raised and adjoin to the raising head. This

adjunction is a zero-level (=head-to-head) adjunction. Since zero-level means "is a word for syntactic purposes", we get the distinction between the languages by parametrizing for the availability of raising and for the directionality of adjunction. Hence, fusion is like zero-level adjunction. The word order freedom in German raising constructions has been studied in Becker, Rambow, and Niv 1992. It is claimed there that the construction exceeds the power of Linear Context Free Rewriting Systems (LCFRSs). The argument is based on the fact that clause union is not bounded. However, if we are right, then the order of arguments is nevertheless restricted by their overt morphology. Since there are only a finite number of cases to deal with, not all serializations of the arguments can go together with the same meaning. This does not affect the string language, though.

Exercise 38. Hungarian shows object agreement. Mainly, different forms exist for definite objects and indefinite.

(4.96) `Látok egy madarat.`
I see-1.SG.INDEF INDEF bird
'I see a bird.'

(4.97) `Látom a madarat.`
I see-1.SG.DEF DEF bird
'I see the bird.'

The following shows that object agreement is used also for complex verbs.

(4.98) `Milyen színben akarod látni a világot?`
In what colour do you want to see the world?

Here, /`akarod`/ 'you want (it)' is the definite form of the 2nd singular. The root is /`akar`/ 'want'. The verb takes /`látni`/ 'to see' as its complement, and they form a complex predicate taking /`a világot`/ 'the world' as its complement. Try to accound for the definite inflection on the verb in view of these facts.

Exercise 39. The English verb /`seem`/ shows rather odd semantic behaviour.

(4.99) `You seem to be clever.`

(4.100) *`Seem you to be clever.`

(4.101) *Seem to be clever.

(4.102) It seems to rain.

The subject of ch6:30 is not the addressee. Rather, the subject is something like 'the fact that you are clever'. So we expect the structure to be

(4.103) [you to be clever] seems

So, /to seem/ has an infinitive subject. However, this infinitive does not assign case to its own subject; and it does not appear to the left of the verb. This is rather exceptional. In Generative Grammar, it is claimed that the subject needs case and so moves to subject position. Provide an alternative account using argument structures where the matrix verb can and must show agreement with the subject of the lower infinitive. This subject must appear to the left hand side of the matrix verb. This will generate exactly the right facts.

4.5 Logical Connectives, Groups and Quantifiers

Merge as we have defined it so far is monotone: the reader may check that if we have structures $\mathfrak{S}_1$ and $\mathfrak{S}_2$, then the meaning of $\mathfrak{S}_1 \oplus \mathfrak{S}_2$ logially implies the meaning of $\mathfrak{S}_1$ and $\mathfrak{S}_2$. Thus the semantics is incomplete: there is no way to implement a semantics of negation. (Actually, this is not quite correct. We could do the following: every formula φ is translated into $p \doteq \varphi$, and the semantics of negation is, for example, $q \doteq \neg p$. Using the calculus of parameters of Chapter 5 this can be implemented easily. Unfortunately, this will not eliminate the problems of polymorphism.) Meanings are therefore only added, there is no way to negate meanings, or quantify over objects. This is obviously not enough to cover the full range of natural language expressions. In this section we shall propose a new mechanism that will allow to deal with logical connectives and quantifiers. The problem with logical connectives (/and/, /or/, /not/ and so on) is twofold: from a semantic point of view they do not take variables but the entire proposition in their scope; from a combinatorial point of view their syntax is very flexible. In general, they may take arguments of any type. Any constituent may be negated, any two constituents may be coordinated. The only restriction is that in the binary cases we may only take two constituents of the same type. It is not the place here to defend the correctness of this analysis. Keenan and Faltz 1985 have argued convincingly

that any syntactic category forms a boolean structure. Moreover, the exceptions to the identity restriction tend to be marginal so that we simply disregard them.

Let us return to Section 2.2. We have outlined there how in DRT complicated logical structures are built up using various connectives such as $\Rightarrow$, $\cup$, $\vee$ and $\neg$. We will now consider how these connectives can be built into the present system. We stress here once again that the use of DRT has only pedagogical reasons. The technique can easily be recast in dynamic predicate logic if needed. The addition we are going to make is the following. First, in addition to standard variables ranging over objects of various types, there are also variables ranging over DRSs (or propositions). We write them $\boxed{1}$, $\boxed{2}$ and so on. Finally, there are also variables over argument structures, denoted by Gothic letters: $\mathfrak{x}$, $\mathfrak{y}$ and so on. In place of AVS α one may also write $\alpha\dagger\mathfrak{x}$ and in place of the single variable x one may write $x\dagger\boxed{1}$. Any of the two variables is optional. An argument identification statement may then assume any of the following nine forms

$$
(4.104)\quad
\begin{array}{ll}
\text{(a)} & \langle x \quad :\partial:\alpha\rangle \\
\text{(b)} & \langle x \quad :\partial:\alpha\dagger\mathfrak{x}\rangle \\
\text{(c)} & \langle x \quad :\partial:\mathfrak{x}\rangle \\
\text{(d)} & \langle \boxed{1} \quad :\partial:\alpha\rangle \\
\text{(e)} & \langle \boxed{1} \quad :\partial:\alpha\dagger\mathfrak{x}\rangle \\
\text{(f)} & \langle \boxed{1} \quad :\partial:\mathfrak{x}\rangle \\
\text{(g)} & \langle x\dagger\boxed{1}:\partial:\alpha\rangle \\
\text{(h)} & \langle x\dagger\boxed{1}:\partial:\alpha\dagger\mathfrak{x}\rangle \\
\text{(i)} & \langle x\dagger\boxed{1}:\partial:\mathfrak{x}\rangle
\end{array}
$$

However, notice that this is just a notational simplification since a variable occurring in the argument section does not have to be part of the semantics. The fact that the variables are denoted differently than ordinary variables will prevent confusion.

The forms (c), (f) and (i) are just impoverished versions of (b), (d) and (h), respectively, where α is trivial. The rules for merge change in the following way. If $\mathfrak{x}$ is present in μ (Cases (b), (c), (e), (f), (h) and (i)), μ can only merge as the functor, it is not an argument. The merge will succeed if the merge succeeds with $\mathfrak{x}$ dropped, that is, with (a) in place of (b), (d) in place of (e) or (g) in place of (h). In this case, $\mathfrak{x}$ is bound to the entire argument structure of the argument, and $\boxed{1}$ is bound to the semantics underlying it.

Let us give examples. Here are argument structures for /and/, /or/ and /not/.

(4.105)

/and/⊜, ⊝

$\mathfrak{x}$, $\langle x\dagger\boxed{1} : \triangledown : \mathfrak{x}\rangle$, $\langle y\dagger\boxed{2} : \triangledown : \mathfrak{x}\rangle$.
$\boxed{2} \cup \boxed{1}$

/or/⊜, ⊝

$\mathfrak{x}$, $\langle x\dagger\boxed{1} : \triangledown : \mathfrak{x}\rangle$, $\langle y\dagger\boxed{2} : \triangledown : \mathfrak{x}\rangle$.
$\boxed{2} \vee \boxed{1}$

/not/⊝

$\mathfrak{x}$, $\langle x\dagger\boxed{1} : \triangledown : \mathfrak{x}\rangle$.
$\neg\boxed{1}$

(We remark here that $\boxed{2} \vee \boxed{1}$ must be a DRS with a head section. So, it is not simply the disjunction of two DRSs each with their own head section. It also creates a new main head section.) Here is the representation for /every/:

(4.106)

/every/⊝, ⊝

$\langle y : \lozenge : \begin{bmatrix} \text{NUM}: sg \\ \text{CAT}: ob \end{bmatrix}\rangle$, $\langle x\dagger\boxed{1} : \lozenge : \begin{bmatrix} \text{NUM}: sg \\ \text{CAT}: ob \end{bmatrix}\rangle$.
x, y
$y \doteq \{\{x\} : \boxed{1}(x)\}$

The variable x in the argument structure is the same as the x in the DRS, but it is bound there.

To see an easy example, we produce the Latin /non dat/ ‘he does not give’.

The argument structure of /dat/ is:

(4.107)

/dat/○, ⧀, ⧁

$\langle e : \triangle : \begin{bmatrix} \text{PERS}: 3 \\ \text{NUM}: sg \end{bmatrix} \rangle,$
$\langle x : \triangledown : \begin{bmatrix} \text{CASE}: nom \\ \text{NUM}: sg \end{bmatrix} \rangle,$
$\langle y : \triangledown : [\text{CASE} : acc] \rangle,$
$\langle z : \triangledown : [\text{CASE} : dat] \rangle.$

e, x, y, z	
give′(e);	act′$(e) \doteq x$;
thm′$(e) \doteq y$;	ben′$(e) \doteq z$;
time$(e) \doteq$ now′.	

Now, the variable $\mathfrak{x}$ can match any argument structure, in particular the one for /dat/ and we get:

(4.108)

/non dat/○, ⧀, ⧁, ⧁

$\langle e : \triangle : \begin{bmatrix} \text{PERS}: 3 \\ \text{NUM}: sg \end{bmatrix} \rangle,$
$\langle x : \triangledown : \begin{bmatrix} \text{CASE}: nom \\ \text{NUM}: sg \end{bmatrix} \rangle,$
$\langle y : \triangledown : [\text{CASE} : acc] \rangle,$
$\langle z : \triangledown : [\text{CASE} : dat] \rangle.$

e, x, y, z

$\neg$

e	
give′(e);	act′$(e) \doteq x$;
thm′$(e) \doteq y$;	ben′$(e) \doteq z$;
time′$(e) \doteq$ now′.	

(We omit boxes around single entries.) So, we get the same argument structure again. The reason is that when /non/ merges with /dat/, the variable $\mathfrak{x}$ is instantiated to the argument structure of /dat/. Since /non/ also exports $\mathfrak{x}$, and $\mathfrak{x}$ is now instantiated to the argument structure of /dat/, this is the resulting argument structure.

When we approach the other connectives in the same way, we meet a small problem. /and/, for example, will take a complement C to its right, and $\mathfrak{x}$ will

be instantiated to the argument structure of C. Subsequently, /and$^\frown C$/ looks to its left for an element with identical argument structure. However, the variables in the argument structure are part of the name of $\mathfrak{x}$, so if by chance D has the same argument structure as C with the variables being named differently, the merge will not succeed. Here is a simple example:

(4.109)

$$\begin{array}{c}
/\texttt{venit}/\bigcirc, \oslash \\
\boxed{\begin{array}{l}
\langle e : \triangle : \begin{bmatrix} \textsc{cat} & : e \\ \textsc{pers} & : 3 \\ \textsc{num} & : sg \end{bmatrix} \rangle, \\
\langle x : \triangledown : [\textsc{case} : nom] \rangle .
\end{array}} \\
\boxed{e, x} \\
\boxed{\mathsf{come}'(e); \mathsf{act}'(e) \doteq x.}
\end{array}
\bullet
\begin{array}{c}
/\texttt{et}/\oslash, \obslash \\
\boxed{\begin{array}{l}
\mathfrak{x}, \\
\langle \boxed{1} : \triangledown : \mathfrak{x} \rangle, \\
\langle \boxed{2} : \triangledown : \mathfrak{x} \rangle .
\end{array}} \\
\boxed{\boxed{2} \cup \boxed{1}}
\end{array}$$

$$\bullet
\begin{array}{c}
/\texttt{vidit}/\bigcirc, \oslash \\
\boxed{\begin{array}{l}
\langle e : \triangle : \begin{bmatrix} \textsc{cat} & : e \\ \textsc{pers} & : 3 \\ \textsc{num} & : sg \end{bmatrix} \rangle, \\
\langle y : \triangledown : [\textsc{case} : nom] \rangle .
\end{array}} \\
\boxed{e, y} \\
\boxed{\mathsf{see}'(e); \mathsf{act}'(e) \doteq x.}
\end{array}$$

For example, we may make the following choice for $\mathfrak{x}$:

$$\mathfrak{x} := \begin{bmatrix}
\langle e : \triangle : \begin{bmatrix} \textsc{cat} & : e \\ \textsc{pers} & : 3 \\ \textsc{num} & : sg \end{bmatrix} \rangle, \\
\langle y : \triangledown : [\textsc{case} : nom] \rangle .
\end{bmatrix} \qquad (4.110)$$

Only with this choice, the last two can structures can merge and we get the fol-

lowing result:

$$\begin{array}{c} \texttt{/venit/}\bigcirc,\oslash \\ \boxed{\begin{array}{l} \langle e : \vartriangle : \left[\begin{array}{l} \textsc{cat} : e \\ \textsc{pers} : 3 \\ \textsc{num} : sg \end{array}\right] \rangle, \\ \langle x : \triangledown : [\textsc{case} : nom] \rangle. \end{array}} \\ \boxed{\begin{array}{l} e, x \\ \hline \mathsf{come}'(e); \mathsf{act}'(e) \doteq x. \end{array}} \end{array}$$

(4.111)
$$\begin{array}{c} \texttt{/et vidit/}\oslash,\oslash \\ \bullet \; \boxed{\langle \boxed{1}, \vartriangle : \left[\begin{array}{l} \langle e : \vartriangle : \left[\begin{array}{l} \textsc{cat} : e \\ \textsc{pers} : 3 \\ \textsc{num} : sg \end{array}\right] \rangle, \\ \langle y : \triangledown : [\textsc{case} : nom] \rangle \end{array}\right] \rangle} \\ \boxed{\boxed{1} \cup \boxed{\begin{array}{l} e, y \\ \hline \mathsf{see}'(e); \mathsf{act}'(e) \doteq y \end{array}}} \end{array}$$

These two structures cannot merge, since $\boxed{1}$ is identified under a different argument structure, namely the following

(4.112)
$$\mathfrak{x} := \left[\begin{array}{l} \langle e : \vartriangle : \left[\begin{array}{l} \textsc{cat} : e \\ \textsc{pers} : 3 \\ \textsc{num} : sg \end{array}\right] \rangle, \\ \langle x : \triangledown : \textsc{case} : nom] \rangle. \end{array}\right]$$

But this choice is in conflict with the requirement of the third argument structure. The problem is the choice of the variable names, which now have become part of the name of the referent $\boxed{1}$. For our present purposes the following can be done. Say that α and β **match**, in symbols $\beta \approx \alpha$, if β results from α by replacing uniformly certain variables. We now define the merge with respect to second order argument structures as follows. $\langle \boxed{1} : \triangledown : \alpha \rangle$ identifies β if $\beta \approx \alpha$. The merge is as follows.

(4.113)
$$\boxed{\begin{array}{l} \omega, \\ \langle \delta : \triangledown : \alpha \rangle \end{array}} \atop \boxed{\phi(\delta)} \quad \bullet \quad \boxed{\beta} \atop \boxed{\theta} \quad = \quad \boxed{\omega} \atop \boxed{\phi(\theta^{\sigma})}$$

Here, σ is a substitution such that $\beta^\sigma = \alpha$. Hence, under these renewed definitions the above merge can be carried out and we get

(4.114)

/venit et vidit/$\bigcirc, \oslash$

$$\left\langle e : \vartriangle : \begin{bmatrix} \text{CAT} : e \\ \text{PERS} : 3 \\ \text{NUM} : sg \end{bmatrix} \right\rangle, \quad \langle y : \triangledown : [\text{CASE} : nom] \rangle.$$

e, y
$\mathsf{come}'(e); \mathsf{act}'(e) \doteq x; \mathsf{see}'(e).$

A note is in order on the possible values of $\mathfrak{z}$. Since we do not use names but descriptions of names, we can have descriptions that are partial. Hence, we shall finally say that $\langle \delta : \triangledown : \alpha \rangle$ **identifies** β if there is a substitution σ such that $\alpha \leq \beta^\sigma$.

Second, note that not only will the argument structures be fused. Recall that a sign contains a set of morphs (see Definition 2.21). A morph is a triple $(\mathfrak{g}, \mathcal{A}, \rho)$, consisting of a glued string $\mathfrak{g}$, a sequence of selectors $\mathcal{A}$ and a rank ρ. The selectors are in one-to-one-correspondence with the items in a AIS. Thus, when the variable $\mathfrak{x}$ is bound to an argument structure, we also bind some (hidden) variable to the vector of selectors $\mathcal{A}$. This has the following consequence. In the items for /and/ and /or/ in (**??**) the variable $\mathfrak{x}$ actually occurs three times. When the first merge is performed, it is instantiated. Likewise the hidden variable is instantiated to $\mathcal{A}$, the vector of selectors of the first argument. This is passed to the second argument. The vector of selectors of the second argument is already fixed and must equal that of the first. As a consequence, the morphological handlers of the coordinated expressions are the same as that of its constituent expressions. (The ranks do not need to be identical; as the coordinated expression is nonempty, rank considerations play no role.) In French, for example, most adjectives follow their nouns while some may precede them. However, it is not possible to coordinate two adjectives, where one precedes the noun and the other follows it.

In this analysis of coordination, a phrase /X⌢and⌢Y/ will always have the structure [X [and Y]]. This is intended. The reason for this is that whether or not we assume G-access, the argument structure of Y will be the last element of the argument structure of /and/ that matches.

We have ignored tense in this analysis. Notice however that the present analysis (with or without taking into account the tenses) does not produce the proper reading for the phrase. For it is clear that the phrase /venit et vidit/ just as

in the English equivalent (/He came and he saw./) /and/ is not a logical functor. For what the phrase says is that there was an event of coming and there was another event of seeing. Moreover, there is a natural expectation that the second event is after the first, so that /and/ can be substituted by /and then/. We shall briefly return to the question of a natural interpretation for these things below.

There is an additional meaning of /and/ that is often not distinguished properly from the logical meaning. This is the group forming meaning of /and/. The phrase /John and Mary/ does not denote a conjunction in any sense of the word, at least if we wish to maintain the view that /John/ and /Mary/ denote individuals. Rather, and this will be the line that we shall take here, /John and Mary/ denotes a group, consisting of both John and Mary. The group forming /and/ is syntactically far more restricted than the logical one. It takes two things of the same kind and forms a group. We shall confine ourselves here to the use where it takes two individuals and forms a group. Its argument structure is the following:

$$\begin{array}{|l|}\multicolumn{1}{c}{\texttt{/and/}\bigcirc, \oslash, \oslash} \\ \hline \langle x : \triangle : \begin{bmatrix} \textsc{num}{:}pl \\ \textsc{cat}\ {:}ob \end{bmatrix} \rangle, \\ \langle y : \triangle : \begin{bmatrix} \textsc{num}{:}sg \\ \textsc{cat}\ {:}ob \end{bmatrix} \rangle, \\ \langle z : \triangle : \begin{bmatrix} \textsc{num}{:}sg \\ \textsc{cat}\ {:}ob \end{bmatrix} \rangle. \\ \hline x, x, z \\ \hline x \doteq \{y, z\} \\ \hline \end{array} \qquad (4.115)$$

The uses where it combines an individual and a group or a group and a group are

as follows:

(4.116)

$$\begin{array}{l} /\mathsf{and}/\bigcirc, \oslash, \oslash \\ \hline \langle x : \vartriangle : \begin{bmatrix} \textsc{num}: pl \\ \textsc{cat}: ob \end{bmatrix} \rangle, \\ \langle y : \vartriangle : \begin{bmatrix} \textsc{num}: sg \\ \textsc{cat}: ob \end{bmatrix} \rangle, \\ \langle z : \vartriangle : \begin{bmatrix} \textsc{num}: pl \\ \textsc{cat}: ob \end{bmatrix} \rangle. \\ \hline x, y, z \\ \hline x \doteq \{y\} \cup z \end{array} \qquad \begin{array}{l} /\mathsf{and}/\bigcirc, \oslash, \oslash \\ \hline \langle x : \vartriangle : \begin{bmatrix} \textsc{num}: sg \\ \textsc{cat}: ob \end{bmatrix} \rangle, \\ \langle y : \vartriangle : \begin{bmatrix} \textsc{num}: sg \\ \textsc{cat}: ob \end{bmatrix} \rangle, \\ \langle z : \vartriangle : \begin{bmatrix} \textsc{num}: sg \\ \textsc{cat}: ob \end{bmatrix} \rangle. \\ \hline x \\ \hline x \doteq y \cup \{z\} \end{array}$$

$$\begin{array}{l} /\mathsf{and}/\bigcirc, \oslash, \oslash \\ \hline \langle x : \vartriangle : \begin{bmatrix} \textsc{num}: sg \\ \textsc{cat}: ob \end{bmatrix} \rangle, \\ \langle y : \vartriangle : \begin{bmatrix} \textsc{num}: sg \\ \textsc{cat}: ob \end{bmatrix} \rangle, \\ \langle z : \vartriangle : \begin{bmatrix} \textsc{num}: sg \\ \textsc{cat}: ob \end{bmatrix} \rangle, \\ \hline x \\ \hline x \doteq y \cup z \end{array}$$

Obviously, in a language which distinguishes also a dual from a plural there are many more individual cases to be distinguished. They can be integrated into a single meaning, but it is worthwile pointing out that our present approach explains the fact that when /and/ is used in the group forming sense the agreement in number (and gender/class and other features) are determined by certain rules taking into account the features of both NPs, while the logical /and/ requires them to be the same and outputs the same argument structure. So, two verbs with singular agreement coordinated by logical /and/ still take a singular subject, while two singular subjects coordinated by group forming /and/ trigger plural agreement on the verb! It is actually no accident that x appears in the head section of the DRSs. Group forming /and/ may in fact not only be used to coordinate two DPs that denote individuals or groups but can be used with events, places and many other things. (It will follow from the analysis of Chapter 5.4 that the individual or group is formed at the moment the DP is complete, and this will take care of the restriction that group forming /and/ can coordinate only DPs and yields a group.)

