Users Guide for the ZAS Database of Clause-Embedding Predicates

The ZAS-PB3 Database Team

February 16, 2017

Contents

Publikationsserver des Leibniz-Instituts für Deutsche Sprache
URN: http://nbn-resolving.de/urn:nbn:de:bsz:mh39-97896

1 Quick start

The ZAS database documents how lexical predicates interact with clausal complementation.

- Contains over **1700 predicates** in contemporary German
- Shows what kinds of clauses they can embedded, with what kinds of grammatical properties
- Through nearly **17000 naturally occurring examples** taken from corpora

The database is thus built around two large tables, which are linked to each other:

The Predicate table contains an entry for each predicate, coded for its grammatical properties, plus links to the examples that demonstrate its use.

The Example table contains an entry for each example, coded for its grammatical properties, plus a link to the predicate it demonstrates.

- You can search in either table, i.e. you can search for examples or for predicates.
- You can specify in great detail what kind of properties the examples or predicates you're interested in should have.
- And you can combine properties from both tables for a single search. So you can search for predicates which have examples which in turn have some set of properties, and vice versa.

Here are the properties predicates and examples are coded for, with links to more information:

Predicate Properties	**Description**	**Info**
predicate	base form of the predicate, usually infinitive	7.1.1
pred. ID	unique number identifying each predicate	7.1.2
category	is the predicate a verb or something else?	7.1.3
morphology	the derivational-morphological make-up of the predicate	7.1.4

Example Properties	**Description**	**Info**
ex. ID	unique number identifying each example	7.2.1
example type	what type of embedding does the example show?	7.2.2
predicate	the predicate being demonstrated	7.2.3
example text	full text of the example	7.2.4
source	info on the corpus source of the example	7.2.5
complementizer	the complementizer that introduces the embedded clause	7.2.6
finiteness	is the embedding finite, an infinitive or a nominalization?	7.2.7
word order	is the embedding V2, verb-final?	7.2.8
semantics	is the embedding an assertion, a question?	7.2.9
verb mood	is the embedding indicative, subjunctive?	7.2.10
definitness	for nominalizations: definite, indefinite, bare?	7.2.11
controler	for infinitives: what controls the null subject?	7.2.12
control shift	for infinitives: is there control shift?	7.2.13
arg. structure	abstract argument structure of the embedding predicate	7.3
arg. realization	concrete realization of the arguments	7.3

Figure 1 shows the basic search interface, labeled with numbers corresponding to the items in the description below. See 8.1 for more detailed information on basic searches.

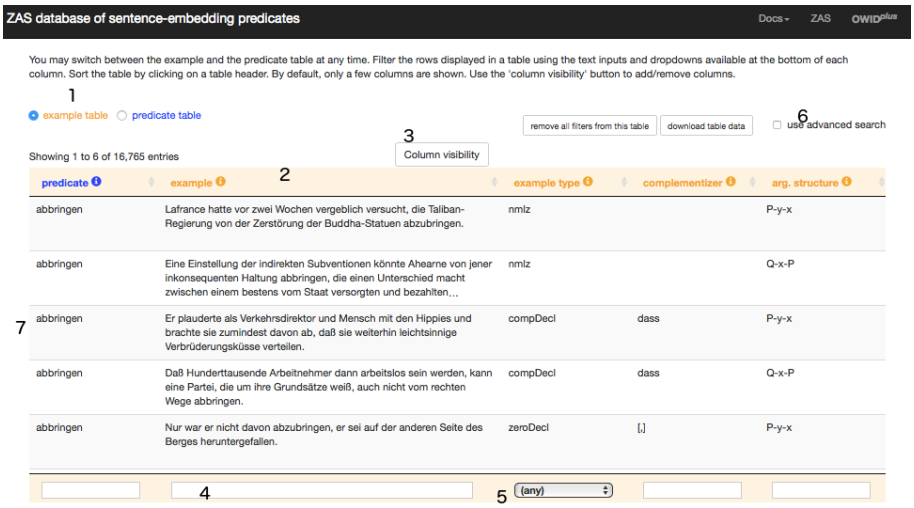

Figure 1: Basic Search

1 Do you want to search for examples or for predicates? See 5.2, 8.1.7 and 9.3 on the cool and complicated aspects of searches on the predicate table.

2 This header shows the properties currently displayed. Blue ones are predicate properties, orange ones are example properties. Click on a property to sort the table by its values.

3 Click here to change which properties are displayed. If you're interested in e.g. **verb mood**, you can select it here, and you can make e.g. **example type** invisible to make room. See 8.1.2

4 This is a text box. Enter what you want to be contained in the relevant property, and the list of results will be updated in real time. For some properties, you will get suggestions as you type. See 8.1.8 if you want to do fancy stuff here, including regular expression searches.

5 This is a pull down. Just select the value you want for properties like this.

6 Click here to build an advanced search, with arbitrary boolean combinations of the various properties. See 8.2 and 9 on how to use it to its full potential.

7 This is the list of entries in the table that match the current search, constantly updated as you enter things for the various properties. Double-click on a row for complete information.

Here's how how to reproduce an example with correct ID (7.2.1) and Source (7.2.5) information and how to cite the database (see 10 for details):

(1) Aber ich ahne, es wird nicht mehr als Blech. (ZDB 256: IDS brz 2006)

Stiebels, Barbara, Thomas McFadden, Kerstin Schwabe, Torgrim Solstad, Elisa Kellner, Livia Sommer and Katarzyna Stoltmann. 2017. *ZAS Database of Clause-embedding Predicates*, release 0.2 (Public Beta) in: OWID^{plus}, hg. v. Institut für Deutsche Sprache, Mannheim, http://www.owid.de/plus/zasembed2017.

2 Overview

The ZAS database grew out of the desire to document and classify the clausal complementation patterns of clause-embedding predicates on a larger scale. In the linguistic literature, the discussion of clausal complementation types and their licensing predicates usually centers around a few clause-embedding predicates characteristic for the respective type of clausal complement (e.g., Karttunen's 1977 list of question-embedding predicates). Extending the set of predicates to be checked for their licensing of specific complementation types helps to determine whether generalizations on licensing put forth in the literature can be maintained. In addition, no study on clausal complementation has taken into account the predicates' full arrays of admissible clausal complements. The impressive study by Levin (1993) on verb classes in English did not consider clause-embedding predicates and their complements. However, Levin's structural approach to verb classes, which defines verb classes in terms of admissible verbal alternations, was taken as a methodological stimulus for the ZAS database project (see Stiebels 2011 for a short motivation). Instead of admissible alternations, which are less informative for clause-embedding predicates, the array of admissible clausal complements was taken as a potential class-defining feature of clause-embedding predicates. This structural approach to verb classes is meant as an alternative to traditional verb-field approaches.

The ZAS database documents the clausal complementation pattern of over **1700 predicates** in contemporary German. It considers infinitival and nominalized complements as non-finite complements and *dass*-clauses, embedded verb-second clauses, and interrogative complements (embedded polar and wh-questions) as finite complements. It also includes minor types such as argument conditionals (introduced by *wenn* 'if'), and is being expanded for later releases to cover parentheticals and direct speech embeddings.

The list of predicates in the database is not intended to be exhaustive; it could be easily extended by adding e.g. further particle or prefix verb derivatives. However, based on the impression that the list of predicates already yields a quite representative picture of clausal complementation in the language, we have ceased the large-scale addition of further predicates in favor of increasing the depth of data on each predicate. New predicates are still added periodically, but our primary efforts in recent years have been devoted to expanding the collection of examples to better approximate exhaustivity in the coverage of types of embedding behavior, and to improving the coding of relevant properties in existing examples.

The development of the database was guided by the following objectives:

- The usage of the predicates should not be exemplified by made-up examples but illustrated by naturally occurring corpus examples.
- Each predicate should be checked in all its meaning variants and valency patterns.
- Properties that appeared relevant for specific complementation types should be checked systematically (e.g. the indicative/subjunctive distinction in embedded verb-second clauses, the controller choice in infinitival complements, and the definiteness of the nominal head in nominalized clausal complements).

The usage of the predicates in terms of clausal complementation is documented by corpus examples mainly taken from the corpus *Digitales Wörterbuch der Deutschen Sprache* (DWDS; http://www.dwds.de), the corpus *Deutsches Referenzkorpus* (DeReKO; http://www1.ids-mannheim.de/

kl/projekte/korpora/) at IDS Mannheim and from German belles lettres (20th/21st century), usually referenced by the respective book's ID in the catalogue of the *Deutsche Nationalbibliothek* (http://www.dnb.de/DE/Home/home_node.html). Section 4 of this guide gives complete information on the sources used.

The database only exemplifies and encodes "surfacy" features in order to keep the annotation simple. The predicates are encoded with respect to their lexical category and their morphology. The examples are encoded throughout for the argument structure and argument realization of the clause-embedding predicate and the example type (e.g., infinitival complement, nominalized complement). Furthermore, the finiteness of the embedded clause, the complementizer – if present – and the position of its predicate in the clause are specified. For finite complements, verb mood is specified. Infinitival complements are encoded for controller choice and control shift, nominalized clausal complements for the definiteness of the nominal head. See Section 7 for a full description of the properties coded in the database.

3 A short history of the database

The ZAS database was gradually built up and extended in various research projects located at the Leibniz-Zentrum Allgemeine Sprachwissenschaft (ZAS) in Berlin (http://www.zas-berlin.de). It was initiated and conceived by Barbara Stiebels as a research tool in her DFG project *Typologie der Kontrollverben: Kohärenz, Strukturbedingungen und lexikalische Klassen* (STI 151/2-2; 2003-2005). The goal was to study the control properties of a larger number of infinitive-embedding predicates in German with examples of their use (in this phase, a collection of about 400 predicates was compiled); the examples were coded for controller choice and argument structure and argument realization of the respective clause-embedding predicate. In the successor project — the subproject P15 *Satzeinbettende Präadikate* (2006-2007) of the DFG-funded research program of ZAS — further complementation types (mainly finite *dass*-clauses) were integrated as well als additional predicates added (yielding a number of about 1550 predicates). In addition, the decision was made to consistently search for relevant corpus examples and replace examples not coming from corpora; however, corpus search and coding was not fully completed for all predicates collected so far. Starting with the BMBF funding of ZAS in 2008, a greater work force was put in the development of the database. As a tool of the project area *Lexikalische Konditionierung syntaktischer Strukturen: Satzeinbettende Prädikate* (PB3), it was re-implemented a full-fledged relational database — also with the aim in mind of integrating other languages and historical stages of German. In addition, a PHP-based database interface was created and modified several times. The collection of clause-embedding predicates to be coded was further extended to the current set of over 1700. Moreover, embedded verb-second complements, interrogative complements, nominalized complements, argument conditionals and — to a lesser degree — direct speech complements and parentheticals were studied and integrated. During the whole period, the database has been constantly modified, updated and expanded in terms of data (inclusion and exclusion of predicates and examples) and coding.

In November 2012, Barbara Stiebels left ZAS to take up a professorship at the University of Leipzig. Thomas McFadden replaced her as coordinator of PB3 and leader of the database project in January 2014, but Prof. Stiebels has remained involved in the project as an external consultant. In mid-2014, a collaboration was initiated with Carolin Müller-Spitzer and Peter Meyer in the Abteilung Lexik of the Institut für Deutsche Sprache (IDS) in Mannheim, with the goal of

making the database publicly available on the OWID^{plus} platform. The new search interface for the database on OWID^{plus} was then programmed by Dr. Meyer in consultation with the ZAS-PB3 team. The current public release of the database contains the contemporary German part, which is by far the largest. Additions are planned for future releases of data on other languages and historical stages of German.

The following people worked in the compilation, extension, and technical implementation of the database at ZAS (in the order of joining the project): Barbara Stiebels, Edmund Pohl, Kerstin Schwabe, Julia Richling, Łukasz Jędrzejowski, Kilu von Prince, Thomas McFadden, Torgrim Solstad, and Katarzyna Stoltmann as researchers; Inga Steinmann, Stephanie Troyke-Lekschas, Sina Zariess, Elisa Kellner (later as researcher), Simon Blum, Johannes Mursell, Tsenguun Bolor, Noemi Geiger, Marianna Patak, Livia Sommer (later as researcher), Jana Bajorat, Gediminas Schüppenhauer, and Sybille Kiziltan as student assistents; Vincent Fahrenholz and Patrick Kudla as technical staff.

4 Where and how the data were collected

The natural language data you find in the database in the form of the example sentences were collected from various linguistic corpora. A corpus is a digital collection of naturally occurring language data (e.g. newspaper articles), which have been analyzed with respect to certain linguistic features, e.g. part of speech, and prepared in such a way as to facilitate searching.