What is needed to properly implement the above proposal is to implement a

distinction between individuals and groups. Although from a purely ontological point of view groups are individuals (witness the fact that a subject of the kind /a group of tourists/ triggers singular agreement), syntax operates on things differently depending on whether it analyzes them as individuals or groups. This means that one and the same thing may at one point be considered an individual and at the next moment a group. As this makes little difference in the actual semantics for the reasons discussed, we shall take it that there exists a feature which decides whether or not something is a group. We may actually take this feature to be [NUM : *pl*] versus (NUM : *sg*]. So, if a referent carries the feature [NUM : *pl*] it will act group like and if it carries the feature [NUM : *sg*] it will act individual like. (This points to the way in which the four meanings of /and/ can be unified.)

Notes on this section. The mechanism has not been implemented in the program. We shall return to the analysis of quantification and numerals in Chapter 5. The use of variables for argument structures is necessary in order to account for type polymorphism. However, there does not seem to be a need to use variables for the purpose, it would be enough to indicate that the AIS imports the entire argument structure. However, the use of variables makes matters clearer. The use of variables for DRSs is not strictly needed, as I have indicated above. If we replaced every possible DRS by an equation $p \doteq \delta$ then the handling of negation and disjunction is actually possible. The disadvantage of the latter approach is that uses too much notation for the effect that it achieves. I have chosen not to explore that variant.

Exercise 40. Try to account for /neither$\cdots$nor/, and /if$\cdots$then/. *Hint.* You may need to make use of discontinuity.

4.6 Implementation Issues

Access control is done in a special way. The so-called diacritics are more complex in the implementation (dia.ml) than shown in this chapter. Diacritics consist of diacritic marks, which are boolean switches, among which the uparrow (u) and downarrow (d) from Section 3.4. There are three more.

- Noskip (n). The AIS where this is set cannot be skipped. Thus, when i_k has

been established and α_{i_k-1} has a noskip feature, then i_{k+1} must be $i_k - 1$.

- Fusion (`f`). If β is not saturated, fusion will fail unless the leader has the fusion diacritic set. (Thus, by not setting the fusion diacritic we disallow that the argument structure can fuse through its AIS.)

- Transformer (`t`). If this is set, the two AVSs in an AIS are allowed to be different.

These switches ensure that the result of merge (fusion) is unique and can be fine-tuned in the lexicon.

There are in addition two global options, `access` (string valued) and `strict` (boolean), defined in `options.ml`. The access option is set to either `G` (default) or `E`. The default for `strict` is `false`. Further, `noskip` should only be set when the `strict` option is off. Otherwise it is without effect.

It is the function `Sargs.merge_list` that computes the pairing function. Notice that a pairing list is an argument to a host of other functions. This is because the list is determined once, and then used to compute not only the argument structure but also the morphology. There is for example also a function cleaning up an argument structure by removing empty AISs. A useful tool to check the working of merge/fusion is the function `diagnose`, called from the top-level system. It can be called to take a detailed look at how the merge is computed and gives a detailed tour through the algorithm.

The implementation is done with a slightly different data structure. It consists of so-called *short argument identification statements* and of so-called *occurrence vectors*. (See the file `sargs.ml`.) A short AIS consists of a diacritic and a pair of AVSs. Thus, the variable has been eliminated from the AIS (this applies likewise to parameters, to be discussed in the next chapter). This has the advantage that the variable names indeed become totally inessential, both in parsing and generating. They are dealt with only in the semantics. In effect, the procedure for parsing is as follows: first, the parser calculates a parse term on the basis of the given string. In a second step, that parse term is then expanded to a structure. This can be done because semantic merge is independent of the morphology. If a term can be found, its semantic value does not depend on anything but the semantic values of the basic parts.

Now, entries e consist in addition to their argument structure of a part that contains a set of pairs $(\mathfrak{m}, T)$, where $\mathfrak{m}$ is a morph and T a set of parse terms. Each member of T unfolds exactly to $\mathfrak{m}$, while their semantics and argument structure is that of e (modulo renaming of variables, as always).

However, when a user requests an entry to be shown, it is shown in the standard way: first the exponent is shown (fractured glued string), together with the morphology. Each exponent is given a number, which is repeated in the bottom part, where parse terms are given for the morph with the given number. Next, the argument structure is shown: for each AIS in turn the variable, the diacritic, the pair of AVS and the parameter section. Below that the DRS is shown. Arguments are coloured, so the arguments can be tracked between the argument structure and the morphology.

Chapter 5

Parameter

In this chapter we shall introduce *parameter* into the referent systems. The mechanics of parameter is distinct from that of ordinary variables. Parameter are the kind of variables that are always present even when they are not needed. Prototypical examples are time points. Many nouns are sensitive to time points, while many are not. However, unlike verbs, the dependency on time has no (or next to no) syntactic or morphological relevance. The omnipresence of parameter offers the possibility to let structures choose freely their set of parameter on which they depend without changing their combinatorial possibilities.

5.1 Properties

In this chapter we shall introduce a new construct, that of a *parameter*. This will then complete our exposition of argument structures. The next chapter will then show a completely worked out example, featuring the basic sentence of Latin. The focus will be on the verbal paradigm, which is complex enough to demonstrate a few major points.

The basic motivation behind parameters is the fact that merge on the one hand can only identify a single pair of variables, while on the other hand there are so many other variables that need to be manipulated as well. The first example that

comes to mind here is time points. Time is not only relevant in the verb phrase; we also find that attributions can be time dependent as well. Recall here the difference between individual level properties like /man/ and stage level properties such as /student/. The same individual can be a student now, and cease to be student, but he can hardly cease to be a man. In the following sentence we must therefore make up our minds as to what time it is that we wish to claim the individual to have been a student: is it now or is it the time when this person was still at school (but presumably not an assistant)?

(5.1) `At school, the assistant has not been good at maths.`

Obviously, there are rules to this, and we shall look at some of them in detail. However, the basic question to ask is what mechanism can account for this behaviour.

This is where parameter come in. The first example of a parameter is however an unlikely candidate, namely *properties*. The behaviour of parameters will be motivated by studying properties for the reason that properties are initially somewhat simpler than time points. However, we shall also show how parameters can be used to model tense and aspect.

Properties are a sort we have not discussed before. There are mainly three reasons for introducing properties. One is that there are plenty of facts showing that properties are distinct from objects or any other kind of entity that we have introduced so far. The other reason is that the mechanism for the assignment of meaning to inflectional morphemes has various technical disadvantages that can only be solved if we assume the existence of properties. The third reason is that because we have no direct mechanism for abstraction, if we want to form properties in the semantics we must actually assume that they exist beforehand. We shall address these questions in turn.

We have seen so far that there are objects in the form of entities and groups, and that there are events. But there is plenty of evidence that we must assume also properties. Here are some constructions in which an NP or an adjective actually denotes a property.

(5.2) `John is a wizard.`

(5.3) `John is clever.`

(5.4) `People call John a fool.`

(5.5) `Paul eats the meat raw.`

In (5.2), the property of being a wizard is attributed to John. In (5.3) it is the property of being clever. (5.4) says that people think that John has the property of being a fool. Finally, (5.5) says that Paul is eating the meat and the meat is in a raw state, that is, having the property of rawness. This motivates the addition of properties into our ontology. However, we must always ask whether the addition of a new sort is really necessary. Perhaps it is possible to construe these examples with objects in place of properties. For example, we might say that (5.2) equates John with a person who is a wizard. Perhaps in this example this is feasible. More likely, (5.2) is not the best way of expressing this. We would rather use (5.6) or (5.7) where we refer to an already established individual.

(5.6) `John is the wizard.`

(5.7) `John is one of the wizards.`

In both examples there is an individual or group introduced in postcopular position and it is said that John is that individual, as in (5.6), or is part of the group, as in (5.7). We claim that no such reading exists for (5.2). This sentence simply attributes a property to John. The same holds for the other constructions.

One difference between objects and properties is that objects can be used with a demonstrative while properties cannot.

(5.8) *`They call John this fool.`

This leaves us with the picture that NPs may alternatively denote properties and objects. This is indeed the case. Evidence for this view comes from Hungarian. As Farkas and Swart 2003 argue, Hungarian shows what is referred to as abstract incorporation.

(5.9) `Az orvos a beteget vizsgált.`
‘The doctor examined the patient.’

(5.10) `Az orvos beteget vizsgált.`
‘The doctor examined patients.’

The difference between (5.9) and (5.10) is among other things the unavailability in (5.10) of backward anaphoric reference to the object. While Farkas and de Swart analyse this as a lack of discourse markers, we propose here to view the object as denoting a property (notice the absence of the determiner). Crucially, properties cannot be referred to using pronouns. Thus, by distinguishing between

a property and an object we introduce the possibility of blocking reference to the kind if necessary.

From a semantic point of view there are much stronger arguments in favour of properties, however. We must have them, be they sorts in their own right (as in property theory) or simply individual concepts. Otherwise the semantics of non-intersective adjectives must remain mysterious. Non-intersective adjectives are such adjectives that modify a property rather than attributing a property to an object. An example is /`good`/. A good teacher might not be a good gardener, since he or she might just be good at teaching. Therefore, in order to say that someone is a good teacher it is not to say that he is good and a teacher, rather, he is good at teaching. Similarly for /`big`/ or /`tall`/. A big mouse is not of the same size as a big elephant. In order to know whether something is big you need to know in what respects it is big. Something can be a small mammal but a big mouse. In these examples it is patently clear that the adjective cannot simply take the object and predicate a property of it, as we assumed previously. Rather, the adjective must know what property was said to hold of that object. Therefore, the property must be explicitly represented. The same holds by the way also for the plural, but we shall discuss number later.

The second reason we shall adduce here in defense of properties is the problem of the association of meaning to morphemes. So far we have assumed that each and every morpheme has a meaning. This applies, for example, to the plural morpheme. So whenever the plural morpheme is attached to some stem it forms a group of things satisfying that property. But exactly this cannot work. Consider the following example from Latin:

	`quattuor`	`magni`	`mures`
(5.11)	four	big-MASC.PL.NOM	mouse-PL.NOM
	'four big mice'		

According to our previous analysis, this would be a group of mice which is in addition a group of big things and a group consisting of four elements. But we cannot construe the adjective like that. Without knowing what property is considered we cannot know whether the right kind of group is being formed. The mice, being mammals, are small mammals. So any group of four mice is a group of four mice which are small mammals. But not every group of four mice which are small mammals will qualify for a group of four big mice. It is therefore useless to ask whether the group consists of big things. Given the group, it may consist

of big things when looked at from one perspective (mice), and of small things when looked at it from another side (mammals). Hence, our previous proposal is doomed to failure with respect to non-intersective adjectives.

What can be done? We shall assume that the numeral, in this case /quattuor/, actually forms the group from the property. Before it does so there simply is no group, just a property. So, we consider both nouns and adjectives as denoting properties. (Sometimes even noun phrases denote properties as we have seen in example (5.2).) We shall take it that the numeral forms the group. This has an immediate consequence. Namely, after the group has been formed it is opaque for non-intersective adjectives. This seems to be incorrect, but we shall hold onto it for the moment. Consider by way of counterexample the sentences (5.12) and (5.13).

(5.12) `This teacher is good.`

(5.13) `This mouse is big.`

(5.12) says that a certain individual is good as far as his abilities as a teacher are concerned. (5.13) says that the individual is big for a mouse. We shall leave the matter at that, however.

There is a third point that deserves mentioning. We shall assume throughout that there is no mechanism for abstraction. Consequently, there is no way of obtainiing a property from an individual or a group. Thus, if we assume that a noun, say /mouse/, only denotes certain objects there is no expression that is formed from it which denotes some property. To get this property one needs abstraction. Recall namely that the word mouse has been given the meaning mouse′(x), which is an open formula. Now, in order to obtain a property from that formula we need internal devices to abstract over a variable. We shall assume, however, that there is no such mechanism. The reason is twofold. First, we assume that language does in fact not use abstraction (at least not in the form of λ-calculus), but prefers to talk rather concretely, that is, using objects whenever possible. The second is that we do not wish to introduce λ-calculus through the back door, since that would make the system rather costly (in terms of processing requirements).

5.2 The Mechanics of Parameters

The way language deals with properties is rather complex. Intuitively, properties are not things, and language prefers things over properties, and it likes to talk about properties in terms of things — where by things I mean mainly humans, animals, and concrete objects. The more abstract the less likely language knows how to deal with it in its own systematic terms. The argument structure uses the morphological features and these are almost universally geared towards things (see Corbette 2012). As a consequence, the agreement mechanism, primarily invented to classify concrete objects, is used for all kinds of denotations, be they groups or abstract entities like properties. One can observe for example that the gender system of Indo-European is an obscure mixture of semantics and morphology. The motivating terminology is the distinction between animate and inanimate, and between male and female among the animate. By definition, any abstract entity must therefore be neuter. However, this is very often not the case. Instead, the system is arbitrarily extended; moreover, some randomness is introduced through mere historic accident (see Corbett 1991 for an extensive discussion of gender in language).

Moreover, looking at agreement systems we find that independently of what the adjective actually denotes, the agreement features are determined by the noun phrase denotation, if anything. For it is at the level of noun phrase where the expression actually denotes an object; before that it simply denotes a property of some kind. In order to account for that we shall assume that properties are actually not classified independently; rather, a property is assigned a variable of an object whose classification features it uses. Whether or not that object variable actually occurs in the semantics will be irrelevant. One may think of this object variable as an object that is in the process of being made. Moreover, there is a fundamental difference between objects and properties. Namely, the meaning of an adjective is usually not a property but rather a function modifying a property. An adjective consumes a property, say q, and returns another property, $p = N(q)$. For example, the meaning of /big/ is $N = \lambda q.\lambda x.\mathsf{big}'(q)(x)$, where q is a variable for a property. Thus, N asks for a property (here q) and returns a property, which, given an individual x attributes to x the property of being big with respect to being q, that is, being a big q-er. So, adjectives modify the property that is attributed to the object. This is in stark contrast to the way the system was assumed to work. The agreement within the noun phrase was made possible through the coherence

of the objects that are being used within the argument structure. Since the noun and all the other adjectives were attributes of the same object they showed agreement in virtue of being predicates of the same object. When the object is gone, the coherence is lost. We shall have to look for it elsewhere. The idea that saves us from loosing coherence is the notion of a *parameter*.

Before we explain the mechanics of the device that handles the adjectives we shall say that the idea of certain things changing through the structure is actually quite pervasive, a theme that has been developed in dynamic semantics, e. g. Groenendijk and Stokhof 1991. For example, time is constantly being reset, not only from one sentence to another. Properties also depend on time; for example, being a prime minister or a director is a time dependent property and language has means to keep track of the time at which a property applies to which object. Similarly, worlds or situations can be reset. When we talk about fictitious things it is not assumed that they exist in this world. Again, there are controlled ways to track the current value of worlds or situations. Last but not least the coordinates speaker/hearer can be reset in a text. We call all these things parameters. (In the literature they are also referred to as indices.) The idea that we shall develop is that while hand shake of referent systems is brought about by sharing an object variable, this handshake can also bring about a sharing of parameters. In order to do this, the parameter is associated with a particular variable. When the variable is shared, so is the parameter associated with that variable. To see how this works we shall outline the semantics of an adjective. This means that we shall study the mechanism of a single parameter. Later we shall be concerned with additional parameters. We shall annotate the name of the referent with a letter, choosing p, p' and q for properties. The parameter is separated from the name (or name change statement) by a double colon (::).

(5.14)

`/big/`⊗
$\langle x : \Diamond : \nu :: p \mapsto p' \rangle$
x, p, p'
$p' \doteq \mathsf{big}'(p).$

So, the parameter of a property is added after the name. Notice that the name may change as well as the parameter. Since the parameter is associated with the name, the value of a parameter can only be reset through passing on the object. This can be seen with a noun. Nouns do not modify a property, hence they only instantiate the parameter.

Example 25. The lexical entry for a nonrelational noun in English will now take the following shape.

$$\text{(5.15)}\quad \begin{array}{|c|} \multicolumn{1}{c}{\texttt{/mouse/}\bigcirc} \\ \hline \langle x : \triangle : \begin{bmatrix} \text{PERS} & : & 3 \\ \text{CLASS} & : & \mathit{neut} \\ \text{NUM} & : & \mathit{sing} \end{bmatrix} :: p \rangle \\ \hline x, p \\ \hline p \doteq \mathsf{mouse}'. \\ \hline \end{array}$$

So, the noun denotes a property, which however is not the main variable; it is a parameter. As before we shall assume that x comes out of the lexicon with certain features being instantiated. It is a morphological requirement to fill some of the remaining features by means of inflectional morphemes. (See also the next chapter.)

This has the following consequence: there is no value for case, indeed as there is no inflection for case, we may just leave that out. The structure in (5.14) will now be spelled out in more detail like this:

$$\text{(5.16)}\quad \begin{array}{|c|} \multicolumn{1}{c}{\texttt{/big/}\oslash} \\ \hline \langle x : \Diamond : \begin{bmatrix} \text{PERS} & : & \mathit{3} \\ \text{CLASS} & : & \top \\ \text{NUM} & : & \top \end{bmatrix} :: q \mapsto q' \rangle \\ \hline x, q, q' \\ \hline q' \doteq \mathsf{big}'(q). \\ \hline \end{array}$$

The idea is that the adjective can combine with a noun (which is 3rd person only) which can in principle be of any gender and number. ⚽

We need to explain how merge deals with parameters. All the other elements function as before. The leading idea is that merge of AISs drags along the paremeters contained in the relevant AISs. They became merged, too, though in a slightly different way. In the example above the variable x is identified by the adjective to its right. When it combines with a noun with variable y that has the same name the merge succeeds, and x of the adjective and y of the noun become shared. Now, x in the adjective has a property parameter p and y in the noun has a property parameter q. In virtue of the variables being shared, the parameters will be shared

as well. So, as a result of combining (5.16) and (5.15) we get

(5.17)

$$\begin{array}{c}\texttt{/big mouse/}\bigcirc\\ \hline \multicolumn{1}{|c|}{\langle x : \vartriangle : \left[\begin{array}{lcl}\text{PERS} & : & \mathit{3}\\ \text{CLASS} & : & \mathit{neut}\\ \text{NUM} & : & \mathit{sing}\end{array}\right] :: q'\rangle}\\ \hline \multicolumn{1}{|c|}{q, q', x}\\ \hline \multicolumn{1}{|c|}{q \doteq \mathsf{mouse}';}\\ \multicolumn{1}{|c|}{q' \doteq \mathsf{big}'(q).}\\ \hline\end{array}$$

The parameter p is being shared resulting in the following semantics: /`big mouse`/ denotes a property of being a big q-er, where q is the property of being a mouse. One can eliminate the occurrence of q, due to logical equivalence, and obtain the equivalent structure

(5.18)

$$\begin{array}{c}\texttt{/big mouse/}\bigcirc\\ \hline \multicolumn{1}{|c|}{\langle x : \vartriangle : \left[\begin{array}{lcl}\text{PERS} & : & \mathit{3}\\ \text{CLASS} & : & \mathit{neut}\\ \text{NUM} & : & \mathit{sing}\end{array}\right] :: q'\rangle}\\ \hline \multicolumn{1}{|c|}{q', x}\\ \hline \multicolumn{1}{|c|}{q' \doteq \mathsf{big}'(\mathsf{mouse}').}\\ \hline\end{array}$$

When there are several parameters, it must be made clear what kinds of parameters there are, and it must be assumed that there is of every kind only one parameter. Those parameters that are not explicitly mentioned but are provided by the argument will be passed on unchanged.

The semantics of intersective adjectives is now a little bit more complex, as shall have to keep track of the property parameter. If red' is a property of individuals (and not a property of properties), the adjective /`red`/ now has the following semantic structure.

(5.19)

$$\begin{array}{c}\texttt{/red/}\ominus\\ \hline \multicolumn{1}{|c|}{\langle x : \lozenge : \nu :: p \mapsto p'\rangle}\\ \hline \multicolumn{1}{|c|}{p, p', x}\\ \hline \multicolumn{1}{|c|}{p' \doteq \lambda x.\mathsf{red}'(x) \wedge p(x)}\\ \hline\end{array}$$

Notice that there is a fair number of adjectives that are used both non-intersectively and intersectively. An example is /`big`/. On the one hand, whether something

is big or not depends on what kind of object it is, on the other hand there is also an absolute notion of what a big object is. This may affect the range of syntactic constructions in which an adjective can appear. Typically, when used in postcopular position an adjective either has to be intersective or a property must be inferred.