4.1 Sources used

The corpus examples included in the portion of the database that is currently publicly available represent written contemporary German, which we define as between 1900 and the present. The corpora used in this database, listed by the abbreviations we employ, are the following:

IDS: Deutsches Referenzkorpus (DeReKo), http://www1.ids-mannheim.de/kl/projekte/korpora/

DeWaC: DeWaCKorpus, http://wacky.sslmit.unibo.it/doku.php?id=corpora, accessible via the interface "CQP" provided by the Humboldt-University Berlin: https://korpling.german.hu-berlin.de/cqpwi/login.php

DWDS: DWDS-Korpus, http://www.dwds.de

TIGER: Tiger-Korpus, http://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/tiger.html

Taken together these corpora contain more than 33 billion "tokens", i.e. words, word parts or punctuation marks. The examples in our database were collected via queries designed to automatically filter the data according several criteria, searching for particular words or patterns in order to yield smaller collections of data that could be examined in detail. Our annotators then sifted through these smaller sets of data to find the specific sentences you now see as examples in the database. E.g. if we were looking for examples of verb-final complement clauses with the predicate *sagen*, we might run automated queries to give a collection of cases where a form of *sagen* appears in the vicinity of the complementizer *dass*. The annotator would then look through these

cases to find ones where *dass* does in fact introduce a clause that is the complement of *sagen*, and where other properties of interest obtain.

Each example sentence in the database can be traced back to the corpus it was found in via the **source** tag. This tag is visible in the detailed view of a record in the example table, which can be brought up by double clicking on an example. In addition to the corpus, the **source** provides information about where the example was collected by the creators of the corpus, e.g. the specific newspaper and the year it was published. Those three data points are always indicated in the form 'Corpus Source Year'. So example number 12617, "Ich habe es mir abgewöhnt, zu solchen Spekulationen Stellung zu nehmen", has the **source** tag *DWDS BZ 2004*. This means it is from the DWDS corpus, where it was originally taken from a 2004 edition of the newspaper BZ (short for "Berliner Zeitung"). See Section 7.2.5 of this guide for complete information on the sources, including a list of the abbreviations used.

In the early stages of the creation of the database, searches of Google Books were used to find additional examples (in particular when specific configurations proved hard or impossible to find in the corpora). Since Google Books is not a proper corpus and does not provide a stable way for us to trace examples back to their original sources, we have tried to provide all such examples with a reference number from the Deutsche Nationalbibliothek (German National Library, http://www.dnb.de). This institution implements the mandate to collect, archive and document all German and German-language publications and assigns every book a unique reference number, which also serves as a permanent link. Such examples are indicated with a **source** like *DNB 1999 S14 957340567*, where DNB stands for Deutsche Nationalbibliothek, followed by the year of publication, the page number and finally the reference number. Each reference number can lead you to the particular publication by entering http://d-nb.info/ followed by the reference number, resulting in a URL like http://d-nb.info/957340567. Nevertheless, there are some examples (at present 76 in total) still in the database from Google Book Search, for which a reference number from the DNB could not be found. Because of their unique illustration of embedding behaviours for certain predicates, which we have not been able to replicate with examples in other corpora, we have decided to keep them. Their **source** tags take the form of GBS (for Google Book Search), followed by the name of the author and the year of publication.

4.2 List of Abbreviations

The following tables list and explain all abbreviations regarding data sources used in the database, organized according to the corpus involved.

- DWDS – Digitales Wörterbuch der Deutschen Sprache (digital dictionary of the German language) http://www.dwds.de

Abbreviation	Source
K-Be	Kernkorpus Belletristik / Corpus of fictional texts
K-Ge	Kernkorpus Gebrauchsliteratur / Corpus of functional literature
K-Wi	Kernkorpus Wissenschaft /Corpus of scientific literature
K-Ze	Kernkorpus Zeitung / Corpus of newspaper articles
PNN	Potsdamer Neueste Nachrichten (no longer available in DWDS)
TS	Tagesspiegel
Zeit	Zeit
BZ	Berliner Zeitung
DDR	DDR-Korpus
Blogs	Blog-Korpus
C4	C4-Korpus (Integration of the four corpora DWDS, ACC (Austrian Academy Corpus), the Swiss text corpus and the corpus South Tyrol)

- IDS – Deutsches Referenzkorpus http://www1.ids-mannheim.de/kl/projekte/korpora/

Abbreviation	Source
bmp	Berliner Morgenpost
brz	Braunschweiger Zeitung
bvz	Burgenländische Volkszeitung
bzk	Bonner Zeitungskorpus
cz	Computer Zeitung
div	Belletristik des 20. und 21. Jahrhunderts: Diverse Schriftsteller
dpr	Die Presse
foc	FOCUS
frr	Frankfurter Rundschau
fsp	Fachsprachen-Korpus 1
goe	Goethes Werke
haz	Hannoversche Allgemeine
hbk	Handbuch-Korpora
hmp	Hamburger Morgenpost
hz	Hohenzollerische Zeitung
klz	Kleine Zeitung
lim	LIMAS-Korpus
ltb	Luxemburger Tageblatt
mk1	Mannheimer Korpus 1
mm	Mannheimer Morgen
nd	Neues Deutschland (no longer available in the corpus)
news	NEWS
nku	Nordkurier
nkz	Neue Kronen-Zeitung
nnn	Norddeutsche Neueste Nachrichten
non	Niederösterreichische Nachrichten
nun	Nürnberger Nachrichten
nuz	Nürnberger Zeitung
nzs	Neue Zürcher Zeitung am Sonntag

nzz	Neue Zürcher Zeitung
oon	Oberösterreichische Nachrichten
pbb	Plenarprotokolle (Landtag Brandenburg)
pbe	Plenarprotokolle (Abgeordnetenhaus Berlin)
pbt	Plenarprotokolle (Deutscher Bundestag)
pbw	Plenarprotokolle (Landtag von Baden-Württemberg)
pby	Plenarprotokolle (Bayrischer Landtag)
phb	Plenarprotokolle (Bremische Bürgerschaft)
phe	Plenarprotokolle (Hessischer Landtag)
phh	Plenarprotokolle (Hamburgische Bürgerschaft)
pmv	Plenarprotokolle (Landtag Mecklenburg-Vorpommern)
pni	Plenarprotokolle (Landtag Niedersachsen)
pnn	Postdamer Nachrichten
pnw	Plenarprotokolle (Landtag Nordrhein-Westfalen)
pp	Plenarprotokolle (collective abbreviation for various legislative records)
prf	profil
prp	Plenarprotokolle (Landtag Rheinland-Pfalz)
psh	Plenarprotokolle (Landtag Schleswig-Holstein)
psl	Plenarprotokolle (Landtag des Saarlandes)
psn	Plenarprotokolle (Sächsischer Landtag)
pst	Plenarprotokolle (Landtag von Sachsen-Anhalt)
pth	Plenarprotokolle (Thüringer Landtag)
rei	Reden und Interviews
rhz	Rhein-Zeitung
sbn	Salzburger Nachrichten
sgt	St. Galler Tagblat
soz	Die Südostschweiz
spk	spektrumdirekt
sz	Süddeutsche Zeitung
taz	die tageszeitung
ts	Der Tagesspiegel
ttz	Tiroler Tageszeitung
van	Vorarlberger Nachrichten
vdi	VDI Nachrichten
wam	Belletristik des 20. Jahrhunderts: Martin Walser
wdd	Wikipedia Diskussionen
wkb	Wendekorpus/West
wkd	Wendekorpus/Ost
wpd	Wikipedia Artikel
ww	Weltwoche
zeit	Die Zeit
zta	Zürcher Tagesanzeiger