(5.20) `This mouse is brown.`

(5.21) `This mouse is big.`

(5.22) `It is big.`

In (5.20) we may say that the object under consideration, a specific mouse, is brown. Assuming that /`brown`/ is an intersective adjective, this is the most unproblematic usage of the adjective. In (5.21) we are left with two choices. We may consider the adjective /`big`/ as being used in an absolute sense, in which case we really have a really big mouse being talked about, or it is used not in an absolute sense, and then a property must be inferred from the context. Presumably in this example the object under consideration is big in its property of being a mouse. Notice that the adjective cannot be used non-absolutely in (5.22) unless the property in question is contextually given.

Let us now look at other parameters. The most pervasive parameter is *time*. Many words denote time dependent entities. We have for example two kinds of properties: so-called *individual level* and *stage level* properties. Properties of the first kind are 'man', 'tiger', properties of the second kind are 'prime minister' or 'director'. Someone may be a prime minister and one time point, and a revolutionary at another. Something may be red at some moment and green at another. So, in order to fully account for truth conditions even for NPs, we need time points in addition to properties as parameters. However, a problem needs to be solved right at the beginning: how do we manage the handling of two parameters? How do the parameters know which is which?

Our solution is as follows. Parameters are identified by some attribute. For example, the property parameter is the value of some attribute PROP, while the time parameter is the value of another attribute, PRED (short for predication time).

Here is an example.

(5.23)

$$
\begin{array}{c}
\texttt{/president/}\bigcirc \\
\hline
\left\langle x : \triangle : \left[\begin{array}{lcl} \textsc{cat} & : & n \\ \textsc{pers} & : & 3 \\ \textsc{class} & : & masc \\ \textsc{num} & : & \star \end{array}\right] :: \left[\begin{array}{lcl} \textsc{prop} & : & p \\ \textsc{pred} & : & t \end{array}\right] \right\rangle \\
\hline
x, p, t \\
\hline
p \doteq \mathsf{president}'(t)
\end{array}
$$

We have used the same attribute value notation for the parameters. Here PRED is the attribute for the predication time. However, the values are variables, in this case t. The constant $\mathsf{president}'$ is a function from time point to properties of individuals. Likewise, an adjective can be time dependent:

(5.24)

$$
\begin{array}{c}
\texttt{/big/}⧁ \\
\hline
\left\langle x : \Diamond : [\textsc{pers} : 3] :: \left[\begin{array}{lcl} \textsc{prop} & : & p \mapsto p' \\ \textsc{pred} & : & u \mapsto u' \end{array}\right] \right\rangle \\
\hline
x, p, p', u, u' \\
\hline
p' \doteq \mathsf{big}'(u)(p); u' \doteq u
\end{array}
$$

Merge must identify p of (5.24) with p in (5.5), and u in (5.6) with t in (5.5). The handshake between parameters is triggered by the identity in name (PROP and PRED, respectively). Notice that the adjective allows for each parameter name to have two values. In a functor, the left hand variable is the variable to be shared with its argument (if it has a parameter of the same name), while the right hand variable is the variable to be exported under that name. We do not require them to be different, so that (5.24) can be simplified to

(5.25)

$$
\begin{array}{c}
\texttt{/big/}⧁ \\
\hline
\left\langle x : \Diamond : [\textsc{pers} : 3] :: \left[\begin{array}{lcl} \textsc{prop} & : & p \mapsto p' \\ \textsc{pred} & : & u \end{array}\right] \right\rangle \\
\hline
p, p', u \\
\hline
p' \doteq \mathsf{big}'(u)(p).
\end{array}
$$

Here, we have written [PRED : u] in place of PRED : $u \mapsto u$], with u both on the left hand side and on the right hand side.

Definition 5.1 (Parameter Handling Statement) *Let P be a set of **parameter names**. A **parameter handling statement** (PHS) over P is either a partial function*

from P and values from the set of referents; we call such types ***simplex****. Or it is a partial function from P with values being pairs of referents. We call such PHSs* ***duplex****.*

We write the PHS in AVM-style notation; the argument appears to the left, the value to the right. Thus we write $[P : x \mapsto y]$ in place of $[P : \langle x, y\rangle]$, which in turn means that $f(P) = \langle x, y\rangle$. PHSs are quite different from AVSs: the values of attributes (here called parameter names) are referents, and there are infinitely many of them. Only the set of parameter names is assumed to be finite. Even though we speak of "parameters" and "referents" there is no difference between the two. The variables are allowed to change from being used in the PHS to being the variable of an AIS and back. For each sort of variable (thing, person, time, world, location) we will actually have several distinct parameter names for variables of that sort (as we did just now for time), but there seems to be an upper bound of four for each.

It is not required that the values for the parameter names are distinct. A referent can appear in as many places as it likes.

Definition 5.2 (Parametrised Argument Identification Statement) *A* ***parametrised argument identification statement*** *(PAIS) is a pair consisting of an AIS $\langle x : \partial : N\rangle$ and a PHS $\mathcal{P}$, written $\langle x : \partial : N :: \mathcal{P}\rangle$, such that (a) if $\partial = -$ then $\mathcal{P}$ is empty, (b) if $\partial = \triangledown$ or $\vartriangle$ then $\mathcal{P}$ is simplex, and (c) if $\partial = \lozenge$ then $\mathcal{P}$ is duplex. We say that $\langle x : \partial : N :: \mathcal{P}\rangle$* ***imports*** *x* ***as*** *P if either (i) $\partial = \triangledown$ and $[P : x] \in \mathcal{P}$ or (ii) $\partial = \lozenge$ and $[P : \langle x, y\rangle] \in \mathcal{P}$ for some y. We say that $\langle x : \partial : N :: \mathcal{P}\rangle$* ***exports*** *x* ***as*** *P if either (i) $\partial = \vartriangle$ and $[P : x] \in \mathcal{P}$ or (ii) $\partial = \lozenge$ and $[P : \langle y, x\rangle] \in \mathcal{P}$ for some y.*

We shall actually assume that what we previously called AISs are in fact PAISs, and the definitions of argument structure, merge and fusion will have to be lifted to the type of structure. Most of this actually goes without changing anything.

I introduce one more piece of notation. I write $\circ$ for a parameter in a PAIS just in case its identity is irrelevant ("anonymous variable"). This is particularly useful when the parameter is not used in the semantics. The notation $\circ$ stands for a variable that occurs only once in the entire structure, namely at the given occurrence of $\circ$. Thus, each occurs of $\circ$ stands for a different variables. This is like writing "_" in certain programming languages.

Recall that the referents of the left hand representation are indexed by 1, and

the referents of the right hand representation are indexed by 2. In the phase of merge, unification of certain referents takes place. Unification happens if two AISs are merged. The rule is in general the following.

> Suppose x is the variable of the functor PAIS imported under a name A. Suppose that y is the variable of the argument PAIS exported under the name A. Then after merge the substitution $[x^1/y^2]$ is being applied.

We illustrate the mechanics with the following merge.

(5.26)
$$\begin{aligned} & \langle x : \Diamond : N :: \begin{bmatrix} \mathrm{P} : p_1 \mapsto p_4 \\ \mathrm{Q} : p_2 \mapsto p_2 \\ \mathrm{R} : p_3 \mapsto p_5 \end{bmatrix} \rangle \bullet \langle y : \vartriangle : N :: \begin{bmatrix} \mathrm{P} : q_1 \\ \mathrm{Q} : q_2 \\ \mathrm{S} : q_3 \end{bmatrix} \rangle \\ = \ & \langle x^1 : \vartriangle : N :: \begin{bmatrix} \mathrm{P} : p_4^1 \\ \mathrm{Q} : p_2^1 \\ \mathrm{R} : q_3^2 \\ \mathrm{S} : p_5^1 \end{bmatrix} \rangle \end{aligned}$$

(The identity of the AVM is unimportant, In place of the name N we may put an AVM and compute as usual.) First, as usual all variables to the left get the superscript ^{1}, all variables to the right get the superscript ^{2}.

1. Apply the substitution $[p_1^1/q_1^2]$. This is because the imported variable of the functor, p_1, has the same name as the exported variable of the argument q_1, namely P.
2. Apply the substitution $[p_2^1/q_2^2]$. This is because the imported variable of the functor, q_2, has the same name as the exported variable of the argument, p_2, namely Q.

The rules of merge are that in these cases the respective referents are to be considered the same.

The rule is this: suppose that the rightward merge succeeds. Then for every parameter name P the parameter that is exported by the righthard AIS under the name P is identified with the parameter that is imported but the leftmost AIS under the same name.

Notes on this section. In the implementation, $\circ$ is simply treated as an empty name. Indeed, the algorithm never looks at the identity of $\circ$, which is rendered by an object of type `Nil`. To see that this works, let us rehearse the definition of merge above. First, no superscript is added to $\circ$. In merge we have

1. $[P : \circ \mapsto v] \bullet [P : u \mapsto \circ] = [P : u^2 \mapsto v^1]$;
2. $[P : \circ \mapsto v] \bullet [P : u \mapsto x] = [P : u^2 \mapsto v^1]$;
3. $[P : y \mapsto v] \bullet [P : u \mapsto \circ] = [P : u^2 \mapsto v^1]$;
4. $[P : y \mapsto v] \bullet [P : u \mapsto x] = [P : u^2 \mapsto v^1]$;

In place of u or v, $\circ$ may appear as well. Since $\circ$ has no occurrence in the semantics, no substitution is needed to be applied on $\circ$. A proper proof of these facts consists in taking in place of each occurrence of $\circ$ a fresh variable and then running the above algorithm. Any variable occurring just once and in a PAVS can be translated (optionally) into $\circ$.

Exercise 41. We have seen in Exercise 24 that $\langle x : \Diamond : M \mapsto N\rangle$ can be replaced up to congruence by $\langle x : \vartriangle : N\rangle, \langle x : \triangledown : M\rangle$, where M and N are names. Also, Exercise 25 dealt with the elimination of $\Diamond$ in the presence of underspecification and Copy AVMs. We shall now extend this result to parameters. Fix some $n \in P$, P the set of parameter names. Now show that $\langle x : \Diamond : M \mapsto N :: [n : u \mapsto v]\rangle$ can be replaced up to congruence by $\langle x : \vartriangle : N :: [n : v]\rangle, \langle x : \triangledown : M :: [n : u]\rangle$. How is this extended to Copy AVMs?

Exercise 42. Here is an exercise to show that we can get rid of parameters in favour of polyadic merge. To eliminate the parameters, introduce a new attribute, PARAM, with values from the parameter name set $P = \{n_0, \cdots, n_{k-1}\}$. The AIS

$$\langle x : \Diamond : M \mapsto N :: \begin{bmatrix} n_0 & :u_0 & \mapsto & v_0 \\ n_1 & :u_1 & \mapsto & v_1 \\ & & \cdots & \\ n_{k-1} & :u_{k-1} & \mapsto & v_{k-1} \end{bmatrix}\rangle \tag{5.27}$$

is replaced by the sequence

$$
(5.28)\quad
\begin{array}{l}
\langle x : \vartriangle : N\rangle \\
\langle v_0 : \vartriangle : [\textsc{param} : n_0\rangle, \langle v_1 : \vartriangle : [\textsc{param} : n_1\rangle, \\
\qquad \cdots, \langle v_{k-1} : \vartriangle : [\textsc{param} : n_{k-1}\rangle, \\
\langle u_{k-1} : \vartriangle : [\textsc{param} : n_{k-1}\rangle, \cdots, \\
\qquad \langle u_1 : \triangledown : [\textsc{param} : n_1\rangle, \langle u_0 : \triangledown : [\textsc{param} : n_0\rangle, \\
\langle x : \triangledown : M\rangle
\end{array}
$$

Define analogous replacements for the other diacritics $\vartriangle$ and $\triangledown$. Show that this transformation yields the same set of signs up to congruence.

5.3 Tense and Aspect

Let us take a closer look at time, tense and aspect. This will give us a good insight into the overall potential of parameters.

The terminology and implementation follows Klein 1994. In particular, we shall basically assume that tense and aspect can be reduced to the concurrent handling of three parameters, a proposal that itself has a long history, dating back at least to Reichenbach. These are

- *reference time*, called *deictic center* in Comrie 1985 and *utterance time* (UT) in Klein 1994, for it is mainly coextensive with the time of the utterance (exception in direct quotation);
- *topic time* (Klein 1994);
- *predication time*, called *situation time* (TSit) by Klein.

Actually, while all three of them are best thought of as intervals, I shall reduce them to time points. This eliminates some of the complexity without taking away essential details. It is no problem to make the approach more finegrained by throwing the intervals back in, but they do not help in elucidating the role of parameters. There are three basic relations: $t \doteq t'$ (t is identical to t'), $t < t'$ (t is before t'), $t > t'$ (t is after t').

According to Klein 1994, tenses specify the relation between topic time and reference time, while aspect specifies the relation between predication time and topic time.

There are three basic relations between two time points, and they correspond to the basic tenses found in many languages. They do not, however, represent an exhaustive list of tenses found in the world's languages.

Classification of Tenses

- *past tense*: the topic time is prior to the reference time
- *present tense*: the topic time is identical to the reference time
- *future tense*: the topic time is after the reference time

Similarly, aspect deals with the relation between predication time and topic time.

Classification of Aspect

- *ongoing* or *presentive aspect*: predication time is identical with topic time.
- *perfective aspect*: predication time is prior to topic time.
- *futurate aspect*: predication time is after topic time.

Morphologically, the is a universal pattern of feature installment.

> [TAM sequence constraint] To the verbal root, modality is added first, then aspect and then tense.

This is not in perfect correspondence with the actual sequence in which the morphological markers occur. If they are suffixes, we expect modality markers to be indeed before aspect and aspect before tense. If they are prefixes, we expect the opposite order. However, complications arise if we have a mixture of prefixes and suffixes; and, more commonly, if some features are not expressed as morphological affixes but on a separate root.

Table 5.1: The Complex Tenses

Tense	Aspect		
	ongoing	perfective	futurate
present	$r = t = p$ present	$p < t = r$ perfect	$s = t < p$?
past	$p = t < r$ past	$p < t < r$ pluperfect	$t < r; t < p$ future in the past
future	$r < t = p$ future	$r < t; p < t$ future II	$r < t < p$?

Table 5.2: Latin Tenses / Aspects

present	`amo`	‘I love’
past	`amabam`	‘I loved’
future	`amabo`	‘I will love’
perfect	`amavi`	‘I have loved’
pluperfect	`amaveram`	‘I had loved’
future II	`amavero`	‘I will have loved’

There are in total nine combinations, of which six are common in European languages (French, German, Greek and Latin), though modern variants have greatly reduced the system. The combinations are displayed in Table 5.1. In this table we use r, t and p to denote the reference time, topic time and predication time, respectively. Let us illustrate this with Latin. Basically, the system knows two aspects: ongoing and perfective. The latter are added to the perfective stem, which in turn is formed by adding the suffix /`v`/. We shall return to the morphological issues in the next chapter. At this point, the only thing of interest is that the tense morphology is added *after* the aspect.

Consider a verbal root, written here in small caps, since the actual form is

irrelevant.

$$\begin{array}{c} \texttt{/run/, /ran/}\bigcirc \\ \hline \multicolumn{1}{|c|}{\langle e : \triangle : \left[\begin{array}{ll} \text{CAT} & : e \\ \text{TENSE} & : \star \\ \text{ASP} & : \star \end{array}\right] :: [\quad] \rangle} \\ \hline \multicolumn{1}{|c|}{e} \\ \hline \multicolumn{1}{|c|}{\mathsf{run}'(e)} \\ \hline \end{array} \tag{5.29}$$

Aspect markers install the aspect. Here is an example of a marker for ongoing aspect.

$$\begin{array}{c} //\oslash \\ \hline \multicolumn{1}{|c|}{\langle e : \blacklozenge : \left[\begin{array}{l} \text{CAT}: e \\ \text{ASP}: \star \mapsto \mathit{ong} \end{array}\right] :: \left[\begin{array}{lll} \text{TT} & : & t_1 \\ \text{PRED} & : & t_2 \end{array}\right] \rangle} \\ \hline \multicolumn{1}{|c|}{e, t_1, t_2} \\ \hline \multicolumn{1}{|c|}{\mathsf{time}'(e) \doteq t_2; t_2 \doteq t_1.} \\ \hline \end{array} \tag{5.30}$$

The entry for the perfective aspect is as follows.

$$\begin{array}{c} //, \texttt{/ed/}\oslash \\ \hline \multicolumn{1}{|c|}{\langle e : \blacklozenge : \left[\begin{array}{l} \text{CAT}: e \\ \text{ASP}: \star \mapsto \mathit{perf} \end{array}\right] :: \left[\begin{array}{lll} \text{TT} & : & t_1 \\ \text{PRED} & : & t_2 \end{array}\right] \rangle} \\ \hline \multicolumn{1}{|c|}{e, t_1, t_2} \\ \hline \multicolumn{1}{|c|}{\mathsf{time}'(e) \doteq t_2; t_2 < t_1.} \\ \hline \end{array} \tag{5.31}$$

Tense requires aspect to be present. This is coded into an entry for tense by adding the condition [ASP : $\top$] to the AVM.

$$\begin{array}{c} //, \texttt{/ed/}\oslash \\ \hline \multicolumn{1}{|c|}{\langle e : \blacklozenge : \left[\begin{array}{ll} \text{CAT} & : e \\ \text{TENSE} & : \star \mapsto \mathit{past} \\ \text{ASP} & : \top \end{array}\right] :: \left[\begin{array}{lll} \text{TT} & : & t_1 \\ \text{PRED} & : & t_2 \end{array}\right] \rangle} \\ \hline \multicolumn{1}{|c|}{e, t_1, t_2} \\ \hline \multicolumn{1}{|c|}{\mathsf{time}'(e) \doteq t_2; t_2 \doteq t_1.} \\ \hline \end{array} \tag{5.32}$$

We could have added [TNS : $\star$] to the AVM of the aspect to make sure it cannot be added after tense, but this is redundant.

Complex tenses can be formed either by affixation or by use of auxiliary verbs. In languages which use the latter strategy, this auxiliary may be either `/to be/` or

/to have/. We shall not be concerned with the selection of this auxiliary. What all these languages have in common is that the tenses of the second series are formed by different means than the corresponding simple tenses. The forms of the markers can be sensitive to any feature that is present.

Example 26. The Latin verb /tangere/ 'to touch' has the present active stem /tang/ and a perfect active stem /tetig/. The forms are active, 1st person singular indicative.

(5.33)

tang-ō	tetig-ī
touch.ONGOING-PRES	touch.PERF-PRES
'I touch'	'I have touched'
tang-ēbam	tetig-eram
touch.ONGOING-PAST	touch.PERF-PAST
'I touched'	'I had touched'
tang-am	tetig-erō
touch.ONGOING-FUT	touch.PERF-FUT
'I will touch'	'I will have touched'

This can be accounted for in the following way. We make the markers of present, past or future sensitive to whether the word they apply to has [ASP : *ongoing*] or whether it has [ASP : *perf*]. So, the perfect stem itself already encodes the notion of the event being completed (i. e. that the event time precedes topic time), while the present stem signals contemporaneity. The tense suffix has two forms, depending on whether it attaches to the simple stem or the perfect stem, and we may therefore say that the tense suffix agrees with the stem in aspect. ⚽

In the same way we can set up the tense systems of German, English and Finnish, which all use an auxiliary. We shall say that the auxiliary carries the tense and it applies only to a carrier of aspect.

As a special case of agreement we note an example reported in Comrie 1985 going back to Randriamasimanana 1981. In Malagassy, certain adverbs must agree with the main verb. The word for *here* is /ao/ in the present but /tao/ in the past.

(5.34) n-ianatra t-ao/*ao i Paoly omaly.
PAST-study PAST-here DEF Paul yesterday

Obviously, in these adverbs there is a sensitivity for the tense. This has however nothing to do with the actual parameters, but constitutes agreement in tense.

Example 27. There are also languages in which there exist more distinctions than simply between past, present and future. The following are the tense suffixes of Yandruwandra (see Comrie 1985):

	`na`	very recent past
	`ṅana`	within the last couple of days
(5.35)	`ṅukarra`	within the last few days
	`nga`	weeks or months ago
	`lapurra`	distant past

Here are the tense suffixes of Yagua (see Comrie 1985):

	`jasiy`	proximate-1 (within a few hours)
	`jái`	proximate-2 (one day ago)
(5.36)	`siy`	within a few weeks
	`tíy`	within a few months
	`jadá`	distant or legendary past

What these tenses add in addition to placing one time point with respect to another they also specify the distance between these time points. ⚽

Exercise 43. The morphology of the aspect and tense in English has been rather minimal. Spell out the details.

Exercise 44. Provide the lexicon for the tenses of Yandruwandra and Yagua as given in Example 27. *Hint.* You may need to add extra semantic primitives.

5.4 Time in the Noun Phrase

As discussed above, NPs are often also dependent on time, though they typically do not display this dependency morphologically. Moreover, while the finite verb is in charge of handling three parameters, the NPs only make use of one of them.

It is however not always clear which one that is. In an intransitive clause, we need to monitor at least four parameters. Tense and aspect regulate how the three verbal parameters interact. What is now less clear is how the additional time parameter introduced by the noun phrase is to be linked to the event.

We shall illustrate this with some German examples. Consider the following sentence.

(5.37) `Der Präsident war in seiner Schulzeit ein schlechter Schüler.`

'In his school days the president was a bad student.'

Here, the noun /`Präsident`/ 'president', as it expresses a time dependent property, must hook itself onto some parameter. But which one? As topic time and event time (= the predication time of the verb) coincide, the only significant choice is between reference time (that is, the "now") and topic time. The preferred reading seems to be topic time, as (5.38) shows.

(5.38) `Der Präsident war in seiner Schulzeit ein schlechter Schüler gewesen.`

'In his school days the president had been a bad student.'

In (5.38) the subject's predication time is the topic time. Thus, the person who is president at topic time (prior to "now") is said to have been a bad student at high school, which is topic time. In German the preferred reading can be suppressed by using the adjective /`heutig`/ 'present day'.

(5.39) `Der heutige Präsident war in seiner Schulzeit ein schlechter Schüler.`

'The present day president was in his school days a bad student.'

Here it is clearly the case that the subject's predication time is reference time. To enforce the topic time, the adjective /`damalig`/ 'of that time' may be used. In the case of (5.39) it enforces a predication at topic time. To enforce the topic time, the adjective /`damalig`/ 'of that time' may be used. Pragmatically, this is an odd sentence.

(5.40) ?`Der damalige Präsident war in seiner Schulzeit ein schlechter Schüler.`

'The president of that time was in his school days a bad student.'

However, the predication time of the subject can also be the event time, see (5.41). The same holds for the object.

(5.41) `Im Jahre 1953 hielt der Präsident eine große Rede.`
'In 1953 the president held a big speech.'

(5.42) `Der Präsident`$_1$ `lernte den Minister während seiner`$_1$ `Schulzeit kennen.`
'The president$_1$ got to know the minister during his$_1$ school days.'

In (5.41) the reference time is "now" (hence past tense), but the subject is preferably formed at topic time. In (5.42) either interpretation for the object noun phrase are OK. We may either conceive of the minister as being the one at topic time or the one at event time. (As indicated by the subscripts, we assume here that the pronoun refers back to the president, otherwise the preferences are inverted.) We can disambiguate the sentence by using either /`heutig`/ or /`damalig`/.

(5.43) `Der Präsident lernte den heutigen Minister während seiner Schulzeit kennen.`
'The president got to know the present day minister during his school days.'

(5.44) `Der Präsident lernte den damaligen Minister während seiner Schulzeit kennen.`
'The president got to know the minister of that time during his school days.'

In (5.43), it is the minister at utterance time (= now) that the president got to know during his school days, while in (5.44) is the minister of topic time (= then).

These facts can be accounted for using parameters. First, when talking about time as a parameter, we shall basically assume that all elements share these parameters. If they do not make use of them that will be fine, but they will still pass them on to all other elements. This means that at all levels we shall have to distinguish three time points (or intervals), namely reference time, topic time and event or predication time. This applies equally to nouns and noun phrases. However, the noun phrase needs only one time point. We may now say that this time point simply is the predication time of the noun, and that the noun phrase may decide to pass on this point of time either as the reference time or the topic time. If this is

the analysis, then the NP acts by shifting the predication time. Another analysis is that the NP does not change the assignment of the parameters but only uses either of them. The disadvantage of the latter analysis is that before a decision is made as to which time point serves for the formation of the NP we must keep the time parameters distinct. We then end up with four parameters rather than three. This is unsatisfactory. We shall therefore assume the first analysis, where the NP is shifting the predication time. To see how this works, we shall present the semantics for nouns and adjectives. The lexical entry for a stage-level property denoting noun is like this:

(5.45)

$$\texttt{/Präsident/}\bigcirc$$

$$\begin{array}{|c|}\hline \langle x : \triangle : \left[\begin{array}{lcr} \text{CAT} & : & n \\ \text{PERS} & : & 3 \\ \text{CLASS} & : & masc \\ \text{NUM} & : & \star \\ \text{CASE} & : & \star \end{array}\right] :: \left[\begin{array}{lcl} \text{PROP} & : & p \\ \text{PRED} & : & t \end{array}\right]\rangle \\ \hline x, p, t \\ \hline p \doteq \mathsf{president}'. \\ \hline \end{array}$$

Note that case and number must be added. See also the next chapter for a fully fledged account of morphological features. There is so far no difference between stage-level and individual level properties. These are distinct semantically (being functions from time points to properties of individuals and properties of individuals, respectively). Below we shall deal exclusively with stage-level properties. An adaptation to cover both types is possible but introduces unnecessary complications. Instead, we shall treat individual level properties as if they were stage-level properties.

We have assumed that those parameters that are not mentioned are simply passed on unchanged. So, the lexical entry for /groß/ can be expanded as follows.

(5.46)

$$\texttt{/groß/}\oslash$$

$$\begin{array}{|c|}\hline \langle x : \Diamond : \left[\begin{array}{lcc} \text{CAT} & : & n \\ \text{PERS} & : & \top \end{array}\right] :: \left[\begin{array}{lcl} \text{PROP} & : & p \mapsto p' \\ \text{REF} & : & t_1 \\ \text{TT} & : & t_2 \\ \text{PRED} & : & t_3 \end{array}\right]\rangle \\ \hline x, p, p', t_1 \\ \hline p' \doteq \mathsf{big}'(p) \\ \hline \end{array}$$

However, the additional parameters may be suppressed as they are not needed. Alternatively, t_2 and t_3 can be replaced by $\circ$. So, we can replace the previous representation by

(5.47)

$$\begin{array}{|c|}\hline \multicolumn{1}{c}{/\mathtt{groß}/\oslash} \\ \hline \langle x : \lozenge : \begin{bmatrix} \textsc{cat} & : & n \\ \textsc{pers} & : & \top \end{bmatrix} :: \begin{bmatrix} \textsc{prop} & : & p \mapsto p' \\ \textsc{pred} & : & t_3 \end{bmatrix} \rangle \\ \hline x, p, p', t_3 \\ \hline p' \doteq \mathsf{big}'(p) \\ \hline \end{array}$$

We have seen earlier that certain adjectives determine whether or not the NP is formed at reference time or at topic time. Their semantics therefore involves more parameters (many of them redundant according to our convention).

(5.48)

$$\begin{array}{|c|}\hline \multicolumn{1}{c}{/\mathtt{heutig}/\oslash} \\ \hline \langle x : \lozenge : \begin{bmatrix} \textsc{cat} & : & n \\ \textsc{pers} & : & \top \end{bmatrix} :: \begin{bmatrix} \textsc{ref} & : & t_1 \\ \textsc{pred} & : & t'_3 \mapsto t_3 \end{bmatrix} \rangle \\ \hline x, t_1, t_3, t'_3 \\ \hline t'_3 \doteq t_1 \\ \hline \end{array}$$

(5.49)

$$\begin{array}{|c|}\hline \multicolumn{1}{c}{/\mathtt{damalig}/\oslash} \\ \hline \langle x : \lozenge : \begin{bmatrix} \textsc{cat} & : & n \\ \textsc{pers} & : & \top \end{bmatrix} :: \begin{bmatrix} \textsc{ref} & : & t_1 \\ \textsc{tt} & : & t_2 \\ \textsc{pred} & : & t'_3 \mapsto t_3 \end{bmatrix} \rangle \\ \hline x, t_1, t_2, t_3, t'_3 \\ \hline t'_3 \doteq t_2; t'_3 < t_1 \\ \hline \end{array}$$

Notice that none of these adjectives contributes to the property in question. They merely reset the predication time for the property. In both cases, there is an unused parameter; in the first case it is the topic time parameter and in the second case the reference time. By our conventions on parameters these can be dropped. We shall remark here that the syntactic behaviour of these adjectives is not totally accounted for by their argument structure. Namely, these adjectives appear typically right after the determiner or the numeral.

(5.50) `der damalige erste Vorsitzende`

(5.51) `?der erste damalige Vorsitzende`

the of.that.time first chairman

‘the first chairman of that time’

(5.52) `die vier damaligen stimmberechtigten Vereinsmitglieder`
`die damaligen vier stimmberechtigten Vereinsmitglieder`
?`die vier stimmberechtigten damaligen Vereinsmitglieder`
the four of.that.time with.right.to.vote club members
‘the four club members of that time who had a right to vote’

The same can be said with respect to the English words /`former`/ and /`alleged`/. One explanation is that for the property that forms the NP it is required that it be homogeneous. Hence, it is disfavoured to shift the time of predication in the middle of the NP. A different case are however the words like /`ehemalig`/ ‘former’ or /`Ex-`/ ‘ex-’, which *explicitly* reset the predication time.

(5.53)

$$\begin{array}{c} \texttt{/Ex-/}\ominus \\ \hline \left\langle x : \blacklozenge : \begin{bmatrix} \text{CAT} & : & n \\ \text{NUM} & : & \star \\ \text{CASE} & : & \star \end{bmatrix} :: \begin{bmatrix} \text{PROP} & : & p \mapsto \\ \text{PRED} & : & t_3' \mapsto t_3 \end{bmatrix} \right\rangle \\ \hline x, p, t_3, t_3' \\ \hline t_3' < t_3; p'(t_3) \doteq \neg p(t_3'). \end{array}$$

This combines with (5.45) to give

(5.54)

$$\begin{array}{c} \texttt{/Ex-Präsident/}\bigcirc \\ \hline \left\langle x : \vartriangle : \begin{bmatrix} \text{CAT} & : & n \\ \text{NUM} & : & \star \\ \text{CASE} & : & \star \end{bmatrix} :: \begin{bmatrix} \text{PROP} & : & p \mapsto p' \\ \text{PRED} & : & t_3' \mapsto t_3 \end{bmatrix} \right\rangle \\ \hline x, p, p', t_3, t_3' \\ \hline t_3' < t_3; p'(t_3) \doteq \neg\mathsf{president}(t_3'). \end{array}$$

This denotes a property, the property of not being at predication time (t_3) what one has been at some time before that, namely, a president. More exactly, it does not even denote a property. Instead, it fixes the property under discussion to be that of being a *p* that was a non-*p* at some earlier time. What is missing is an element that actually predicates the property at predication time. This is discussed in the next section.

Exercise 45. Here is an example from Nootka, which is reported in Sapir 1921 (our discussion is based on a paragraph in Comrie 1985). In Nootka, nouns may

optionally be specified for whether the referent possesses the indicated property right now or whether it possessed that property in the past. The example given is

(5.55) `inikw-ihl-'minih-'is-it-'i`
fire-in:house-plural-diminutive-past-nominal
'the former small fires in the house'

Implement the morphology of Nootka.

5.5 Reconsidering the Structure of the Noun Phrase

We shall now review once again the structure of the noun phrase. Several issues need to be reconsidered. We shall assume that the complex consisting of adjectives and the head noun only specifies a property. Given this property, an individual or a group is being formed. This is done for example by using a numeral or other element designating quantity. Let us take an earlier example again.

(5.56) `quattuor magni mures`
four big-MASC.PL.NOM mouse-PL.NOM
'four big mice '

We shall ignore case for the moment. The lexical entries for /`mus`/ 'mouse' is as follows (with some morphological detail omitted).

(5.57)
$$\begin{array}{c} /\mathtt{mus}/, /\mathtt{mur}/\bigcirc \\ \hline \langle x : \triangle : \begin{bmatrix} \textsc{cat} & : & n \\ \textsc{class} & : & masc \\ \textsc{num} & : & \star \end{bmatrix} :: [\textsc{prop} : p] \rangle \\ \hline x, p \\ \hline p \doteq \mathsf{mouse}' \end{array}$$

Now the lexical entry for plural is as follows. (Compare this with (4.51): the property argument is now turned into a parameter.)

(5.58)
$$\begin{array}{c} /\mathtt{es}/, /\mathtt{i}/\oslash \\ \hline \langle x : \blacklozenge : \begin{bmatrix} \textsc{cat} & : & n \\ \textsc{num} & : & \star \mapsto pl \end{bmatrix} :: [\textsc{prop} : p] \rangle. \\ \hline x, p \\ \hline \end{array}$$

Notice that the plural suffix does not change the property parameter; nor does it contain any meaning. The lexical entry for /`magn`/ 'big' is like this

(5.59)
$$\begin{array}{c} /\mathtt{magn}/\oslash \\ \begin{array}{|c|}\hline \langle x : \Diamond : \begin{bmatrix} \textsc{cat} & : & n \\ \textsc{class} & : & \star \\ \textsc{num} & : & \star \end{bmatrix} :: [\textsc{prop} : p \mapsto p'] \rangle \\ \hline x, p, p' \\ \hline p' \doteq \mathsf{big}'(p) \\ \hline \end{array} \end{array}$$

The gender agreement morpheme has a straightforward semantics. Finally, we introduce the numeral.

(5.60)
$$\begin{array}{c} /\mathtt{quattuor}/\oslash \\ \begin{array}{|c|}\hline \langle x : \Diamond : \begin{bmatrix} \textsc{cat} & : & n \\ \textsc{num} & : & pl \end{bmatrix} :: \begin{bmatrix} \textsc{prop} & : & p \\ \textsc{pred} & : & t \end{bmatrix} \rangle \\ \hline x, t, p \\ \hline (\forall y)(y \in x \rightarrow p(t)(y)) \wedge \sharp x \doteq 4) \\ \hline \end{array} \end{array}$$

This semantics for the numeral works as follows. First, the property is lifted to a property not of individuals but of groups. Next a group is created, whose size is four and has the property of consisting entirely of p-ers.

The structure of the expression is now as follows.

(5.61) `quattuor`((`magn i`) (`mur es`))

The inflection is added first (for that we need fusion), and then the words are composed, with the adjective composing with the noun, and then the numeral is merged with the complex.

It is important to note that it is the numeral that forms the group and which lifts the individual property to a group property. To attribute the group forming property to the plural would make the semantics unduly complicated. For a non-intersective adjective in the plural will expect from its head noun a group property and not an individual property. For example, the adjective /`big`/ is a function from properties to properties. In the singular its semantics is

(5.62) $p' \doteq \mathsf{big}'(p)$

The semantics of /`big`/ in the plural would then be as follows:

(5.63) $p' \doteq \lambda x.(\forall y \in x)\mathsf{big}'(p)(y)$

So, p' is the property of consisting entirely of big p-ers. Leaving the semantics unchanged would give the following result. The property /`magni mures`/ would not be the property of being a set of big mice but the property of being a big set of mice. This is clearly not as it should be. Hence, the semantics of the adjective would have to be changed rather substantially when put into the plural. However, if we take plural not to form the group, matters are in fact quite straightforward. The property is attributed to each individual.

Immediately, one problem appears. In this seupt there is no way to tell from the argument structure whether or not the group has actually been formed. We therefore need some device that informs us about that. For example, we might assume in that case the parameter is *not* passed on. A group is then distinct from a property in that it bears no property parameter. However, this solution is excluded on the grounds that to have no parameter is not in violation of anything. It just means that the variable is anonymous.

Another alternative is to relegate this matter to the presence or absence of the determinateness feature. A noun phrase is complete only when this feature is set, and it in turn can only be set after the group is formed—if a group is formed at all. Indeed, the present framework allows for several alternatives. The first is to assume that the determiner does nothing but to mark off the left edge of the phrase. Here is how the entry for the English determiner will look like.

(5.64)

$$
\begin{array}{|c|}
\multicolumn{1}{c}{/\texttt{a}/, /\texttt{an}/ \rhd} \\
\hline
\langle x : \Diamond : \begin{bmatrix} \textsc{cat} & : & n \\ \textsc{num} & : & sg \\ \textsc{def} & : & \star \mapsto - \end{bmatrix} :: [\textsc{prop} : p] \rangle \\
\hline
x, p \\
\hline
\varnothing \\
\hline
\end{array}
$$

A different representation of /a/, /an/ is one that creates an object from the property. We may either assume that it thereby eliminates the parameter or that it does not.

(5.65)

$$
\begin{array}{|c|}
\multicolumn{1}{c}{/\texttt{a}/, /\texttt{an}/ \rhd} \\
\hline
\langle x : \Diamond : \begin{bmatrix} \textsc{cat} & : & n \\ \textsc{num} & : & sg \\ \textsc{def} & : & \star \mapsto - \end{bmatrix} :: \begin{bmatrix} \textsc{pred} & : & t \\ \textsc{prop} & : & p \end{bmatrix} \rangle \\
\hline
x, t, p \\
\hline
p(t)(x) \\
\hline
\end{array}
$$

Here the indefinite does nothing but to factually attribute the parametric property of the object at predication time! Notice that the variable x was doing no service at all during the composition of the noun phrase except in its function as a "coherence device". We shall assume that it only has the latter function, namely attributing the property of the individual, in case no numeral is present.

Notice that the predication time is used in the determiner in creating the object in question.

Next we look at the definite determiner. It may be used to convey the uniqueness of the object or its salience. In the first case its structure is:

(5.66)

$$\begin{array}{|c|}
\multicolumn{1}{c}{\texttt{/the/}\circleddash} \\
\hline
\langle x : \Diamond : \begin{bmatrix} \textsc{cat} & : & n \\ \textsc{num} & : & sg \\ \textsc{def} & : & \star \mapsto + \end{bmatrix} :: [\textsc{prop} : p] \rangle \\
\hline
x, p \\
\hline
p(x); (\forall y)(p(y) \rightarrow y \doteq x) \\
\hline
\end{array}$$

In the plural, the marker for the indefiniteness is empty in English. Its semantics is the same as in the singular. It attributes the property to the group and asserts that the group is unique with this property. So, the phrase /`the four mice`/ will be interpreted as a group consisting of four mice and which is unique in consisting of four mice. Notice that the determiners do not change the property parameter.

There is a list of quantifiers that provides additional evidence for the existence of properties. These are the so-called proportional quantifiers like /`few`/, /`many`/, /`most`/, /`three quarter of`/, /`all`/. What is common to them is that they do not specify an absolute quantity but a quantity that is relative to the size of the largest group.

We note in passing that /`few`/ also has an absolute reading. For example, /`a few soldiers`/ means a small group of soldiers, while /`few soldiers`/ usually means a small group of soldiers compared the number of soldiers.

Take for example /`all`/. A group consisting of all soldiers is a group comprising all individuals that are soldiers. Without knowing who is and who is not a soldier it is impossible to form that group. Alternatively, and this is the line we are taking here, the group consisting of all soldiers is the set formed by using the

property of soldierhood:

(5.67)

$$
\begin{array}{|c|}
\hline
\langle x : \Diamond : \begin{bmatrix} \text{CAT} & : & n \\ \text{NUM} & : & pl \\ \text{DEF} & : & \star \mapsto - \end{bmatrix} :: [\text{PROP} : p \mapsto p'] \rangle \\
\hline
x, p, p' \\
\hline
x \doteq \{y : p(y)\} \\
p' \doteq \lambda x.(\forall y \in x)p(y) \\
\hline
\end{array}
$$

(header of the box: `/all/`$\oslash$)

So, `/all/` forms the group of all things satisfying the imported property, p. Notice that it also sets the definiteness value to $-$. Even though it forms a group, this group is not definite. With this, the ungrammaticality of the following example is accounted for:

(5.68) *`Watson read these/the all newspapers.`

This is so since the determiner needs as a complement a phrase with undefined definiteness value. Yet, the definiteness is already set, so no determiner may be present. Notice that there is a construction, shown in (5.69), which involves `/all/` and is nevertheless grammatical.

(5.69) `Watson read all (of) the newspapers.`

(5.70) `Watson read few/most/many of the newspapers.`

Similarly with the numerals. This use is most easily accounted for by allowing them to take a full definite plural NP as a complement. This NP must be in the genitive. The expression `/three quarter/` allows only the latter type of construction and may not be used with a property:

(5.71) *`Wayne sent three quarter soldiers to the camp.`

This shows that although the two constructions—taking a property as a complement or a definite plural NP—are related, they are syntactically independent and elements may individually choose to occur in just one or both of the construction types.

With the definiteness value set, the noun phrase may or may not be complete. If the NP is indefinite, then it is already complete. If the NP is definite, it may additionally receive what we call for want of a better name a **proximity value**.

In English, a definite NP can be formed using either the plain definite determiner /the/ or the words /this/ or /that/. We shall assume that they set the proximity value to ± (there may be more values in other languages). These words may also considered as deictic words.

Example 28. Hungarian has a determiner, with two forms, /a/ and /az/. In addition, there are spatial deictics /e/, /ez/ 'this' and /a/, /az/ 'that'. In all three cases, the form ending in /z/ is chosen if the next word begins with a vowel. The determiner does not inflect, unlike the deictics.

(5.72) Volt-am a/egy ház-ban.
be.PAST-1.SG DEF/a house-INESS
'I was in the/a house.'

(5.73) Volt-am eb-ben/ab-ban a ház-ban.
be-PAST-1.SG PROX-INESS/DIST-INESS DEF house-INESS
'I was in this/that house.'

(5.74) Minden ház-ban volt egy cica.
Every house-INESS be-PAST-3.SG a cat
'A cat was in every house.'