4.3 Basic principles for inclusion of examples

As it is the goal of the database to document as completely as possible the embedding behavior of a substantial list of predicates, the corpora were searched for examples of each predicate with a series of embedding types, displaying an array of different properties, as well as possible variations in the argument structure of the embedding predicate and how the arguments are realized. From here on out we will refer to the embedding types as **example types**, a type of information that plays an important role in the organization of the database, and is described in detail in Section 6. The properties searched for vary depending on the **example type**, and the **example types** themselves are, to a certain extent, language specific, as languages differ e.g. in their inventory of non-finite verb forms that can head clause.

The **example types** relevant for the contemporary German part of the database are those with embedded verb-second clauses (i.e. declarative clauses without a complementizer, therefore annotated as *zeroDecl*), embedded verb-final clauses (declarative clauses with complementizers, hence *compDecl*), embedded questions (*interr*), infinitive clauses (*inf*), nominalizations (*nmlz*), direct speech clauses (*dSpeech*) and parentheticals (*paren*). The collection and coding of *dSpeech* and *paren* examples is still work in progress, thus they are not included in the current public release. They will, however, be included in future versions. The following paragraphs describe how examples were searched for for each of the **example types**, and provide some concrete examples.

If a predicate embeds *zeroDecl* clauses, i.e. embedded verb second, an exhaustive search was done with regards to the **verb mood** (indicative, subjunctive I or II, see Section 7.2.10) of the embedded clause. An example is given in (2). Note that all examples from the database are reproduced here with information about the source, in the format that we recommend, as described in Section 10.2.

(2) Predicate: *argumentieren* 'argue'

a. Subjunctive II
Eltern argumentieren meistens damit, sie stünden unter Zeitdruck. (ZDB 896: DWDS BZ 1999)
'Parents usually argue that they are under time pressure.'

b. Indicative
Nun mag man naiv argumentieren: Wer kocht schon ein Präparat zusammen und lässt es an Menschen ausprobieren, wenn er es nicht später einmal verkaufen will. (ZDB 894: DWDS Zeit 1967)
'One might argue naively: Who is concocting a medical product and has it tested on humans, if he doesn't want to sell it later on.'

For *compDecl*, i.e. finite declarative clauses with a complementizer, the exhaustive search took into account **verb mood**, but also embedded sentences with specific complementizer forms (for German dass, 'that', and wenn, 'if', see Section 7.2.6) as seen in (3).

(3) a. Predicate: *arbeiten* 'work'
dass-complementizer
Und er arbeitet dafür, dass es so bleibt. (ZDB 16928: DWDS TS 2005)
'And he works so that it will remain so.'

b. Predicate: *besorgen* 'worry'

wenn-complementizer
Es besorgt mich nicht, wenn wir das alleine erledigen müssten. (ZDB 25106: DWDS TS 2003)
'It does not worry me if we have to do it alone.'

For *interr*, i.e. embedded questions, the search included **verb mood** and **complementizer**, with the addition of looking for both *wh*- and polarity-questions. Examples of both from the database are given in (4).

(4) a. Predicate: *erinnern* 'remember'
Sie kann sich nicht erinnern, ob sie an ihrem ersten Tag auch so schnell gegangen ist. (ZDB 4237: DWDS BZ 2000)
'She cannot remember, whether she left that fast on her first day too.'
b. Predicate: *festlegen* 'determine'
Die Parteien legten nicht fest, wer künftig den Markt reguliert. (ZDB 4864: TIGER)
'The parties did not determine who would regulate the market in the future.'

For *inf*, searches were made for different types of controller (see 7.2.12), as well as for control shift (see 7.2.13). An example from the database is given in (5).

(5) Predicate: *gratulieren* 'congratulate'
Ich gratuliere Lexer dazu, in der jetzigen Situation nicht ans Aufgeben zu denken. (ZDB 5369: IDS klz 2000)
'I congratulate Lexer on not thinking about giving up in the present situation.'

For the **example type** *nmlz*, i.e. nominalizations, examples were sought with both definite and indefinite articles for each predicate, as shown in (6-a) and (6-b), respectively (see 7.2.11).

(6) a. Predicate: *rühren* 'touch sb.' (emotionally)
Die Verleihung der Ehrenbürgerschaft rührt ihn. (ZDB 7475: IDS non 2008)
'The awarding of the honorary citizenship touches him.'
b. Predicate: *anbieten* 'offer'
Die Lufthansa bot den Passagieren eine kostenlose Umbuchung auf andere Flüge oder Bahnfahrten an. (ZDB 320: DWDS BZ 2001)
'Lufthansa offered the passengers a free transfer to other flights or rail journeys.'

For some predicates, no example with an indefinite nominalization was found in any of the corpora, and in such cases an attempt was made to search for examples with no article. Note that this is currently work in progress, and the data are not yet complete on this point, so one should not draw any conlcusions from the **lack** of a certain type of example in the database. The example given in (7) illustrates the embedding of the predicate *alamieren* 'alert' for direct speech. (Note that this example is not included in the current release, as it belongs to **example type** *dSpeech*.)

(7) Predicate: *alarmieren* 'alert'
Er alarmierte seine Eltern: "Ein Dinosaurier ist in unserem Garten!" (ZDB 268: DWDS Zeit 1997)
'He alerted his parents: "A dinosaur is in our garden!"'

For parentheticals, an example is given in (8). Although in the example only a short parentheses is given, it could also be that the insertion consists of more words or even an entire sentence. (Note that this example is not included in the current release, as it belongs to **example type** *paren*.)

(8) Predicate: *preisgeben* 'disclose'
Selbst die Stellung der Füße, gibt Anja preis, sei Beweis für die Echtheit der Vorsprechenden. (ZDB 7061: DWDS K-Ze 1996)
'Even the position of the feet, Anja discloses, is proof for the authenticity of the participants of the audition.'

5 The structure of the database

In order to use the database, it is important to have some understanding of how it is put together. This will especially be the case if you want to really take advantage of its capabilities by formulating advanced searches and interpreting the results that they return. The full internal complexity of what is running under the hood is a bit daunting, but the fundamentals are quite straightforward and pretty easy to understand on the basis of the OWID^{plus} search interface. Fortunately, these fundamentals are all you really need to use the database like a pro.

5.1 Predicates and examples

The database is built primarily around two sets of data and the connections between them. On the one hand there is a table of clause-embedding predicates, and on the other there is a table of naturally occurring example sentences taken from corpora. Each sentence in the Example table is associated with one predicate in the Predicate table — it demonstrates one particular use of that predicate for embedding. The two tables consist of a series of entries, each storing several pieces of information relating to a single predicate or example, respectively. So an entry in the Example table contains the text of the example itself, plus for instance information about the corpus source and the argument structure of the example, and an entry in the Predicate table gives the form of the predicate as well as information about its category and morphological structure.

Under the hood there is some additional complexity in how information is organized in the database, but in the end, it all comes down to information about examples and information about predicates. The distinction between the two is all that you need to be concerned with as a user of the database, in order to formulate searches and interpret their results. Most importantly, it is what the OWID^{plus} search interface is built around: at any given time it displays either an example table view or a predicate table view, and every search that you run is thus ultimately interpreted as either a search for predicates fitting certain criteria or a search for examples fitting certain criteria. In general, this is fairly straightforward and shouldn't lead to much confusion. However, there are some cases where it can be tricky to keep example properties straight from predicate properties. Furthermore, the distinction really matters for advanced searches, since the two types of properties are treated differently in a way that is crucial for the system to work. This can be a bit tricky to grasp at times, so we will discuss it in great detail in Section 9 Here we'll cover the basics.

The complexity arises from the fact that the examples are there to demonstrate the behavior of the predicates, so every example is tied to — and ultimately tells us something about — one of the

predicates. This can lead to two types of confusion. The first is that some pieces of information can reasonably be understood as properties both of an example and of the predicate it contains. For example, you might be interested in investigating how separable prefixes might interact with embedding, as in example (9):

(9) Verbraucherschützer raten deshalb davon ab, nur nach dem Beitragssatz zu schauen. (ZDB 101: DWDS BZ 2005)

'Consumer protection groups therefore recommend against only looking at the premiums.'

Now, being a separable prefix verb is of course a property of predicates, but having such a verb as the embedding predicate is a property of examples, and so it is something you might search for, whether you're interested in finding predicates or examples. The second issue is that the most interesting properties of a predicate tend to be about the kinds of clauses that it can embed, like whether it can take an infinitival or interrogative clause as its complement. The tricky thing is that the characteristics of those embedded clauses really describe the specific example, not the predicate, and so they are treated as example properties by the database. This means that when you formulate searches for predicates, you won't just be using predicate properties. Typically, a search for predicates will involve a combination of predicate and example properties. And the same goes the other way around: searches for examples often have to make reference to predicate properties.

So let's consider in a bit more detail what predicate properties and example properties are and the respective roles they play. The predicates are at the heart of the database. In a way, they are what the whole enterprise revolves around: the attempt to document and understand how different lexical predicates and predicate classes behave with respect to clausal embedding. The database and its search interface are thus designed to make it possible to run sophisticated searches to obtain lists of predicates with complex combinations of properties. On the other hand, the examples make up most of the actual substance of the database. They constitute the bulk of the data collected, curated and coded for research use in the database, and indeed, what primarily interests us about the predicates is what kinds of clauses they can embed. The information about these embedded clauses is recorded in example properties, and thus to a large extent we search for predicates not by specifying their own properties, but those of the examples they embed.

Because of this, there are far more example properties than there are predicate properties in the database. And the disparity here is actually inflated by the fact that a number of predicate-specific properties that are recorded in the database are not part of the current public release, because they pertain specifically to the language or historical language stage that a predicate belongs to. Since the current release of the database is restricted to contemporary German, information about language and language stage is uninformative and hidden from view in the search interface. Detailed information on the predicate properties currently used in the database, including their possible values, can be found in Section 7.1 of this guide. Detailed information on the example properties that are currently in use are given in Section 7.2.

5.2 Predicate properties and example properties in complex searches

The way that all of this comes together in running a complex search query is as follows. First, consider the possibility that you are interested in finding a list of examples that meet certain conditions, some involving example properties, others involving predicate properties. You will for-

mulate these conditions as criteria, each of which places restrictions on either example properties or predicate properties. The search will then return all examples for which the example criteria hold **and** which have predicates for which the predicate criteria hold. So an example search with the example criterion '**example type** is *interr*' and the predicate criterion '**pred. morphology** is *Pt-V*' will return all interrogative examples containing a predicate which is a particle verb. This is fairly straightforward.

Now consider that you are interested in finding a list of predicates that fit certain criteria, again some involving example properties, others involving predicate properties. It is easy to see what role the predicate criteria will play. They will directly describe the predicates you wish to find, e.g. as having a certain **category** or not having a certain **morphology**. With the example criteria it gets a bit more complicated. The basic idea is that they will be describing something about the examples that are associated with the predicates you are looking for. So a predicate search with the example criterion '**example type** is *inf*' will return only predicates that have at least one example that is of that type.

Where it gets tricky is when you combine together multiple example criteria on a predicate search, or where you negate one of them. If you do a predicate search with the example criteria '**example type** is *interr*' and '**verb mood** is *KONJ I*', what will you get? Will it return a list of predicates that have at least one example that has an embedded interrogative in the subjunctive I mood? Or will it return a (probably longer) list of predicates that have at least one embedded interrogative example, and at least one subjunctive I example, which may or may not be the same example as the interrogative one? These are both reasonable things that you might be interested in searching for. Similarly, if you do a predicate search with the example criterion '**example type** is not *inf*', what will that give you? Will it give you predicates that have at least one example that is not **example type**:*inf*? Or will it give you predicates that do not have **any** examples with **example type**:*inf*? Again, either one of these would be a reasonable search that you might want to run, depending on what you're researching.

The answer to both questions is that searches of both types are possible, and what you get depends on whether you check a particular box before running the search, in particular whether you click the box **independent example criteria**, or leave it unclicked and get the default 'single example criteria'. In a search with 'single example criteria', for each predicate, it goes through the examples one by one and tries to find at least one that satisfies all of the criteria. If it finds a single such example, the predicate counts as a hit. If a criterion is negated, it's just a matter of finding a single example that satisfies the negated criterion. In the case described above, this would be one example that is not **example type**:*inf*. In a search with 'independent example criteria', for each predicate, it goes through the criteria one by one and makes sure that each criterion can be satisfied on the examples. If a criterion is positive, it just takes one fitting example to satisfy it, and this doesn't have to be the same example that satisfies other potential criteria. If a criterion is negative, then it takes just one counterexample to violate it. In the case described above, this would mean that if any example with a predicate has **example type**:*inf*, then that predicate won't count as a hit.