(5.75) ez-ek-ben a fehér ház-ak-ban
PROX-PL-INESS DEF white house-PL-INESS
'in these white houses'

Notice that when the proximity marker inflects, the /z/ assimilates to the next consonant. Thus, in place of */ezben/ or */azban/ we get /ebben/ and /abban/. We gloss /e/, /ez/ as PROX (proximate) and /a/, /az/ as DIST (distal). They do not set the definiteness value. This is done by /a/, /az/. The English (near) equivalent /this/ would then have to be glossed as PROX.DEF and /that/ would be DIST.DEF. Case must be repeated after the deictic element. ⚽

We will now address a topic that has so far been kept in the background, namely the relationship between morphological and syntactic bracketing. The default assumption, namely that morphological bracketing is just part of the syntactic bracketing, can be shown to be problematic for many reasons. One is a semantic one. Take the adjective /former/ and the prefix /ex-/. Both have the same semantics, but one is a separate word while the other is only part of a word.

If the syntactic bracketing and the morphological bracketing coincide we would not expect the following two to mean the same.

(5.76) `Peter is the former director of the Bank of Scotland.`

(5.77) `Peter is the ex-director of the Bank of Scotland.`

However, both mean the same thing and therefore /`ex-`/ takes scope over the phrase /`director of the Bank of Scotland`/. The semantics that we have developed is however in large parts associative and therefore there is in this case no need to assume that the syntactic analysis is distinct from the morphological analysis. Nevertheless, there are cases when the semantics is not associative. One such case is the composition of the Hungarian noun phrase. Here, case and plural marking are suffixed to the head noun, which is at the end of the NP. Therefore, the adjectives, quantifiers, numerals and the determiner do not show agreement at all. We have previously argued that this is a morphological fact. In the morphology it is specified that only nouns inflect for number and case. (This applies however also to the deictic words /`ez`/ 'this' and /`az`/ 'that', see Example 28 above.) Now we are in a conflict. An inflected noun needs the adjective to agree with it in the features in which it inflects. But there is no overt agreement. We could argue at this point that adjectives do inflect for all these categories but all forms are identical. If this is assumed we have no problem, we can simply proceed as if Hungarian was like German or Finnish. However, it does not seem to us not the most obvious of all solutions. It also is historically incorrect. It is known that many cases, for example the inessive, have once been inflected nouns. If only morphological cases are iterated, we must assume that at that stage there was no agreement for the inessive. All that happened after that was that the postposition got weaker and eventually became a suffix. It is quite absurd to assume that Hungarian has implemented full case agreement when no such stage can ever be attested.

Thus, we assume that in Hungarian adjectives and determiners do not inflect. Since there is no direct evidence to distinguish these two approaches we shall argue from a historical point of view. If we assume that the categories in which a language categorizes elements from a morphological point of view are by and large arbitrary then we must assume that those categories that the morphology does not use at all are simply undefined rather than being defined but underdetermined. Suppose however that a category exists in the form of a distinct element, for example a postposition, that gradually reduces to, say, a case ending. From the standpoint of the system we previously had no reason to suspect that words are discriminated for case (take by way of example a language like English, Chinese

or Tagalog). Once the morphology has changed and the postposition has been reduced to a case, we do however have a new morphological category, namely case. Now, what shall we say: is case a category of all words or just of some, for example the head noun? I think there is every reason to believe the second. (Mel'cuk 1993 – 2000 argues that case on nouns is *different* from case on adjectives.) The first option would be the result of a development when for example case distinctions are gradually lost (as in English) and the system may still list them as distinct cases, while their forms are already nondistinct. (The English nominative and accusative are a case in point. The two cases are only distinct in the pronouns.) This state of affairs is highly instable, as one might suspect, and will be reshaped into one where the irrelevant distinctions are eliminated. Moreover, once a category has lost all distinctions it may simply be removed.

We conclude from this discussion that it may well be that case morphology is selective in certain categories and that case may be undefined in others. Applied to Hungarian this means that case and number are undefined for the adjective, the numeral, the determiner and the quantifiers. But if that is so, the adjective can no longer combine with an inflected noun. Its case value is ⋆, but that of the complement noun is defined. The solution to this problem is to assume the following analysis for (5.75).

(5.78) `(ez)-ek-ben (a fehér ház)-ak-ban`
PROX-PL-INESS (DEF white house)-PL-INESS
'in these white houses'

Let's assume that the Hungarian plural and case suffixes are not word affixes but phrasal affixes. How can this be achieved? A simple mechanism is to assume that (nominal) case and number markers select a complement that has a defined definiteness value (which may be either definite or indefinite) but whose proximity value is undefined. The consequence is that the noun phrase must be finished up to the determiner /a/, /az/ before the case ending is attached. Moreover, the case ending must be attached there. That the proximity marker also carries case can be explained by the fact that case agreement is mandatory if it wants to combine with the NP, because that NP has the case and number features instantiated. However, we must obviously assume that it actually can inflect for these categories and therefore we must assume that case attaches also to elements in which proximity and definiteness are defined. Moreover, there are nouns which inflect for case in particular the demonstratives /ez/ and /az/. If that is so, the following is expected

Figure 5.1: Phrasal and Word Case in Hungarian

/ban/$\oslash$
$\langle x : \lozenge : \begin{bmatrix} \textsc{cat} & : & n \\ \textsc{case} & : & \star \mapsto \mathit{iness} \\ \textsc{def} & : & \top \\ \textsc{prox} & : & \star \end{bmatrix} \rangle$
$\varnothing$
$\varnothing$

/ban/$\oslash$
$\langle x : \blacklozenge : \begin{bmatrix} \textsc{cat} & : & n \\ \textsc{case} & : & \star \mapsto \mathit{iness} \\ \textsc{def} & : & \top \\ \textsc{prox} & : & \top \end{bmatrix} \rangle$
$\varnothing$
$\varnothing$

to be grammatical as well.

(5.79) *(ez a fehér ház)-ak-ban.
(this DEF white house)-PL-INESS
(lit.) 'in this white houses'

To solve this problem, we shall assume that we have two kinds of affixes, one being a word affix and the other being a phrasal affix. The final nominal case and number suffixes are phrasal (as are the possessive markers), while the case and number markers that are suffixed to the demonstratives and the proximity markers are actually word affixes. They are distinguished as follows. The phrasal suffix needs the proximity value to be $\star$. The word affix on the other hand requires the proximity value to be different from $\star$. The inessive case suffixes are shown in Figure 5.1. Notice that the word affix is fusional, the phrasal affix nonfusional. By this assumption, the example (5.79) is ruled out because the phrasal case affix needs an undefined proximity value. Notice that the same problem appears in the English NP. Here, as there is not much of a case distinction left, there is nevertheless the category of number. However, number is marked at the NP only at the head noun, and in addition at the proximity markers (/this/ vs. /these/ and /that/ vs. /those/). The indefinite article also two forms (/a(n)/ vs. $\varnothing$). In English we must assume that number is a phrasal affix which attaches to a phrase that has its definiteness value undefined. We may however assume that numerals take complements with a number value assigned to them. Therefore the bracketing of the English NP is as follows.

(5.80) these four unsolved thorny problems

this-PL four (unsolved thorny problem)-PL

This is the only bracketing possible, since otherwise the the adjectives cannot combine with their complements.

Notes on this section. As we have argued earlier (see 2.7) the semantics of the actual inflection marker is empty. However, there are exceptions to this rule. In Hungarian the plural marker is obligatorily absent in the presence of a numeral. Thus, the plural marker signals a multitude, just as the numeral /négy/ 'four' signals 'four'. Let me also briefly remark on the issue of pluralia tanta. The difference beween pluralia tanta and ordinary nouns is that the former are listed in the lexicon without a root form. For example, the Latin word /litterae/ is ambiguous between the plural of /littera/ 'letter (of the alphabet)' and the pluralium tantum /litterae/ 'letter'. The lexicon contains both /littera/ 'letter of the alphabet' as a root noun and /litterae/ 'letter' which has the argument structure of a plural noun.

5.6 Predicative and Attributive Adjectives

This section is devoted to adjectives. It is a bit programmatic, identifying problems areas more than showing actual solutions to them. It will be clear that the kinds of problems that we encounter with adjectives will also appear elsewhere.

Adjectives occur basically in three types of environments. They can be modifiers of a noun, they can modify verbs (in which case they are called *adverbs*; we shall group both categories together here). They can be used predicatively, for example in postmodifiers in English or in postcopular position, and finally they can occur in what is syntactically often analysed as a small clause. Each of these constructions is distinct, and one can find that languages group them in different ways, as we have seen earlier. Here we shall be concerned with the implications of these facts for the semantic structure of adjectives. Let us first illustrate these types of contexts.

(5.81) `John is a clever student.`

(5.82) `John is running fast.`

(5.83) `John, proud of his achievement, went into the office.`

(5.84) `John is clever.`

(5.85) `John drove the car drunk.`

(5.86) `John drank himself stupid.`

In (5.81) the adjective modifies a noun, in (5.82) it modifies a verb. (5.83) shows an adjectival phrase in postnominal position. Typically, this construction is used to make another assertion, one whose connection with the main assertion can only be guessed (here it seems simply that the two are contemporaneous). (5.84) is a case of a postcopular adjective and (5.85) is a depictive. In the syntactic literature this construction is analysed as a small clause type of construction, (5.86) is a resultative.

Certain things need to be noted. First, none of these constructions is restricted to adjectives (PPs or NPs can also be used in virtually them); second, not all adjectives can be put into all of these contexts. A good example is /`alleged`/, which refuses to appear in postcopular position or as an adverbial. So, some care has to be exercised with respect to the generalizations that will arise from the semantics. The basic problem with adjectives is that their representations only license them to appear as nominal modifiers. They also cannot be in postnominal position because they are prenominal modifiers in English. They cannot be in postcopular position because they need a noun to modify and there is none. They can also not be depictives or resultatives. That this is no accident is corroborated by the fact that these constructions are marked by morphological distinctions. In German, the adjective inflects only when used as a prenominal modifier. Otherwise, it takes one and the same form. We might therefore propose that the other three construction types require an adverbial. However, we consider an adverbial only a modifier of a verb, and by this criterion the postcopular and the postnominal attribute is certainly not an adverbial. In Hungarian, the adverb is distinct from the adjective and is used only is the true adverbial context. (Note that the copula is zero in the third person.)

(5.87) `János csendes-en dolgozik.`
János silent-ly works.
'Janos works quietly.'

(5.88) `Ez a motor csendes.`
PROX DEF motor (is) silent.
'This motor is quiet.'

(5.89) `Ez-ek a motor-ok csendes-ek.`

PROX-PL DEF motor-PL silent-PL
'These motors are quiet.'

We see therefore that the constructions must be kept distinct. (In English this is generally also the case; however, certain verbs do not require the adverbial form, like /`drive`/, and some adjectives are nondistinct from their derived adverbs like /`fast`/.)

In Finnish the adjective must appear in the essive if it is used in a depictive and in the translative if used in a resultative. Now the semantic difference is as follows. While the postnominal adjective does not take part in the formation of the group it functions practically as a separate assertion on the group. We may analyze postnominal adjectives as if they head separate clauses. One difficulty in accounting for the various facts surrounding the adjective is that in some languages they inflect in some positions and not in others, and in still other languages it might be different. In Hungarian, the adjective does not inflect in prenominal position, but it does in postcopular position. In German it is the converse. In French it inflects in both positions. In Georgian it inflects differently when used postnominally (see Fähnrich 1993). Another difficulty is that adjectives appearing in postcopular position function as if they are nouns. For if the copula takes the adjective as one argument and the subject as another, there is still the complement of the adjective missing in the construction. The construction would be incomplete in this way. We shall therefore assume that the adjective appears with a dummy property inserted, which may for example be equated with the property that the subject provides. For example, take the sentence

(5.90) `This mouse is big.`

We shall assume that either the mouse is said to be big in the absolute sense or that it is big in the sense of being a mouse (or in another contextually given sense). It is the latter interpretation that interests us. Take the lexical entry for the English adjective /`big`/.

(5.91)

/`big`/$\oslash$

$\langle x : \Diamond :$	CAT : n NUM: $\star$	$:: [\text{PROP} : p \mapsto q]\rangle$
x, p, q		
$q \doteq \mathsf{big}'(p)$		

We need to get rid of the variable x. Here is a first solution. Let us assume that there is an empty element ONE that acts as an argument to the adjective.

$$(5.92)\quad \begin{array}{|l|}\multicolumn{1}{c}{/\textsc{one}/\bigcirc}\\ \hline \langle x : \vartriangle : [\textsc{cat} : n] :: [\textsc{prop} : p]\rangle \\ \hline x, p \\ \hline \varnothing \\ \hline \end{array}$$

It follows that the adjective together with ONE has the following structure

$$(5.93)\quad \begin{array}{|l|}\multicolumn{1}{c}{/\texttt{big} + \textsc{one}/\bigcirc}\\ \hline \langle x : \vartriangle : [\textsc{cat} : n] :: [\textsc{prop} : q]\rangle \\ \hline x, q, p \\ \hline q \doteq \mathsf{big}'(p) \\ \hline \end{array}$$

A different route is to assume that the adjectival root actually has a property variable to begin with.

$$(5.94)\quad \begin{array}{|l|}\multicolumn{1}{c}{/\textsc{big}/\bigcirc}\\ \hline \langle p : \vartriangle : [\textsc{cat} : p]\rangle \\ \hline p \\ \hline p = \mathsf{big}' \\ \hline \end{array}$$

It is then turned into nominal modifier:

$$(5.95)\quad \begin{array}{|l|}\multicolumn{1}{c}{/\textsc{one}/\bigcirc}\\ \hline \langle x : \vartriangle : [\textsc{cat} : n] :: [\textsc{prop} : p]\rangle \\ \langle p : \triangledown : [\textsc{cat} : p\rangle \\ \hline x, p \\ \hline \varnothing \\ \hline \end{array}$$

We shall return to this issue below.

Next, we shall assume that the copula has the following form

$$(5.96)\quad \begin{array}{|l|}\multicolumn{1}{c}{/\texttt{be}/\bigcirc, \oslash, \ominus}\\ \hline \langle e : \vartriangle : [\quad]\rangle \\ \langle x : \triangledown : [\textsc{cat} : n]\rangle \\ \langle x : \triangledown : [\textsc{cat} : n] :: [\textsc{prop} : p]\rangle \\ \hline p, x \\ \hline p(x) \\ \hline \end{array}$$

The exported variable is not actually needed. Some semanticists think that the event consists in x's being p, but I do not see much gain in this analysis. A big disadvantage of this structure is that it does not specify the relation between the two arguments of the copula. There is no specification that they should agree, for example. However, evidence for agreement is mixed; in French, there is full agreement in all relevant categories between the subject and the postcopular adjective. In German it is absent, as we have seen. If we allow the introduction of variables for values of attributes, then agreement in gender is written into the structure as follows.

(5.97)

$$\begin{array}{c} /\mathtt{be}/\bigcirc, \otimes, \otimes \\ \hline \begin{array}{|l|} \langle e : \vartriangle : [\quad] \rangle \\ \langle x : \triangledown : \left[\begin{array}{l} \text{CAT}: n \\ \text{GEN}: \gamma \end{array}\right] \rangle \\ \langle x : \triangledown : \left[\begin{array}{l} \text{CAT}: n \\ \text{GEN}: \gamma \end{array}\right] :: [\text{PROP} : p] \rangle \\ \hline p, x \\ \hline p(x) \\ \hline \end{array} \end{array}$$

The role of the two arguments is semantically asymmetric. The inner argument supplies a property, the outer (= subject) and individual. If this is correct we expect that NPs that do not denoted properties are not allowed in predicative position. This is borne out.

(5.98) `John is a fool.`

(5.99) `John is the biggest fool on earth.`

(5.100) *`John is every husband.`

(5.101) `They are the soldiers.`

(5.102) ?`They are a few soldiers.`

(5.103) ?`They are most of the soldiers.`

Quantified NPs are generally disallowed in postcopular positions. This is because they do not denote properties. (There seems to be an exception to this only the fact that a quantified NP can ascribe that the subject contains that many individuals of the described property. This is a plausible reading for these sentences. This reading would have to be accounted for, but our present discussion provides no means for doing so.) We claim that constructions of the kind "X is Y" ascribing

to X the property of being identical to Y.

(5.104) `John is the dean of this faculty.`

(5.105) `Tully is Cicero.`

On the other hand, the subject must denote an individual or quantify over individuals. The requirement that the subject be an individual is not so strict, however.

Now, in order to be able to prevent the quantified NPs from appearing in predicate position and in order to assign the proper semantics to those NPs that do appear there we must in fact assume that NPs exist in two kinds: as object denoting NPs and as property denoting NPs. We shall therefore, for want of a better solution, introduce a feature PROP with values + for a property and − for a non-property, ie an individual or a group. (This picks up a theme that we have discussed inconcluively in the previous section.) The idea is that only entities with [PROP : +] can appear in postcopular position. To make this work, we shall take it that adjectives are [PROP : +] together with simple nouns, and that the numeral or quantifier resets this value to [PROP : −]. The property feature is therefore an indicator of whether or not a group or an individual has been formed or whether the NP is taken to denote a property. The determiners can do both. The indefinite can be used to form an NP denoting an individual, while it can be used to form a property as well. Likewise the definite determiner, although there is a preference to use it to create individuals. But note the use in (5.99) of the definite determiner in connection with adjectives in the superlative. A different solution is to adopt a new empty element which can change an NP into a property:

(5.106)

/PRP/⊗

$\langle x : \Diamond :$	DEF : + PROP : − $\mapsto$ +	$:: [\text{PROP} : \circ \mapsto q]\rangle$
x, q		
$q \doteq (\lambda y.y \doteq x)$		

Here, only the relevant details are shown. Notice the interplay between the parameters and the objects. The object x disappears in the semantics (even though it is formally still present), while it is recoded as the property of being identical to x. Notice that it is required that the NP is definite. This makes sure that the object x has been formed. Moreover, it would fail badly if it were applied to indefinite NPs as well.

The distinction between properties and individuals is also useful for a number of verbs that rather than taking an object as argument require a property. A clear example is /`to call`/.

(5.107) `The people call Arno a master.`

It is clear that /`a master`/ is not an indefinite NP but rather a property attributed to Arno. An interesting fact about such verbs is that in certain languages the property denoting NP shares its case with the object.

(5.108) `Die Leute nennen Arno/ihn einen Meister.`
The people call Arno-ACC/him-ACC a-ACC master-ACC

(5.109) `Arno/Er wird von den Leuten ein Meister genannt.`
Arno-NOM/he-NOM is by the people a-NOM master-NOM called.

In our framework this can be implemented by introducing variables for values. Then we can let the subject and the property share the same variable, and this ensures agreement in the features.

Now we turn to the adverbs. What the adverbs have in common with the postcopular adjectives is that they are construed without a complement. Hence, we shall assume that they are construed with the help of the element ONE. In contrast to nominal modifiers the adverbs must also determine which of the arguments they want to modify. This is called **orientation**.

(5.110) `Walter is driving the car fast.`

(5.111) `Walter is driving the car drunk.`

In (5.110) the adverb /`fast`/ modifies the speed of the car, not that of Walter (he could use telecontrol, for example). In (5.111) it is Walter who is drunk, not the car. In the first case we speak of **object orientation** and in the second case of **subject orientation**. Notice however that orientation is not determined by grammatical status. It changes with diathesis. If an adjective shows object orientation in an active sentence it shows subject orientation in the corresponding passive sentence. In German there also exists an impersonal passive. An adjective showing subject orientation in an active sentence can be used in the impersonal

passive:

(5.112) `Johann warf den Ball weit weg.`
‘John threw the ball far away.’

(5.113) `Der Ball wurde weit weg geworfen.`
‘The ball was far thrown far away.’

(5.114) `Gesine tanzte schön.`
‘Gesine danced beautifully.’

(5.115) `Es wurde schön getanzt.`
‘People danced beautifully.’

The object that ends up being far is the object in (5.112) and the subject in (5.113). The performers of the dance are the subject in (5.114) and ist left unexpressed in (5.115).

We make a first attempt at giving an entry for a morpheme that produces the actor-oriented adverbial from an adjective such as /`hasty`/.

(5.116)

/`ly`/ $\bigcirc \oslash$
$\langle e : \lozenge : [\quad]\rangle, \langle x : \triangledown : [\textsc{cat} : n] :: [\textsc{prop} : p]\rangle$
x, p
$(\forall t)(t \in \mathsf{time}'(e) \to p(t)(x)),$ $\mathsf{act}'(e) \doteq x.$

We encounter a problem: we cannot get rid of the argument variable. Hence either we obligatorily feed the argument ONE (5.92) or we assume that the adjectives are formed form a root, as in (5.94). We prefer the latter alternative. Thus we change the entry above as follows.

(5.117)

/`ly`/ $\bigcirc \oslash$
$\langle e : \lozenge : [\quad]\rangle, \langle p : \triangledown : [\textsc{cat} : n]\rangle$
x, p
$(\forall t)(t \in \mathsf{time}'(e) \to p(t)(x)),$ $\mathsf{act}'(e) \doteq x.$

Notice that in German, the adjective does not inflect in predicative position. This fits well with our proposal that the predicate adjective is actually derived directly from the adjectival root.