6 The Example type system

One of the most important example properties in the database is **example type**, as the system of types plays a central role in the organization of the database which may not be immediately

obvious. A bit of understanding of the motivation and implementation of the system can go a long way toward running complex queries and getting at some of the most interesting data that the database can offer. This section is devoted helping you get there.

6.1 The idea

Recall that the central idea of the database is to document, as exhaustively as possible, all of the kinds of embedding that can be done with specific lexical predicates. This means that the driving impetus behind our data collection has been to identify a fixed set of types of embedded clause, and then see whether each one is attested with every predicate in the database. There are several different ways to identify types of embedding, involving a number of cross-classifying dimensions. Many of these are reflected in the various example properties, e.g. how the embedded clause fits into the argument structure of the embedding predicate, what kinds of control relations obtain between matrix arguments and the embedded subject, and whether the embedded clause is inflected as an indicative or some kind of subjunctive. The central notion, however, is that we can identify a handful of coarse-grained embedded clause types, which seem to be relevant for selection, and then classify clause-embedding predicates according to which of these clause types they can actually select.

The way that the database was built up is that a series of such clause types was identified for each language, which then played a central role, both in how data was collected and in how additional information about examples was encoded. The term we use for these designated clause types is **example type** (in part because at least some of them characterize not just the embedded clause, but also certain aspects of its relationship to the matrix clause, and hence the entire example). The central dictum for collecting data is that considerable effort was made to find attestations in the corpora of every **example type** for every predicate, even in cases where the native-speaker intuitions of the annotators would have led them to expect that such embedding would be impossible. While such intuitions are reliable in most cases about the acceptability of particular, concrete structures, native speakers are not always good at imagining different variations on a structure that might be acceptable in the right context. For this, corpora are invaluable, and indeed there are several instances where a particular predicate was found to be able to embedd a particular **example type** against expectations, under the right circumstances.

The database is thus an excellent tool for investigating such cases of unexpected embedding. The offshoot of this procedure is that the database can be used as a uniquely reliable indication not only of what clause types particular predicates can embed, but also of what types they **cannot** embed. That is, if the database does not contain an example of a given predicate with a given **example type**, it can be concluded with a fair degree of confidence that such embedding is either impossible or at least restricted to the point of being vanishingly rare. As described in Section 4.3, we followed similar procedures for more specific types of embedding examples, i.e. we always attempted to find examples of embedding with each predicate with a series of distinct properties, like if a predicate is known to embed finite verb-final clauses, we attempted to find examples with all the different moods. Here as well, conclusions can often be drawn on the basis of the lack of examples, but it is with the **example types** that our efforts were most extensive and thus these evidentiary basis for such conclusions is strongest. (In the case of some properties, we have not yet been able to approximate exhaustivity, and so no conclusions can be drawn from the lack of examples. Such instances are mentioned explicitly in the relevant places in Section 7.)

6.2 The role of example type within the database

Every example in the database belongs to a single **example type**. That is, the **example types** are not descriptive, but classificatory, and thus they are defined to be strictly non-overlapping. In some cases this leads to slightly convoluted definitions, since a series of different, partly orthogonal criteria have to be used to identify the relevant clause types for a given language. In general though, this does not lead to problems, and in cases where this would lead to ambiguity about the properties of certain examples, we have introduced additional properties like **finiteness**, **word order** and **semantics** to record the relevant information (see 7.2.7 for discussion).

In addition to their evidentiary and classificatory roles, the **example types** serve an important organizing function for the encoding of additional properties in the database. Many of the other example properties only really make sense for certain types of clauses, and thus they are only defined in the database for certain **example types**. E.g. **definiteness** is only defined for *nmlz*, and **controller** is only defined for *inf*. In this way, we can actually group some of the **example types** together according to the properties that they are associated with. Thus all and only the finite **example types** have **verb mood** defined. You will thus see when you build up search queries that certain types of information will only be specified on an example if it is of the right **example type**. Here is a list of the relevant dependencies:

verb mood is only defined for *compDecl*, *zeroDecl* and *interr*

definiteness is only defined for *nmlz*

control shift, controller are only defined for *inf*

6.3 The example types

Here we give a quick rundown of the **example types** that are currently in use in the public version of the database. This list will expand in future releases for two reasons. First, we are completing our collection and coding of contemporary German data for two additional **example types** at present, *paren* for parentheticals and *dSpeech* for direct speech embeddings. Second, by their nature, **example types** have to be at least partly language specific — e.g. German clauses can be either interrogative or infinitive but not both, making them reasonable distinct **example types**, but English has interrogative infinitives, meaning that its **example types** must be set up differently. We thus have slightly different **example type** systems in the other languages and languages stages we are working on, which will be made public in the future. Here is the current list for contemporary German, along with an example for each type:

***interr*:** interrogative clauses, i.e. finite argument clauses beginning with *ob* or a *w*-word

> *Man kann an ihnen direkt ablesen, ob es irgendwo in der Welt kriselt.* (ZDB 11712: DWDS Zeit 1962)

***compDecl*:** declarative clauses with a complementizer, i.e. finite argument clauses beginning with a non-interrogative complementizer like *dass*, and typically having verb-final order

> *Vielleicht hat man sich auch nur abgefunden, daß Sie bis zu Ihrem Lebensende in Wien bleiben.* (ZDB 12: DWDS BZ 1997)

***zeroDecl*:** declarative clauses without a complementizer, i.e. finite argument clauses with no overt, initial complementizer or *w*-word, typically having verb-second word order

Aber ich ahne, es wird nicht mehr als Blech. (ZDB 256: IDS brz 2006)

***inf*:** argument clauses with an infinitive as the highest verb

Gewöhnen Sie sich ab, den Teller unbedingt leer essen zu müssen. (ZDB 12615: IDS sgt 2000)

***nmlz*:** arguments that are not clauses, but NPs built around a nominalized verb, typically where the nominalization has inherited at least one of the arguments of the underlying verb

Die knappen Kassen halten nicht von der Durchführung seines offiziellen Landesfests ab. (ZDB 28: DWDS TS 2003)

7 Properties coded in the database

In this section we list and describe all of the properties that are coded in the database, first the predicate properties and then the example properties. Where appropriate, we list the possible values for the properties and how to interpret them and provide tips for running searches making use of them. Here and throughout the manual, names of properties are formatted like **example type**, names of values are formatted like *zeroDecl*, and a property-value pair is formatted like **example type**:*zeroDecl*.