Finally, we shall turn to resultatives. This is a very interesting construction. The resultative introduces a result state of the event; moreover, it adds a new transitive object to the verb. In German, the verb plus resultative behaves just like a transitive verb; the resultative object can be passivised and scrambled. Furthermore, resultatives that consist of a directional PP show the same behaviour for this PP.

(5.118) `Peter trank seinen Kumpel unter den Tisch.`

(5.119) `Seinen Kumpel trank Peter unter den Tisch.`

(5.120) `Unter den Tisch trank Peter seinen Kumpel.`
'Peter drank his buddy under the table.'

(5.121) `Peters Kumpel wurde unter den Tisch getrunken.`
'Peter's buddy was drunk under the table.'

Exercise 46. Create an entry for deriving an adverbial with object orientation.

Exercise 47. If the adverb /`csendesen`/ is derived from the adjectival root /`csendes`/ by the addition of a subject-orientation marker (as suggested above, see also the previous exercise), we would expect that (5.87) means that John is working and that he is quiet. Discuss why this analysis is semantically deficient. Can you suggest a better one?

5.7 Sequence of Tense

Tenses can be both deictic and anaphoric. The difference can be made manifest in this calculus in the way the time parameters are being linked. The linking of parameters actually ties this phenomenon together with another one that has received growing attention in recent years, namely what is known in traditional grammars of Latin as **consecutio temporum** or in modern terminology **sequence of tense** (see Abusch 1997 and Ogihara 1996). The problem is simply put the following. In subordinate clauses, tenses do not necessarily take the reference time of the main clause as their reference time, but may instead choose to set the

reference time differently. For example, Russian differs from English in that the subordinate clause sets its reference time to the event time of the main clause, while in English the reference time is not adjusted. The difference comes out clearly in the following example.

(5.122) `Pjetja skazal, čto Misha plačet.`
Pjetja said that Misha is crying
'Pjetja said that Misha was crying.'

This shift in tense does not appear in relative clauses:

(5.123) `Pjetja vstretil čeloveka, kotory plačet.`
Pjetja met a person who is crying
'Pjetja met a person who is crying.'

How do we account for the different behaviour of tenses in Russian and English? Recall that verbs like /`say`/, /`promise`/ and so on select a tensed subordinate clause. They may therefore adjust the parameters of the subordinate clauses. Therefore, the following appears in the argument structure of the verb /`to say`/.

$$\left\langle e : \triangle : \alpha :: \begin{bmatrix} \text{REF} & : & t_1 \\ \text{TT} & : & t_2 \\ \text{PRED} & : & t_3 \end{bmatrix} \right\rangle, \left\langle e' : \triangledown : \alpha' :: \begin{bmatrix} \text{REF} & : & u_1 \\ \text{TT} & : & u_2 \\ \text{PRED} & : & u_3 \end{bmatrix} \right\rangle \tag{5.124}$$

There are six parameters, three for the main clause and three for the subordinate clause. Since the tenses of the subordinate clause fix u_2 and u_3 with respect to u_1, we minimally need to give a value to u_1. The different choices are to set u_1 to one of t_1, t_2 and t_3. Suppose that u_1 is set to t_1. Then the reference time of the subordinate clause is the same as the reference time of the main clause. In this case we have to use past tense if the event of the subordinate clause happens at the same time as the one of the main clause and the main clause is in the past tense. This is the situation in English. If we set u_1 to t_2, then if both events happen at the same time, the subordinate clause is in the present tense. This is the situation in Russian. The same would happen if we took u_3 to be t_3. The results would be different if the main clause was in the pluperfect. We are not in a position to test the difference, however.

The situation is however somewhat more involved than that. Here is an exam-

ple.

(5.125) `Yesterday, John decided that tomorrow morning he would start working.`

The embedded event happens in the future, seen from the perspective of the main clause. Yet, we do not get the future tense, but what is known as *future in the past*. This tense is used when the topic time is in the past from the reference time but the predication time is in the future of the topic time. We conclude therefore that in English the subordinate clause not only fixes the value of the reference time of the subordinate clause, but also the topic time. The topic time is set to the predication time of the main clause (the time of John's decision). The reason why we get the future in the past is the following. The verb in the embedded clause must be tensed, and the tense must be such that the topic time of the embedded clause is anterior to its reference time. However, the predication time is after the topic time (and also after the reference time, but that does not count here), and so the resulting tense is future in the past.

It is expected that if the reference time of a sentence is reset, the time referred to by temporal adverbials is shifted as well. However, we find that there are three classes of adverbials. The first class may be called **event relative**, the second **utterance relative** and the third **absolute**. Absolute adverbials are dates, such as /`on 1st of May`/, /`in 1900`/ and so on. By definition, the time point they refer to is fixed, and does not depend on the context, in particular utterance time or any other time points in the sentence. We give an example from Comrie 1985. Let us assume that today is the 13th of May. On the 8th of May Kolya says

(5.126) `Ja pridu četyrnadcatogo maja.`
'I will arrive on the 14th of May.'

If this is reported today, one would have to say

(5.127) `Kolya skazal, čto on pridet četyrnadcatogo maja.`
'Kolya said that he would arrive on the 14th of May.'

Utterance relative adverbials are /`now`/, /`tomorrow`/, /`yesterday`/. When they are used, they fix the time point relative to the point of utterance. For example, by using /`yesterday`/ in the main clause and /`tomorrow`/ in the subordinate clause in the example (5.126), it is guaranteed that the predication time of the main clause is

the day before the utterance, while the predication time of the subordinate clause is the day after the utterance, in particular it happens after the predication time of the main clause (which is why the future in the past is obligatory). It the example we expect that we can exchange the expression /`on 16th of May`/ with /`tomorrow`/. In English this is fine. The same in Russian:

(5.128) `Kolya skazal, čto on pridet zavtra.`
'Kolya said that he would arrive tomorrow.'

However, as Comrie notes, if today was the 15th of May and not the 13th, then we can say

(5.129) `Kolya skazal, čto on pridet četyrnadcatogo maja.`
'Kolya said that he would arrive on the 14th of May.'

but we cannot say

(5.130) `Kolya skazal, čto on pridet včera.`
'Kolya said that he would arrive yesterday.'

This, he explains, is a fact of Russian grammar. It is not allowed to collocate an adverbial with past reference with a future tense. What is crucial is that the past reference must be overtly marked on the adverbial, and not simply accidental, as with absolute adverbials.

The third class of adverbials are the event relative adverbials. These are /`the day after`/, /`the day before`/, /`on that day`/. Hence we find the following.

(5.131) `Džon skazal: 'Ja ujdu zavtra.'`
'John said: 'I will leave tomorrow.''

(5.132) `Džon skazal, čto on ujdet na sledujuščij den.`
'John said that he would leave the following day.'

Although the tenses in the subordinate clauses are different, as explained above, the adverbials function in the same way. They fix the predication time relative to the topic time. We have already said that the topic time of the subordinate clause is set to the time of John's uttering that sentence, which is the predication time of the main clause. The adverbial /`the following day`/ establishes that the predication time is one day after the topic time.

The semantics of these adverbials is as follows. Adverbials modiy the event, and they may therefore modify any of the three time parameters. Topic time modification is exemplified by date expressions.

(5.133)

$$\begin{array}{l} \texttt{/on 14th of May 1999/}\ominus \\ \hline \left\langle e : \lozenge : \left[\textsc{cat} : e\right] :: \left[\textsc{tt} : t\right] \right\rangle \\ \hline \varnothing \\ \hline t \in \mathsf{14\text{-}May\text{-}1999'} \end{array}$$

(The directionality shall not be of importance here.) Somewhat more difficult are relative expressions like /`the day before`/, because they set the topic time relative to some other time point established earlier. We mimick this by assuming that the event passes upwards a different time point, which can be picked up by connecting elements such as higher heads.

(5.134)

$$\begin{array}{l} \texttt{/the day before/}\ominus \\ \hline \left\langle e : \lozenge : \left[\textsc{cat} : e\right] :: \left[\textsc{tt} : t \mapsto t'\right] \right\rangle \\ \hline \varnothing \\ \hline \mathsf{one\text{-}day\text{-}before'}(t, t'). \end{array}$$

On the other hand, absolute expressions need no such mechanism.

(5.135)

$$\begin{array}{l} \texttt{/yesterday/}\ominus \\ \hline \left\langle e : \lozenge : \left[\textsc{cat} : e\right] :: \left[\textsc{tt} : t \mapsto \circ\right] \right\rangle \\ \hline \varnothing \\ \hline \mathsf{one\text{-}day\text{-}before'}(t, \mathsf{now'}). \end{array}$$

Notice the use of the deictic now′ rather than utterance time.

Example 29. Let us look at the sequence of tense in Latin. Consider a subordinate sentence A with its superordinate sentence S. If A is a so-called independent subordinate sentence the tense/aspect is calculated largely independently of S. If however A is an attitude report or an indirect speech report then rules are as follows.

(5.136)

$\downarrow S$	$\to A$		
	simult with$S$	before$S$	after$S$
present, future I	present subj	perfect subj	`-turus sim`
other tenses	past subj	pluperfect subj	`-turus essem`

(5.137)	`Scimus quid`	`agas`	`egeris`	`acturus sis`
	We know what	you do	you did	you will do
(5.138)	`Scibamus quid`	`ageres`	`egisses`	`acturus esses`
	We knew what	you did	you had done	you would do

For the forms of the Latin verb see the next chapter.

Exercise 48. Develop entries to account for the rules of Latin as shown in Example 29.

Chapter 6

Latin

In this chapter we present an implementation of Latin, both of the verbal paradigm and the nominal declension. Though the implementation is not full, it shows the essential details.

6.1 The Morphology of Latin

Latin has a rather rich morphology and is an ideal testing ground for the present theory. Matthews 1978 has written an entire book on the subject, mainly arguing against the item-and-arrangement model (such as the present one) and in favour of the item-and-process model. Similarly, Mel'cuk 1993 – 2000 allows for a modicum of nonconcatenative processes, though in reality he makes only limited use of such processes. Whenever possible he will arrange for a concatenative solution. Recall that since we allow for discontinuity, a concatenative solution in our sense is far more general than normally understood. It covers infixation, transfixation, suprafixation, wrapping, and more. In the light of this, it will come as no surprise that the concatenative approach with discontinuous constituents is quite powerful.

This said, we now delve into the subject matter. Clearly, we will not show the complete morphology of Latin, as this can be found in many books, and is not the primary goal of this chapter. Instead, we just provide a rough outline. The implementation is actually far richer (but even that is not complete). The Latin verb distinguishes four classes of verbs, divided into two groups. The first group

contains the thematic verbs, covering the a-class (laudāre), e-class (delēre) and the i-class (audīre); the second groupd consists of the consonantal verbs, which fall into two subgroups, the consonantal verbs proper (tegere) and so-called short-i-verbs (capere). Irregular verbs invariably are consonantal, but can be of either kind. The thematic verbs are quite alike. The key difference is the vowel that is added to the verbal stem. The vowel determines some minor changes in the affixes, mostly with respect to the first person singular.

We shall show the full paradigm of /laudāre/ 'to praise', and then comment on the difference in the other paradigms. The paradigm consists in 4 Tables. Table 6.1 shows the forms of the present, past and future active, as well as the infinitive, gerund and participle as well as the imperatives (there are two, the imperative I is like the usual imperative; the imperative II exists in forms for the 2nd and the third person, though the forms in the singular are identical). Table 6.2 shows the corresponding passive forms. Note that there is no participle, and there is no 2nd plural form for the imperative II. The listed form is of the 3rd plural.

There is otherwise a strong parallel between active and passive forms. The tense and mood are expressed by a single affix: zero for the present indicative, /e/ (or /a/) for the present subjunctive, /ba/ for the past indicative, /re/ for past subjunctive and a variety of forms for the future tense: /b/ plus some vowel, or /a/, depending on conjugation class. The suffixes are completely parallel in the active and passive voice. However, in the perfective aspect things fall apart.

It is thus mostly in the endings for the subject agreement that we can distinguish passive from active. It is easy to see that there is a separate set of forms for the passive. The 1st person singular shows some idiosyncratic behaviour in the present and future indicative.

The perfective active is marked by the suffix /v/, added after the thematic vowel. Also, the personal endings in the perfect indicative are irregular in the 2nd person (/sti/ and /stis/). The tense and mood markers are quite different from the ones in the presentive aspect, even though the perfective aspect is clearly marked. We have /i/ in the perfect indicative, /eri/ in the subjunctive, /era/ in the pluperfect indicative, /isse/ in the pluperfect subjunctive, and, finally, /er/ or /eri/ in the future II (which make the forms nondistinct from the perfect subjunctive with the exception of the 1st singular).

Finally, the passive forms in the perfective are formed with the help of the participle /laudātus/ 'praised', which inflects thematically, like an adjective, and the

auxiliary /esse/ ‘to be’. The participle shows agreement in case (invariably nominative), number and gender with the subject, so this requires additional treatment. The table shows only the form of the auxiliary. There are no imperative forms in the passive perfect, but the forms of the auxiliary are listed nevertheless.

I omit the supine and other minor forms. This completes the paradigm of the a-class.

I briefly turn to the two other classes. In the e-class, there are two differences. The first is that the thematic vowel is not empty in the 1st singular present: /dēleō/ ‘I destroy’. The second difference is the affix for the present subjunctive, which is /a/ rather than /e/. Again, the thematic vowel is present. Thus the forms are: /dēleam/ ‘I would destroy’, /dēleās/ ‘you would destroy’, /dēleat/ ‘he/she/it would destroy’, and so on. Similarly for the i-class, the only difference being that the thematic vowel is i: /audiō/ ‘I hear’, /audiam/ ‘I would hear’, /audiās/ ‘you would hear’, /audiat/ ‘he/she /it would hear’.

We shall omit the consonantal inflection except for the perfective. Here, the tense, mood and personal affixes are the same as in the thematic classes, the difference is that the perfective stem is formed using an array of different means (see Matthews 1978): adding /s/ (which in writing means turning /c/ into /x/, for example), reduplicating the onset, ablaut, loss of nasalisation, and so on.

In the nominal inflection there are again different two groups, thematic and athematic. The thematic nouns are are subdivided into the a- and o-class nouns, which are by far the biggest classes, and the e- and u-class. We shall deal only with a- and o-class nouns. The forms are listed in Table (6.6) as the forms of the adjective /bonus/ ‘good’. Thematic adjectives can only use the forms of the a- and o-class nouns, the rule being that a-class is used for feminine agreement and o-class for masculine and neuter agreement. The corresponding nominal forms are the same. In general, a-class nouns are feminine, though exceptions exist (/nauta ‘seafarer’ is one). The o-class nouns ending in /us/ are masculine and the o-class nouns ending in /um/ are neuter.

There are many more declension paradigms. I omit the other thematic declensions. There is once again a separate athematic or consonantal declension. This time, the endings are different from the thematic endings, unlike the verbal paradigms. The adjective inflects like a noun, but differences exist. The genitive plural sometimes ends in /um/ in place of /ium/. The ablative singular often ends in /e/ rather than /i/; a handful of adjectives show the same behaviour. Adjec-

Table 6.1: laudare ‘to praise’: Present, Past and Future Active Forms

Present	Indicative	Subjunctive
1.Sg	laudō	laudem
2.Sg	laudās	laudēs
3.Sg	laudat	laudet
1.Pl	laudāmus	laudēmus
2.Pl	laudātis	laudētis
3.Pl	laudant	laudent
Past		
1.Sg	laudābam	laudārem
2.Sg	laudābās	laudārēs
3.Sg	laudābat	laudāret
1.Pl	laudābāmus	laudārēmus
2.Pl	laudābātis	laudārētis
3.Pl	laudābant	laudārent
Future		
1.Sg	laudābō	
2.Sg	laudābis	
3.Sg	laudābit	
1.Pl	laudābimus	
2.Pl	laudābitis	
3.Pl	laudābunt	
Infinitive	laudāre	
Gerund	laudāndī	
Participle	laudāns	
Imperative I.Sg	laudā	
Imperative I.Pl	laudāte	
Imperative II.Sg	laudātō	
Imperative II.Pl	laudātōte	
	laudāntō	

Table 6.2: laudare ‘to praise’: Present, Past and Future Passive Forms

Present	Indicative	Subjunctive
1.Sg	laudor	laudem
2.Sg	laudāris	laudēs
3.Sg	laudātur	laudet
1.Pl	laudāmur	laudēmus
2.Pl	laudāmini	laudētis
3.Pl	laudantur	laudent
Past		
1.Sg	laudābar	laudārem
2.Sg	laudābāris	laudārēs
3.Sg	laudābātur	laudāret
1.Pl	laudābāmur	laudārēmus
2.Pl	laudābāmini	laudārētis
3.Pl	laudābantur	laudārent
Future		
1.Sg	laudābor	
2.Sg	laudāberis	
3.Sg	laudābitur	
1.Pl	laudābimur	
2.Pl	laudābimini	
3.Pl	laudābuntur	
Infinitive	laudārī	
Gerund	laudandus	
Participle	—	
Imperative I.Sg	laudāre	
Imperative I.Pl	laudāmini	
Imperative II.Sg	laudātor	
Imperative II.Pl	laudantor	

Table 6.3: `laudare` 'to praise': Perfect, Pluperfect and Future II Active Forms

Perfect	Indicative	Subjunctive
1.Sg	laudāvī	laudāverim
2.Sg	laudāvisti	laudāveritis
3.Sg	laudāvit	laudāverit
1.Pl	laudāvimus	laudāverimus
2.Pl	laudāvistis	laudāveritis
3.Pl	laudāvērunt	laudāverint
Pluperfect		
1.Sg	laudāveram	laudāvissem
2.Sg	laudāverās	laudāvissēs
3.Sg	laudāverat	laudāvisset
1.Pl	laudāverāmus	laudāvissēmus
2.Pl	laudāverātis	laudāvissētis
3.Pl	laudāverant	laudāvissent
Future II		
1.Sg	laudāverō	
2.Sg	laudāveris	
3.Sg	laudāverit	
1.Pl	laudāverimus	
2.Pl	laudāveritis	
3.Pl	laudāverunt	
Infinitive	laudāvisse	

Table 6.4: `esse` ‘to be’ Present, Past and Future Active Forms

Present	Indicative	Subjunctive
1.Sg	sum	sim
2.Sg	es	sīs
3.Sg	est	sit
1.Pl	sumus	sīmus
2.Pl	estis	sītis
3.Pl	sunt	sint
Past		
1.Sg	eram	essem
2.Sg	erās	essēs
3.Sg	erat	esset
1.Pl	erāmus	essēmus
2.Pl	erātis	essētis
3.Pl	erant	essent
Future		
1.Sg	erō	
2.Sg	eris	
3.Sg	erit	
1.Pl	erimus	
2.Pl	eritis	
3.Pl	erunt	
Infinitive	esse	
Imperative I.Sg	es	
Imperative I.Pl	este	
Imperative II.Sg	estō	
Imperative II.Pl	estōte	
	suntō	

tive may have one, two (like /brevis/) or three different forms in the nominative singular; however, all other forms are regular. The most important point of irregularity is the stem. Nouns and adjectives derive their forms generally from the *oblique stem* with the exception of the nominative singular, and in the case of neuter noun, also the accusative, since that is generally the same as the nominative. There exist rules of thumb to discern the gender and the oblique stem. Here are some examples.

- Nouns in /or/ denote an actor. They are masculine. The oblique stem is identical to the nominative stem. Example: /orātor/ 'orator', genitive /oratōris/.
- Nouns in /tūdo/ denote abstract properties. They are feminine. The genitive stem is formed by adding /tūdin/ instead. Example: /turpitūdō/ 'shame', genitive /turpitūdinis/.
- Nouns in /men/ denote abstract things. They are neuter. The genitive stem is formed by adding /min/ instead. Example: /certamen/ 'battle', genitive /certaminis/.

The change from nominative to oblique stem is often minor; it may consist in a change of the vowel (certamen/certaminis), lenition in the nominative (ars/artis), or orthographic idiosyncrasies (vox/vōcis).

Notice that the dative is nondistinct from the ablative in the o-class and generally in the plural. Notice also that neuter nouns do not distinguish nominative from accusative.

Morphological Irregularities Main irregularities are of the following kind. There are nouns that may exist only in the singular or only in the plural. Nouns that are only in the singular are by far those where the plural can be formed but is semantically meaningless, the biggest group of which are the mass nouns. Nouns that have only plural forms, so called **pluralia tanta**, are however different. They are of various kinds.

- Only the plural form exists, with plural meaning: /arma/ 'weapons'.
- Only plural form exists, with singular (or plural) meaning: /castra/ 'camp'.