7.1 Properties of predicates

7.1.1 Predicate

This is the base form of the predicate itself, typically the infinitive for verbal predicates. In some cases, **predicate** may also contain more than one lexical unit. For example, a number of predicates occur only in negated contexts and therefore are listed along with a negative particle (e.g. *nicht umhinkönnen*, 'not having a choice but to do sth.'). Others involve a verb plus another predicative element like an adjective, as in *offen lassen* 'leave open', or a predicative element plus the copula, like *im klaren sein* 'be clear on sth.' Complex forms are discussed in detail in Sections 7.1.3 and 7.1.4.

7.1.2 (Pred.) ID

Each predicate in the database has its own unique **ID**, which is an integer of 1 to 4 digits. In order to find a specific predicate again later on, one can simply make note of the corresponding **ID** and do an advanced search, where **pred. ID** is one of the criteria used.

7.1.3 Category (pred. category)

Each predicate listed in the database has been coded according to its grammatical **category**. There are three possible values for **category**:

category	description	example
V	lexical verb	*fragen* 'ask', *befragen* 'question sb.'
copP	non-verbal predicate + copula	*stolz sein* 'be proud'
cP	complex predicate	*infrage kommen* 'be a possibility'

Predicates that are a single lexical verb — whether simplex or derived — have the value *V* for **category**. Predicates that consist of a non-verbal lexical predicate combined with the copula have the value *copP* for **category**. Finally, the value *cP* stands for other types of complex predicates that consist of two or more lexical units, typically an adjective or PP plus a non-copular verb. More detailed information about the individual units that make up a complex predicate is found under the next property, **morphology**.

7.1.4 Morphology (pred. morphology)

Morphological properties of each predicate are given under the property **predicate morphology**. The relevant information in particular pertains to whether the predicate has been constructed via derivation or compounding and whether it contains a prefix or particle. The possible values of a predicate for **morphology** are tightly connected to its value for **category**, as the following table makes clear:

category	morphology	Explanation	example
V	-	simplex verb	*hören* 'listen'
	DA	deadjectival	*legalisieren* 'legalize'
	DN	denominal	*loben* 'praise'
	Pt-V	particle verb	*dazugehören* 'belong to sth.'
	Px-V	prefixed verb	*beschließen* 'decide, terminate'
cP (separable)	*A_V*	adjective+verb	*offen lassen* 'leave sth. open'
	N_V	noun+verb	*angst haben* 'be afraid'
	PP_V	PP+verb	*zur Kenntnis nehmen* 'take notice of sth.'
	V_V	verb+verb	*einfließen lassen* 'incorporate sth.'
	Part_V	participle+verb	*bestätigt sehen* 'consider sth. proven'
	Adv_V	adverb+verb	*zustande bringen* 'achieve sth.'
	zuV_V	*zu*-infinitive+verb	*zu tun haben* 'have to do with sth.'
cP (inseparable)	*AV*	adjective+verb	*frohlocken* 'rejoice'
	NV	noun+verb	*gewährleisten* 'guarantee sth.'
	AdvV	adverb+verb	*rückfragen* 'check with so.'
copP	*A*	adjective + *sein*	*lieb sein* 'would like'
	PP	*PP* + *sein*	*im klaren sein* 'be clear on sth.'
	Part	participle + *sein*	*eingestellt sein*
	zuV	*zu*-infinitve + *sein*	*zu tun sein*

For complex predicates with **category**:*cP*, the values for **morphology** consist of two category labels, potentially separated by an underscore _. The underscore indicates that the lexical components of the predicate are syntactically separable. For example, the predicate *offen lassen* 'leave open', with the **morphology** value *A_V*, is flexible as to the position of the two word parts, as in (10)

(10) Woher das Plutonium stammte, ließ der Angeklagte offen (ZDB 11822: TIGER).
'The defendant left open, where the plutonium came from'

If there is no underscore, like in *AV*, *NV* or *AdvV* in the table above, it means that the parts of a composed predicate are not syntactically separable. Thus the predicate *frohlocken* 'rejoice', which is annotated with **morphology**:*AV*, is not seperable, as you can see in (11):

(11) Auf der ganzen Welt frohlocken Pubbesucher, wenn das irische Kultbier langsam aus dem Zapfhahn in ihr Pint hineinfliesst (ZDB 20797: IDS sgt 2011)
'All over the world, pub visitors rejoice when the Irish cult beer flows slowly from the taps into their pint.'

7.1.5 Predicate meaning

This is a special property used to distinguish multiple meanings associated with what might look like a single predicate. Homonymous predicates like *raten* have a single phonological form which is associated with two totally different meanings. Whereas *raten*^{1} means 'advise', *raten*^{2} means 'guess'. In the database, these different meanings are represented by distinct predicate entries that happen to have the same form. Each example is then assigned to only one of these two predicates, according to the meaning used. This state of affairs is indicated in a special way in the search interface, since it has to do with identifying distinct predicates. Which **predicate meaning** is involved in a given instance is indicated by a superscripted numeral at the end of a predicate's name, in both the predicate table and the example table. This means that **predicate meaning** does not appear in its own column in the predicate and example tables, nor can it be added as a separate criterion in an advanced search. Instead, if you want to search for a specific **predicate meaning**, you can indicate it in the **predicate** column or a **predicate** criterion in an advanced search, by putting the name of the predicate in double quotes and appending the number of the desired meaning preceded by #. So if you want to search for just the 'guess' meaning of *raten*, you would enter "raten#2" into the **predicate** search field. Having separate entries for homonymous predicates makes it possible to keep their distinct embedding behavior apart. For example, *raten*^{1} embeds *inf*, *nmlz*, *compDecl*, *zeroDecl* and *interr*, while *raten*^{2} only embeds *interr* and *compDecl*. If we were to collapse the two meanings in a single predicate entry, this distinction in embedding behavior would be obscured.

We have tried to make a distinction between proper homonymy and polysemy, and only recognized distinct predicates in cases of the former. Polysemous items are semantic variants of a single lexeme. That is, they share not only their phonological form but also some of their semantic properties. In practice, telling the difference between homonymy and polysemy is extremely difficult, and it is not even clear in principle that there is a sharp line dividing them. Nearly any decision one could make on whether two meanings should be split up into two predicate entries or not has the potential to be controversial and leave someone dissatisfied. We are thus taking a relatively conservative approach, only splitting up entries in the cases where it is most clear, and where it is most apparent that the two meanings are associated with distinct embedding behavior. In the current release of the database we dinstinguish 44 pairs of homophonous predicates. In future releases, we may expand this practice as seems appropriate, depending in part on how useful and usable the current distinctions turn out to be.