Table 6.5: bonus 'good'

Singular	Masculine	Feminine	Neuter
Nominative	bonus	bona	bonum
Genitive	bonī	bonae	bonī
Dative	bonō	bonae	bonō
Accusative	bonum	bonam	bonum
Ablative	bonō	bonā	bonō
Plural			
Nominative	bonī	bonae	bona
Genitive	bonōrum	bonārum	bonōrum
Dative	bonīs	bonīs	bonīs
Accusative	bonōs	bonās	bona
Ablative	bonīs	bonīs	bonīs

Table 6.6: brevis 'short'

Singular	Masculine	Feminine	Neuter
Nominative	brevis	brevis	breve
Genitive	brevis	brevis	brevis
Dative	brevī	brevī	brevī
Accusative	brevem	brevem	breve
Ablative	brevī	brevī	brevī
Plural			
Nominative	brevēs	brevēs	brevia
Genitive	brevium	brevium	brevium
Dative	brevibus	brevibus	brevibus
Accusative	brevēs	brevēs	brevia
Ablative	brevibus	brevibus	brevibus

- Singular form exists, but plural form has a different meaning: /auxilium/ ‘help’, /auxilia/ ‘support troops’.
- Singular and plural form exist, but plural form has an additional nonderived meaning. /littera/ ‘(alphabetical) letter’, /litterae/ ‘(alphabetical) letters’, /litterae/ ‘letter’.

Interestingly, when used with numerals, a distinction can be made between pluralia tanta and other words. Normally, the number is expressed using a so-called cardinal number; however, pluralia tanta require the distributive numbers. Thus we have /bīnae litterae/ ‘two letters’ as opposed to /duae litterae/ ‘two letters’ (in the sense of alphabetical letters). Here, /duae/ is the nominative plural of the cardinal for ‘two’, while /bīnae/ is the corresponding form of the distributive (‘two each’).

Similarly, while many verbs cannot a passivise (for example the intransitives), there also exist verbs that have *only* passive forms. These are called **deponent verbs**. An example is /hortāri/ ‘to urge’. There are also verbs with perfective morphology but presentive meaning: /meminisse/ ‘to recall’. Impersonal verbs have only forms in the third singular: /mē pudet/ ‘I am ashamed’. A handful of verbs only have a restricted set of forms: /inquam/ ‘I say’, which has only a handful of other forms.

6.2 The Verbal Paradigm: Relation Change and Verbal Agreement

The verbal paradigm is analysed in the dictionary file latin-verb.xml. The dictionary contains some verbal roots together with all affixes so that a fully inflected verb form can be generated (or analysed). The details can be looked up in that dictionary. Here we shall discuss certain design principles and choices made. Notice that from now on we do not indicate vowel length. The forms are exclusively shown the way they normally appear in print. A second note is that the division between ending and thematic vowel is not always as it is shown in the text books. This is not because of some disagreement but because of the desire to get by with the minimum amount of complication. The text book analysis can be

implemented as well, but in some cases we have chosen not to. What counts, after all, is the output form.

First, consider an inflected form. It contains a set of specifications:

- **voice**: whether it is active or passive,
- **aspect**: whether it is presentive or perfective,
- **tense**: whether it is present, past or future,
- **mood**: whether it is indicative, subjunctive, or imperative,
- **person**: whether it is 1, 2, or 3,
- **number**: whether it is singular or plural.

From a morphological point of view, tense and mood are inseparable, just as person and number. Thus, there will be combined affixes for tense/mood and person/number.

The sequence in which these affixes are added is as follows:

(6.1) voice > aspect > tense/mood > person/number

With respect to voice, this decision has been made in accordance with universal principles of affixation in verbal paradigms. It will become clear shortly why this is a good idea. Notice that from a pure concatenative point of view one could have delayed the addition of voice and instead added a combined person/number/voice affix. Instead, what has been done here is the following: a feature value for "voice" is added by the voice affix (which is however empty), and the person/number affixes have two allomorphs: one for active one for passive morphology.

However, there is more, as we shall see.

Let us begin with the root itself. The semantics may specify a number of participants, some of which will in the end become subject and object, others prepositional objects, again others are subject to adverbial modification. Let us concentrate at this point on the subject and object. As is well known, subjects of passives are objects of actives. Moreover, since subjects are in the nominative and objects in the accusative, it is not the root that will assign nominative or accusative.

Thus, as far as the immediate arguments of the verbs are concerned, the root does not select them as arguments at all.

There are two ways to go from here. The first is to let the root supply the arguments. The second, taken here, is to install two parameters, GF1 and GF2, for grammatical function 1 (subject) and grammatical function 2 (object). The semantics may further specify what type of thematic role they have. In principle, it would be possible to derive the assignment of grammatical functions from the assignment of roles plus transitivity (see Van Valin and LaPolla 1997), but we have not done so here. Instead, roots have the GFs already in place.

However, notice that roots do not assign case, at least not to their subjects and objects. Indeed, these are absent altogether from the argument structure. They will be added later. A root does however specify its transitivity. The feature TRS has two values: *true* (for transitive verbs), and *false* (for intransitive verbs).

The first round of affixes is devoted to relation change. There is only to consider here, and that is the passive. Passive may only apply to transitive verbs (marked by [TRS : *true*]). It moves the GF2 to GF1 and changes the transitivity to *false*. As a consequence, the GF1 disappears. The subject of actives become a so-called chômeur in Relational Grammar terminology (see Perlmutter 1983). The analogy with referent systems is quite striking and has been studied in Kracht 2002b. However, as has been noted in the quoted article, we need to take care of the fact that subject chômeurs behave differently from others, for example object chômeurs. The first must appear in Latin in a PP with preposition /ā/ or /ab/ plus ablative, while the second appear generally in the accusative (like ordinary objects). It is possible to solve this problem, for example by creating a special parameter to catch the GF1 after passivization.

Deponent verbs look as if passive morphology has applied already. They carry the feature [VOICE : *pass*].

Both the active and the passive morphemes are empty. They are immediately followed by the thematic vowel.

Next follow aspect, tense and mood, which we shall discuss in the next section. At the end, the personal agreement suffixes are added. Since the verb does not yet expect any subject or object arguments in its argument structure, these must now be added. The first is object agreement (AGRO). It is morphologically zero, owing to the fact that Latin does not show object agreement. However, the morpheme is

not redundant. Its role is to promote the variable for the object from a parameter (GF2) to a variable with its own AIS. That AIS contain the variable together with a specification of the argument's properties: that it must be in accusative case. It is at this point that we need to make a decision as to whether the accusative object is to the left of the verb or to its right. By consequence, as word oder is free in Latin, we have two minimally different object agreement affixes: one that expects the argument on the left, and another that expects in on the right. Notice that this multiplies the number of parses by two.

Right after comes AGRS. This agreement suffix has many different forms. The most obvious is that there are six variants, one for each person and number. Furthermore, there are forms inthe active and forms in the passive. These are needed to identify whether the verb is in the passive. This gives a total of 12 entries. Within each entry, various allomorphs must be distinguished.

1. The personal endings of the perfect active are different from the other combinations of tense, aspect and mood. Specifically, we get /`isti`/ in the second singular, and /`istis`/ in the second plural.
2. The first singular is sometimes /`m`/ and sometimes /`o`/, and sometimes /`i`/. Moreover, when it is /`o`/, some vowels are omitted (e. g. the thematic vowel of the a-conjugation). To control for this behaviour, the morphology contains an attribute OFORM with values *m*, *o* and *i*. These values are being set by the tense/mood morphemes. In the passive first singular is /`or`/ when it is /`o`/ in the active; otherwise it is /`r`/. Thus the controlling parameter is the same, only the forms are different.

6.3 Tense, Mood and Aspect

The next step is the addition of aspect. There are only two in Latin, presentive and perfective. The presentive is unmarked, both in the active and the passive. The perfective aspect however is remarkably complex.

In the thematic conjugation, the perfective is signaled by /`v`/. This follows the thematic vowel, so that we get the forms /`laudav`/, /`delev`/ and /`audiv`/. The passive perfective forms are derived from yet another stem, formed by adding a /`t`/. Thus we get /`laudat`/, /`delet`/, and /`audit`/. In the consonantal class we

need a separate listed form for both. For example, the verb /`tangere`/ 'to touch' uses /`tetig`/ for the perfective active and /`tac`/ for the perfective passive. The way this is implemented deserves special attention. In the entry, three allomorps are listed. Their morphology contains an attribute BASE with values *a* (presentive), *f* (perfective active) and *p* (perfective passive). Regular verbs have only one morph, with no value added for BASE. The forms can therefore be distinguished on the basis of this feature. The perfective aspect needs a perfective stem, it thus looks for the value of BASE. If it is *f*, then it uses that form and adds a /*v*/ for thematic verbs, and nothing for consonantal verbs. To be able to discriminate the conjugational paradigms, another morphological feature is introduced, STEM, with the following values

- *a* thematic a-conjugation,
- *e* thematic e-conjugation,
- *i* thematic i-conjugation,
- *c* consonantal conjugation,
- *ci* consonantal, short i-conjugation,
- *voc* thematic conjugation with vowel added,
- *vc* consonantal, with vowel added,
- *vci* consonantal short i-conjugation with vowel added.

Similarly, the perfective passive uses the base [BASE : *p*]. It universally adds a /`t`/ (but see the exercises).

One could have insisted that the other stems, perfect active and perfect passive, are simply a package: they are but a different form for the sequence of stem plus voice plus perfective aspect. In other words, they are idioms. Thus, the perfect active form is not analysable into any combination. However, this approach is not satisfactory. The problem is as follows. Whatever meaning the perfective aspect has, it has that meaning depending on whether the verb is regular or not. If /`tetig`/ was an idiom it needed to have the perfective meaning inbuilt, since it will not be added. By contrast, the solution here is to start with /`tetig`/ as an alternate stem and add voice and aspect to it. Since they are both empty, there

is no visible change. Contrast this with the form /memin/, which has perfective morphology without however perfective aspect. Thus we have /meministi/ 'you recall'.

This solution is necessitated also by the sequence of tense, discussed in the previous chapter. There we have seen that tenses have two uses, one direct and one in a report of an attitude. This double nature is independent of the regularity of the verb. Irregular verbs participate in it just as regular ones do.

After the aspect markers have been added we add tense and mood. From a morphological point of view, the two cannot be decomposed. Suffixes vary greatly depending on whether we are in presentive aspect, perfective active or perfective passive. We therefore get the following variety.

1. Present indicative. Zero form. First singular is in /o/.
2. Present subjunctive. Forms are /e/ (a-conjugation) and /a/ in all others. The suffix added without the vowel in the a- and the consonantal conjugation. Takes /m/ in the first singular. Thus we have /laudem/ and /deleam/.
3. Past indicative. Form is /ba/. Takes /m/ in the first singular.
4. Past subjunctive. Form is /re/. Takes /m/.
5. Future I. Forms: either of the following
    - /b/, /bi/, /bu/ (a- and e-thematic conjugation), takes /o/ in the first singular;
    - /a/ (i-thematic, consonantal and short i-consonantal conjugation). Takes /m/ in the first singular.

In the perfective aspect the forms are as follows.

1. Present indicative. Zero form. First singular is in /i/.
2. Present subjunctive. Forms are /eri/. Takes /m/ in the first singular. Thus we have /laudaverim/.
3. Past indicative. Form is /era/. Takes /m/ in the first singular.

4. Past subjunctive. Form is /isse/. Takes /m/.
5. Future I. Form: /er/ (first singular), /eri/. Takes the /o/ form. Thus /laudavero/. Is identical with the perfect subjunctive in all forms but the first singular.

Exercise 49. There are some verbs that form the PPP using /s/ in place of /t/. The PPP of /videre/ 'to see' /visum/ 'seen'. Propose a change to the dictionary that accounts for this morphological fact.

6.4 The Simple Clause

Appendix: Coding and Notation

In this appendix I will illustrate both the data structures and the way they get encoded and shown to the user. This will clarify some issues on how the system actually works. I will proceed by illustrating first the global structure and then zooming on various subparts.

The Main System

Options The file `options/defaults.xml` stores the default settings. The main file responsible for handling them is `options.ml`. The settings are stored as a single map. Many functions take such an options map as an argument. The options are responsible for:

- handling output style
- storing global dictionary options (for example, access)
- setting the global data structure

For example, there are many switches that control the way in which the structures are shown to you.

Dictionaries All actions that you request are recorded in a dialog box. If you want the sequence of actions dumped into a file, type

(6.2) `dump <return>`

and you can take a look at the dialog in the file `tmp/session.dmp`. Alternatively, if you type

(6.3) `dump` *`filename`* `<return>`

the session is recorded instead in `tmp/`*`filename`*`.dmp`. (These are ASCII files, like all others we are dealing with.)

By default, dictionaries are stored in `dict/`. When the program is started, it scans this directory and lists the files in the top right corner. You can either double click on the dictionary or type alternatively

(6.4) `read dict` *`dictname`* <return>

or

(6.5) `read dictionary` *`dictname`* <return>

The name of the directory can be changed by setting the `dictdir` option. Thus, in principle you can have any number of bundles of dictionaries, though only one is displayed at any given time.

You can deal with dictionaries in many ways. First, you can take a look at the raw file and edit this. Suppose you have a dictionary `tagalog.xml` in the current directory. Then type

(6.6) `open tagalog` <return>

This will open an editor window (with gvim) and so that you can edit the file.

After editing, you can also read the dictionary again. When you are working with a dictionary, all entries that you create on the way will be added to a temporary dictionary, called workspace. There are several commands to save a dictionary. These are

(6.7) `save dict to` *`filename`* <return>

or simply

(6.8) `save to` *`filename`* <return>

This will cause the current workspace (not just the original dictionary) to be saved in XML format (in the current directory for dictionaries).

If you decide to switch to another dictionary, type

(6.9) `clear workspace` <return>

or

(6.10) `clear workplace <return>`

This will however not cause the session dialog to be emptied. By default, the recording continues. If you want to clear everything, then type

(6.11) `clear all <return>`

Changing to another dictionary can be done in two ways. The first is to read in a new dictionary *and* eliminate the previous one. The other is by *adding* the new dictionary. The second method does not delete the old entries, it just adds new ones. This is achieved by typing

(6.12) `add dictionary` *`name-of-dictionary`* `<return>`

or

(6.13) `add dict` *`name-of-dictionary`* `<return>`

Now that you have loaded a dictionary, there are several things that can be done. Perhaps the most useful to start with is the command

(6.14) `show all <return>`

This will prompt a window to be created where all elements of the dictionary are listed by their name. All elements can be clicked on, which prompts the particular element to be displayed. If only the entries are of interest, type

(6.15) `show dict <return>`

and they will be shown (though without hyperlinks). If you want to see the entries of the entire workspace, type

(6.16) `show workspace <return>`

instead.

As was explained earlier, items can be drawn from the dictionary and put onto a stack. This stack is displayed on the left hand side. The top element can be looked at by typing

(6.17) `show last <return>`

or simply

(6.18) `show <return>`

If you want a specific element from the stack to be shown, type

(6.19) `show number` *`number-as-displayed`* `<return>`

There is also the command `parse`. You use it in the following way: type `parse`, then a space, and finally the string you want to parse enclosed in double quotes. Finally, hit <`return`>. Alternatively, type the string into the upper window and click on the button <`Parse`>. This will do the same. As a result you get a display of all parses. If you want to see the parse table, close that window and type

(6.20) `show parse <return>`

This will display the entire chart of the parse. Notice, though, that this chart looks different precisely because the chart is done over a somewhat different data structure: it uses occurrences, which are just pairs of numbers indicating the beginning and end of the substring. Also, there is no semantics. Everything is done with lists of so called *short argument identification statements* (see Section (4.6)).

This should suffice as an overview. (There are still more options, but you can always type “help” to request a list of them together with a short description.)

Items

The various items are defined in the `.ml`-files. For an item of some type with name *`item-name`* there are two functions, which are defined immediately upon the definition of the type (in the same file): *`item-name`*`_to_xml`, which takes the item and produces an XML Data structure, and the opposite function `xml_to_`*`item-name`*, which converts an XML Data structure to the appropriate item. Additionally, `latex_dump.ml` contains for all these types functions of the form *`item-name`*`_to_ltx`, which convert the XML Data structure to LaTeX-code, which serves as input to the display via PDF. Knowing this, you can basically change the layout of these elements, but remember that changes must be followed through at various places.

Glued Strings The string /dog/ is encoded as

(6.21)
```
<gsts>dog</gsts>
```

Do not insert blanks or any form of whitespace unless you want them. A sequence of 5 blanks is encoded like this:

(6.22)
```
<gsts zero="5"/>
```

This is done for technical reasons: a sequence of blanks is otherwise confused with an empty element.

Glued strings consist of a string plus left and right glue conditions. They look like this:

(6.23)
```
<gst>
    <pre pos="p1" neg="p2"/>
    <text>dog</text>
    <post pos="q1" neg="q2"/>
</gst>
```

If the string is empty, use the zero attribute on <gst>, but no text element. The elements *p1*, *p2*, *q1*, *q2* are strings, where each element is separated by a "+" from the next. Do not use blanks, they will otherwise be read as string conditions!

Pre- and postconditions can be stored independently. To do that, use the attribute `id` on the element to give it an identifier. Next time you want to use that same element, just write

(6.24)
```
<pre idref="identifier"/>
```

or, in the case of postconditions,

(6.25)
```
<post idref="identifier"/>
```

The visual rendering is somewhat different. Conditions are enclosed in square brackets, and positive requirements are given first, each prefixed by "+"; then follow the negative conditions, each prefixed by "−". The precondition is put before the string, the postconditions after.

Exponents are sequences of glued strings. They are simply coded in a tag <exp> in the natural order. Schematically, this looks as follows.

(6.26)
```
<exp>
    <gst>···</gst>
    <gst>···</gst>
        ...
    <gst>···</gst>
</exp>
```

Put as many of them as there are sections. (Clearly, there can also be <gsts> in place of the <gst>.) The visual output uses ⊗ to separate the sections from each other. Furthermore, constructing exponents with the help of merge inserts morpheme boundaries. The option `edet` determines whether and if so how they are displayed. With `edet` set to `dots` (default) they are shown as centered dots; with `edet` set to `bare` they are not shown.

Diacritics The XML code is pretty straightforward. It is

(6.27)
```
<dia name="diac"/>
```

where *diac* is a string. If the string contains `n`, the diacritic mark `noskip` is set; if the string contains `f` the diacritic mark `fusion` is set; if the string contains `d` the diacritic mark `down` is set; and if the string contains `u` the diacritic mark `up` is set. The combinations receive the following LaTeX-symbols:

(6.28)

`fusion`	`down`	`up`	Symbol
false	false	false	◊
false	false	true	▽
false	true	false	△
false	true	true	–
true	false	false	♦
true	false	true	▼
true	true	false	▲
true	true	true	–

Handlers and Duplicate Lists There are two types of handlers: general and specific. The general handlers are coded using the tag <hdlg>, the specific ones

with <hdl>. The <hdlg> has an attribute lg (length, optional) and gen, where the trivial name is given. <hdl> has an attribute lg (length of the argument) and daughters <unit>. Each of the latter in turn is a sequence of elements of the form

(6.29) <prt pos="*number*" fct="*bool*"/>

pos specifies the position, and fct is a boolean. If this boolean is true then pos="1" specifies the second (!) element of the functor (= left hand argument), and if the boolean is false, it is the second element of the argument (= right hand argument).

The following names for general handlers are recognised (here · is concatenation):

- rc (right compose) $\mathtt{rc}(\mathfrak{g}, \mathfrak{h}) := \mathfrak{g} \otimes \mathfrak{h}$.
- lc (left compose) $\mathtt{lc}(\mathfrak{g}, \mathfrak{h}) := \mathfrak{h} \otimes \mathfrak{g}$.
- rg (right glue) $\mathtt{rg}(\mathfrak{g} \otimes m, n \otimes \mathtt{h}) := \mathfrak{g} \otimes m \cdot n \otimes \mathfrak{h}$.
- lg (left glue) $\mathtt{lg}(\mathfrak{g} \otimes m, n \otimes \mathtt{h}) := \mathfrak{g} \otimes n \cdot m \otimes \mathfrak{h}$.
- rp (right push) $\mathtt{rp}(\mathfrak{g}, m{\otimes}n{\otimes}\mathtt{h}) := \mathfrak{g} \otimes m \cdot n \otimes \mathfrak{h}$.
- lp (left push) $\mathtt{lp}(m{\otimes}n{\otimes}\mathfrak{g}, \mathtt{h}) := \mathfrak{h} \otimes m \cdot n \otimes \mathfrak{g}$.

The symbols for general handlers are as follows (see latex.ml).

(6.30)

Operation	Symbol
rc	$\stackrel{\circ}{\rightarrow}$
rg	$\stackrel{\bullet}{\rightarrow}$
rp	$\stackrel{*}{\rightarrow}$
lc	$\stackrel{\circ}{\leftarrow}$
lg	$\stackrel{\bullet}{\leftarrow}$
lp	$\stackrel{*}{\leftarrow}$

For displaying specific handlers, there is an option ddet. It can have values dir, gen and num.

- Value `dir`. Show the directionality only, by using > (functor before argument) and < (argument before functor) and − (no directionality).
- Value `gen`. Show nothing.
- Value `num`. Show handlers as sequences separated by a vertical bar. These indicate the sections of the resulting glued string. Within each section, the parts are shown as follows. ◦ is a section from the functor, and • a section from the argument. Numbers over the elements indicate which part is being taken.

The identity conditions are coded by statements of the form

(6.31)
```
<dupl>
    <equals>
        <!-- Prt1 -->
        <!-- Prt2 -->
    </equals>
    <equals>
        <!-- Prt1 -->
        <!-- Prt2 -->
    </equals>
          etc
</dupl>
```

Thus, `<dupl>` contains a sequence of `<equals>`, which each contain two `<prt>`, one for each `Tuples.prt`.

Attribute Value Structures Attribute value structure are coded in the following way.

(6.32)
```
<tag
    att1="val1"
    att2="val2"
        ...
    attn="valn"
/>
```

Here, *tag* is a tag. The system uses in total four of them:

- `avi`: Input AV.
- `avo`: Output AV.
- `mi`: Input Morphological Class AV.
- `mo`: Output Morphological Class AV.

The values are of the following form.

- `top` for the totally undetermined value;
- `ast` for the empty set of values
- `idem` for the repeat of a value (see below)
- A sequence $\mathtt{v1} + \mathtt{v2} + \cdots + \mathtt{vm}$, where $\mathtt{v1}$ through $\mathtt{vm}$ are strings.

In AVIs we work with pairs of AVSs. Given a pair (A, B) of AVSs, the second element may optionally contain the value ✓, as discussed in Section 3.4 Its XML code is `idem`.

On the LaTeX side, the output is as shown in Section 3.4. Four cases arise.

1. The diacritic is $-$. Then nothing is shown. (In fact, (A, B) are obligatorily empty.)
2. The diacritic is $\triangledown/\blacktriangledown$. Then only A is shown. (B is obligatorily empty.)
3. The diacritic is $\vartriangle/\blacktriangle$. Then only B is shown. (A is obligatorily empty.)
4. The diacritic is $\lozenge/\blacklozenge$. Then the pair is shown in the form

$$\begin{bmatrix} \mathtt{att1} : \mathtt{vi1} \mapsto \mathtt{vo1} \\ \mathtt{att2} : \mathtt{vi2} \mapsto \mathtt{vo2} \\ \cdots \\ \mathtt{attn} : \mathtt{vin} \mapsto \mathtt{von} \end{bmatrix} \tag{6.33}$$

where $\mathtt{vi1}$ is the value in A of $\mathtt{att1}$, while $\mathtt{vo1}$ is its value in B. For this to work properly, the sets of attributes of A and B must by identical. However, the system gracefully interpolates missing attribute values pairs.

Morphological Structures A morph has the form:

(6.34)

```
<mor id="ident">
    <exp>
       <!-- Exponent -->
    </exp>
    <rank>
         <!-- Rank -->
    </rank>
<marg>
<!-- Sequence of Selectors -->
</marg>
</mor>
```

A selector contains five parts:

1. `mi` : Morphological In-Class
2. `mo`: Morphological Out-Class
3. `man` : Handler
4. `mdia` : Morphological Diacritics
5. `dupl` : Duplex Conditions

The pair (M,N) of morphological in-class and morphological out-class is displayed more or less as the pairs of AVSs. Morphological diacritics are ignored. Handlers have been discussed above.

Variable, Parameter AVSs Variables names have the form $\vec{x}\vec{y}$, where $\vec{x}$ is a lower case alphabetical string and $\vec{y}$ an sequence of digits. The digits represent a number. Variables unify only if their alphabetical strings are identical, since these represent types. The XML code of variables is

(6.35)

```
<var name="varname"/>
```

The so called raw code is just the sequence *varname*. A set of variables is coded as follows:

(6.36)

```
<varset elts="varlist"/>
```

where *varlist* is the concatenation of the raw names of variables with + as separator (no blanks).

Substitutions are coded in XML like this.

(6.37)

```
<subst>
    <vp in="vi1" out="vo1"/>
    <vp in="vi2" out="vo2"/>
              ...
    <vp in="vin" out="von"/>
</subst>
```

A PAVS is coded as follows.

(6.38)

```
<pavs>
    <vp name="name1" in="ivar1" out="ovar1"/>
    <vp name="name2" in="ivar2" out="ovar2"/>
                       ...
    <vp name="namen" in="ivarn" out="ovarn"/>
</pavs>
```

Here, *name1* is the kind of the parameter 1, *ivar1* is the name under which it is input (given as raw code), and *ovar2* the name under which it is output (also given as raw code). Similarly for the other parameters. If you want to omit a name, use the empty string. It will be reproduced in LaTeX as $\circ$.

The output is similar to that of AVSs. If the diacritic is ◊/♦, then the individual statements are issued in the form *name* : *ivar* $\mapsto$ *ovar*, otherwise the formt is that of a single AVS.

Argument Structures Argument structures are coded by

(6.39)

```
<argx>
    <arg>
        <!-- first AIS -->
    </arg>
    <arg>
        <!-- second AIS -->
    </arg>
        <!-- etc -->
</argx>
```

Each AIS has the following form.

(6.40)
```
<arg>
    <var name="x0"/>
    <dia name="x0"/>
    <avi/>
    <avo/>
    <pavs/>
</arg>
```

Here, of course instead <avi/>, <avo/> and <pavs/> a nonempty structure may figure as well.

The number of AISs must match the number of <ma> in an individual <marg> for any morph associated with that AVS.

In LaTeX code, the argumments are listed vertically, each line containing one AIS. The AISs are shown in different colours, to make them identifiable. (If you do not want to have the colours, set `farbe` to `false`.) An AIS is given as $\langle x : d : A :: P\rangle$, where x is the variable, d the diacritic, A the AVS (or paired AVS), and P a PAVS.

Parse Terms Normally, you do not want to write XML or LaTeX code for parse terms. It is done for you entirely. Nevertheless, it is important to understand how they look like and how they are displayed. The relevant definitions can be found in `term.ml`. Terms can be basic or complex.

- Simple terms consist of pairs $e : m$, where e is an entry identifier and m a morph identifier;
- unary operations consist of an operation name, a list of numbers, and a term;
- binary operation names consist of an operation name, a list of pairs of numbers and two terms.

The XML code is as follows. Either the term is empty

(6.41)
```
<term type="e"/>
```

or simple of the form $e : m$:

(6.42)
```
<term type="g" entry="e" morph="m"/>
```

If obtained from t using a unary operation, o with a list l, write

(6.43)
```
<term type="u" op="o" lst="l">
    <!-- subterm -->
</term>
```

The list code is a sequence of integers, separated by semicolon, enclosed in square brackets.

For a binary operation o with two subterms:

(6.44)
```
<term type="b" op="o" lst="l">
    <!-- subterm1 -->
    <!-- subterm2 -->
</term>
```

The list is given as pairs of numbers (i, j), separated by semicolon, enclosed in square brackets.

The names of binary operations and their LaTeX-symbols:

(6.45)

Operation	Access	XML Symbol	LaTeX
forward merge	G	`mgr`	$\boxplus$
forward merge	E	`mer`	$\ogreaterthan$
backward merge	G	`mgl`	$\boxminus$
backward merge	E	`mel`	$\olessthan$
forward fusion	G	`fgr`	$\boxplus_f$
forward fusion	E	`fer`	$\ogreaterthan_f$
backward fusion	G	`fgl`	$\boxminus_f$
backward fusion	E	`fel`	$\olessthan_f$
forward transformation merge	G	`tgr`	$\boxtimes$
forward transformation merge	E	`ter`	$\boxslash$
backward transformation merge	G	`tgl`	$\boxbslash$
backward transformation merge	E	`tel`	$\boxbslash$
forward transformation fusion	G	`tgr`	$\boxtimes_f$
forward transformation fusion	E	`ter`	$\boxslash_f$
backward transformation fusion	G	`tgl`	$\boxbslash_f$
backward transformation fusion	E	`tel`	$\boxbslash_f$

At present, the lists are not also output. They remain for internal use.

DRSs A DRS consists of a head section and a sequence of sub-DRSs of various kinds. It all begins like this:

(6.46)
```
<drs head="vars">
    <!-- list of clauses -->
</drs>
```

Here, *vars* lists the variables separated by + (not by blank). The clauses are of the following kind.

- Literal
- Equation
- Unary Constructor plus DRS (eg negation)
- Binary Constructor plus DRS (eg implication)
- Unary Binder-Constructor plus DRS (eg sum)
- Binary Binder-Constructor plus DRS (eg quantifier)

A *literal* is form with the tag <lit>. Inside it is a term. (There is no type regime, so a term might as well be a proposition.) A *term* has two shapes.

1. A variable: <termv name="varname"/>
2. A term:

   (6.47)
   ```
   <termf fc="name">
       <!-- list of arguments -->
   </termf>
   ```

   The list can be of any length. The arguments will appear in LaTeX in the order given, in the usual way. First the function name, in sans-serife, with an added prime (like this: now'). Then an opening bracket and a comma separated list of arguments followed by a closing bracket. If the list is empty, no brackets will be printed.

An *equation* uses the tag <eq>, which encloses two terms. This comes out as $t_1 \doteq t_2$, where t_1 is the first term and t_2 the second.

A unary constructor is coded as follows.

(6.48)
```
<undrs op="opname">
    <!-- argument DRS -->
</undrs>
```

The *opname* is the name of the operator. If you want negation, write `neg`, all other operators have to be entered just as LaTeX requires the string for the symbol. Beware: do not forget the slashes in front of the LaTeX-commands! Moreover, make each slash double. For example, if your operator is $\oplus$, then write `\\oplus`, because the LaTeX-code for that symbol `\oplus`.

A binary constructor is coded like a unary binder-constructor.

(6.49)
```
<bindrs op="opname">
    <!-- argument DRS1 -->
    <!-- argument DRS2 -->
</bindrs>
```

The *opname* is the name of the operator. This time, only `or` is recognised as primitive. All others have to be handcrafted manually from the LaTeX-code.

A unary binder-constructor is coded as follows.

(6.50)
```
<uqdrs op="opname" head="opname" >
    <!-- argument DRS -->
</uqdrs>
```

The *opname* is the name of the operator. Only `sum` is encoded primitively. `head` has as values a set of variable names, separated by `+`.

A binary binder-constructor is coded as follows.

(6.51)
```
<bqdrs op="opname" qvars="opname" >
    <!-- argument DRS1 -->
    <!-- argument DRS2 -->
</bqdrs>
```

The *opname* is the name of the operator. Only `all` is encoded primitively (the universal quantifier). `qvars` has as values a set of variable names, separated by `+`.

Symbols

$\frown$, 36
$\llcorner\lrcorner$, 36
ε, 36
$\curlywedge$, 40
[ ], 47
$\wedge$, $\vee$, $\top$, $\equiv$, 47
$\perp$, 48
$\leq$, 49
f^{+}, 49
$\star$, 51
$\mathfrak{z}$, $\otimes$, 54
$\bigotimes$, 55
$\cup$, $\neg$, $\Rightarrow$, $\vee$, 75
$\sim_{v}$, 76
$s(\Delta)$, 80
ℓ_1, ℓ_2, 80
$\vartriangle$, $\triangledown$, $\Diamond$, 91
∂, $\flat$, $\sharp$, 91
$\checkmark$, 96
$[y/x]$, 100
::, 153
$\circ$, 158

Index

$\alpha \approx \beta$, 152
$\alpha \vdash \beta$, 160
▼, ♦, 142
$\mathcal{S}$-matrix, 49

abstract incorporation, 165
abstract syntax, 91
access, 104
accessibility, 76
    direct, 76
adjunct, 92
adverbial
    absolute, 202
    event relative, 202
    utterance relative, 202
AIS, 91
appropriateness function, 158
argument, 92, 102
argument handling statement, 97
argument identifiation statement
    empty, 97
argument identification statement, 91, 97
argument structure, 73, 84, 97
    saturated, 106
attribute value matrix, 47
AVM, 47
    empty, 47

base form, 44
body, 84

carrier, 92
category, 111
class
    morphological, 46
combinatorics, 37
concatenation, 36
consecutio temporum, 200
contiguity, 39
Copy AVM, 96

diacritic, 91
diacritic mark, 91
    constituent, 91
    vertical, 91
dimension, 66
directionality, 59
discourse representation structure, 75
Discourse Representation Theory, 72, 75
DRS (see discourse representation structure), 75
Dutch, 136
Dyirbal, 130

E-access, 104
E-preemption, 86
empty string, 36
English, 90, 104, 136, 200

entry, 68
equivalence, 96
exponent, 37
export function, 84
exposure, 133

feature installment, 118
feature space, 49
Finnish, 195
form
  base, 44
  neutral, 44
French, 176
functor, 102
fusion, 106
  adicity, 108
  monadic, 106, 107
  polyadic, 108

G-access, 104
Georgian, 195
German, 104, 114, 123, 124, 136, 176, 195, 200
glued string, 38
  fractured, 54
  occurrence, 39
Greek, 176

handler, 55
  consumptive, 59
  dimension, 57
  leftward, 59
  linear, 61
  neutral, 59
  proper, 57
  rightward, 59
handlers
  product, 58
head, 92

Hungarian, 36, 41, 200

I-preemption, 86
identified
  minor, 68
identifier, 68
  major, 68
import function, 84
individual level property, 172
item-and-arrangement, 36
item-and-process, 36

Jiwarli, 130

Latin, 123, 129, 149, 176
lexicalism, 37

matching, 152
meaning, 37
merge, 80, 106
  adicity, 107
  monadic, 106, 107
  polyadic, 107
mode, 68
Mordvin, 118
morph, 66
  dimension, 66
morpheme, 67
morphological class
  fully specified, 65
  ingoing, 46
  outgoing, 46

N-system, 84
name, 84
  export, 84
  import, 84
neutral form, 44
Nootka, 184

occurrence, 39
orientation, 199
  object, 199
  subject, 199
overexposure, 132
overlap, 39

pairing function, 109
parameter, 163, 169
parameter handling statement, 173
  duplex, 173
  simplex, 173
PHS, 173
pivot, 109
product, 58
proximity, 189

range, 48
referent, 85
  consumed, 92
referent system, 72, 84, 85
requirement, 38
  left, 38
  negative, 38
  positive, 38
  right, 38

Sanskrit, 129
section, 54
selector, 65, 74
sequence of tense, 200
sign, 37, 101
stage level property, 172
string, 35
  glued, 38
subsumption ordering, 160
supervenience, 86
surface orientation, 36

tectogrammar, 91
transformation, 92
truth, 76
type 0, 158
type 1, 158
typing function, 158

unfolding, 69
unification, 51
union, 76

variable
  bound, 76
  free, 76

Bibliography

Abusch, Dorith. “Sequence of Tense and Temporal *De Re*”. In: *Linguistics and Philosophy* 20 (1997), pp. 1–50.

Bauer, Laurie. *Introducing Linguistic Morpphology*. 2nd ed. University of Georgetown Press, 2003.

Becker, Tilman, Owen Rambow, and Michael Niv. *The Derivational Generative Power of Formal Systems or: Scrambling is Beyond LCFRS*. Tech. rep. IRCS Report 92–38. The Institute For Research In Cognitive Science, University of Pennsylvania, 1992.

Blake, Barry J. *Case*. Cambridge Textbooks in Linguistics. Cambridge University Press, 1994.

Calcagno, Mike. “A Sign–Based Extension to the Lambek Calculus for Discontinuous Constituents”. In: *Bulletin of the IGPL* 3 (1995), pp. 555–578.

Comrie, Bernhard. *Tense*. Cambridge Textbooks in Linguistics. Cambridge University Press, 1985.

Corbett, Greville. *Gender*. Cambridge Textbooks in Linguistics. Cambridge: Cambridge University Press, 1991.

Corbette, Greville. *Features*. Cambridge Textbooks in Linguistics. Cambridge University Press, 2012.

— *Number*. Cambridge Textbooks in Linguistics. Cambridge University Press, 2000.

Dixon, R. M. W. *The Australian Languages. Their nature and development*. Cambridge: Cambridge University Press, 2002.

Fähnrich, Heinz. *Kurze Grammatik der georgischen Sprache*. Leipzig: Langenscheidt, 1993.

Farkas, Donka and Henriëtte de Swart. *The semantics of incorporation: from argument structure to discourse transparency*. Stanford: CSLI, 2003.

Fine, Kit. *Semantic relationism*. London: Blackwell, 2007.

Fromkin, V., ed. *Linguistics: An Introduction to linguistic theory*. London: Blackwell, 2000.

Gazdar, Gerald et al. *Generalized Phrase Structure Grammar*. Oxford: Blackwell, 1985.

Gillon, Brendan. “Word order in classical Sanscrit”. In: *Indian Linguistics* (1996).

Groenendijk, Jeroen and Martin Stokhof. “Dynamic Montague Grammar”. In: *Papers from the Second Symposium on Logic and Language*. Ed. by L. Kálmán and L. Pólos. Akademiai Kiadó, 1990, pp. 3–48.

— “Dynamic Predicate Logic”. In: *Linguistics and Philosophy* 14 (1991), pp. 39–100.

Haider, Hubert. *Deutsche Syntax — generativ. Vorstudien zur Theorie einer projektiven Grammatik*. Tübingen: Gunter Narr Verlag, 1993.

Hausser, Roland R. *Surface Compositional Grammar*. München: Wilhelm Finck Verlag, 1984.

Jackendoff, Ray. *$\overline{X}$–Syntax: A Study of Phrase Structure*. Linguistic Inquiry Monographs 2. Cambridge (Mass.): MIT Press, 1977.

Kamp, Hans and Uwe Reyle. *From Discourse to Logic. Introduction to Modeltheoretic Semantics of Natural Language, Formal Language and Discourse Representation*. Dordrecht: Kluwer, 1993.

Keenan, Edward L. “On Semantics and Binding Theory”. In: *Explaining Language Universals*. Ed. by John A. Hawkins. 1988.

Keenan, Edward L. and Leonard L. Faltz. *Boolean Semantics for Natural Language*. Dordrecht: Reidel, 1985.

Keresztes, László. *Chrestomathia Mordvinica*. Budapest: Tankönyvkiadó, 1990.

Klein, Wolfgang. *Time in Language*. London: Routledge, 1994.

Kornai, András. *On Hungarian morphology*. Budapest: Hungarian Academy of Sciences Institute of Linguistics, 1994.

Kracht, Marcus. “Against the Feature Bundle Theory of Case”. In: *New Perspectives on Case and Case Theory*. Ed. by Eleonore Brandner and Heike Zinsmeister. Stanford: CSLI, 2002.

— “Against the feature bundle theory of case”. In: *New Perspectives on Case Theory*. Ed. by Ellen Brandner and Heike Zinsmeister. CSLI, 2003, pp. 165–190.

— *Interpreted Languages and Compositionality*. Studies in Linguistics and Philosophy 89. Springer, 2011.

— “Mathematical Aspects of Command Relations”. In: *Proceedings of the EACL 93*. 1993, pp. 240–249.

— *Mathematics of Language*. Berlin: Mouton de Gruyter, 2003.

— “Referent Systems and Relational Grammar”. In: *Journal of Logic, Language and Information* 11 (2002), pp. 251–286.

Lass, Roger. *Phonology. An Introduction to Basic Concepts*. Cambridge University Press, 1984.

Matthews, P. H. *Inflectional morphology. An introduction to the theory of word-structure*. Cambridge Textbooks in Linguistics. Cambridge University Press, 1978.

Mel’cuk, Igor. *Cours de Morphologie Générale*. Vol. 1 – 5. Les Presses de l’Université de Montréal, 1993 – 2000.

Michaelis, Jens. “On Formal Properties of Minimalist Grammars”. PhD thesis. Universität Potsdam, 2001.

Müller, Stefan. *Deutsche Syntax deklarativ: Head-Driven Phrase Structure Grammar für das Deutsche*. Linguistische Arbeiten 394. Tübingen: Max Niemeyer Verlag, 1999.

Ogihara, Toshiyuki. *Tense, Attitudes and Scope*. Dordrecht: Kluwer, 1996.

Perlmutter, David M. *Studies in Relational Grammar, Vol. 1*. Chicago and London: The Chicago University Press, 1983.

Pesetzky, David. *Zero Syntax. Experiencers and Cascades*. Current Studies in Linguistics 27. MIT Press, 1995.

Plank, Frans. “(Re-)Introducing Suffixaufnahme”. In: *Double Case. Agreement by Suffixaufnahme*. Ed. by Frans Plank. Oxford University Press, 1995, pp. 3–110.

Randriamasimanana, Charles. “A Study of Causative Constructions in Malagasy”. PhD thesis. University of Southern California, Los Angeles, 1981.

Sapir, Edward. *Language: an introduction to the study of speech*. New York: Harcourt, Brace & World, 1921.

Stabler, Edward P. “Derivational Minimalism”. In: *Logical Aspects of Computational Linguistics (LACL ’96)*. Ed. by Christian Retoré. Lecture Notes in Artificial Intelligence 1328. Heidelberg: Springer, 1997, pp. 68–95.

Steedman, Mark. “Gapping as Constituent Coordination”. In: *Linguistics and Philosophy* 13 (1990), pp. 207–263.

Van Valin, Robert D. and J. LaPolla Randy. *Syntax. Structure, meaning and function*. Cambridge Textbooks in Linguistics. Cambridge University Press, 1997.

Vermeulen, Kees F. M. “Merging without Mystery or: Variables in Dynamic Semantics”. In: *Journal of Philosophical Logic* 24 (1995), pp. 405–450.

Visser, Albert and Kees F. M. Vermeulen. “Dynamic bracketing and discourse representation”. In: *Notre Dame Journal of Formal Logic* 37 (1996), pp. 321–365.