7.2 Properties of examples

7.2.1 (Example) ID

Each example in the database has its own unique **ID**, which is an integer of 1 to 5 digits. In order to find a specific example again later on, one can simply make note of the corresponding **ID** and do an advanced search, where **example ID** is one of the criteria used. When including a specific example from the database in a publication, you should provide its **example ID** along with other **source** information to allow for easier reviewing. See Section 10.2 for the proper format for doing this.

7.2.2 Example type

This property classifies each example into one of a series of types, which are essentially the set of clause types that a clause-embedding predicate can select for in contemporary German. This is discussed in great detail in section 6.3 in this guide. Here are the values currently in use for contemporary German:

***interr*:** interrogative clauses, i.e. finite argument clauses beginning with *ob* or a *w*-word

***compDecl*:** declarative clauses with a complementizer, i.e. finite argument clauses beginning with a non-interrogative complementizer like *dass*, and typically having verb-final order

***zeroDecl*:** declarative clauses without a complementizer, i.e. finite argument clauses with no overt, initial complementizer or *w*-word, typically having verb-second word order

***inf*:** argument clauses with an infinitive as the highest verb

***nmlz*:** arguments that are not clauses, but NPs built around a nominalized verb, typically where the nominalization has inherited at least one of the arguments of the underlying verb

7.2.3 Predicate

This gives the predicate used in the example sentence and serves as the interface to the predicate properties. This is really just a link to the entry for that predicate in the predicate table, so the forms here are the same as described in section 7.1.1 above.

7.2.4 Example text

This is the full text of the example. Usually, it consists of a single sentence containing one or more embedded clauses. In some cases, additional context was given to facilitate a more accurate interpretation of the example. In order to keep the **example texts** to a manageable size, information not deemed necessary for correctly interpreting the example was not included. Ellipsis at the beginning or the end of a sentence is indicated by three dots, ellipsis in the middle of the sentence by three dots in parentheses.

7.2.5 Source

This is an abbreviated record of where the respective example was collected from. It consists of the name of a text corpus (digital collection of written language data), followed by the name of the original source of the example (e.g. a newspaper, web publication or book title) and the year it was published. A full list of the abbreviations employed can be found in section 4.2. When including a specific example in a publication, you should provide its **source** information along with its **example ID** to allow for easier reviewing. See Section 10.2 for the proper format for doing this.

7.2.6 Complementizer

This records the lexical identity of the **complementizer** that introduces the embedded clause if there is one. Note that, for the purposes of this property, we treat *wh*-words like *wer*, *was* and *wann* as **complementizers**. This is of course an oversimplification from an analytical perspective, but it allows us to record important information in a reasonable place, and it should not lead to any confusion, since these words are in complementary distribution with overt complementizers in contemporary German.

7.2.7 Finiteness

The property of **finiteness**, along with the next two to be discussed, **word order** and **semantics**, currently have a somewhat odd status in the database. They were recently added in order to record information that is not always unambiguously indicated by **example type**. However, this information is most useful for examples of **example type** *dSpeech* and *paren* and for languages other than contemporary German, all of which are not included in the current release of the database. In the current version, the information in these properties is largely predictable on the basis of an example's **example type** (e.g. *compDecl* clauses will always be **finiteness**:*Finite* and **semantics**:*Assert*), and so they are of somewhat limited use. Still, there are a few instances where they do supply extra information (while the vast majority of *interr* examples have **word order**:*VLast*, a few dozen of them have *V2*), and one can also use these properties to search for groupings of examples in different ways (e.g. **word order**:*VLast* brings together all of the *inf* and *compDecl* examples, plus most of the *interr* ones). Of course, these properties will become especially useful in future releases of the database.

Finiteness classifies embedded clauses into broad finiteness categories according to the morphosyntactic form of the highest verb. For contemporary German this amounts to a distinction between the possible values *Finite*, *Infinitive* and *Nominalization*. The currently restricted utility of this property is partly due to the fairly limited inventory of non-finite clause types in contemporary German.

7.2.8 Word order

This property classifies examples according to the position of the verb in the embedded clause. For contemporary German, e.g., it is important to distinguish between finite embedded clauses with verb-second and verb-final order, indicated by the values *V2* and *VLast*, respectively. For certain **example types**, it does not make sense to distinguish different **word orders**, and there the value *Unm* (for 'unmarked') is used instead.

7.2.9 Semantics

This property gives a very rough indication of the semantics of the embedding, corresponding approximately to the kind of force involved. We currently distinguish the following values:

Assert indicates embedded declaratives.

Quest indicates embedded interrogatives.

Unm is used for types of clauses where no distinction between declaratives and interrogatives is possible, in contemporary German *inf* and *nmlz*.

The determination of whether a clause should be assigned to *Assert* or *Quest* is done systematically according to the complementizer: *ob* and *w*-forms with the exception of *wenn* lead a clause to be classified as *Quest*, otherwise it will be *Assert*.

7.2.10 Verb mood

This property indicates the grammatical mood of the finite verb form in the embedded clause, and thus is only found with the finite **example types**, i.e. *compDecl*, *zeroDecl* and *interr*. The three possible values for modern German are *INDC* (indicative), *KONJ I* (subjunctive I) and *KONJ II* (subjunctive II). In cases where the form of the subjunctive is indistinguishable from the indicative, the **verb mood** is coded as *INDC*.

7.2.11 Definiteness

This property is used only for examples of **example type**:*nmlz* and specifies the **definiteness** of the nominalization itself. (12-a) shows an example with an indefinite nominalization, while (12-b) shows a definite one.

(12) a. Sein Arzt riet ihm dringend von *einer Teilnahme am Super Bowl* ab. (ZDB 17688: DWDS TS 2005)
'His doctor advised him urgently against participation in the Super Bowl.'

b. Trotzdem hielten sich die deutschen Teilnehmer mit *dem Vergleichen der beiden Ideologien* sehr zurück. (ZDB 17532: DWDS PNN 2005)
'Nonetheless, the German participants held back considerably in the comparison of the two ideologies.'

The following specifications values are in use:

def nominalizations with a definite determiner

indef nominalizations with an indefinite determiner

null bare nominalizations, i.e. ones without a determiner

7.2.12 Controller

This property is used only with **example type**:*inf* and encodes the control properties of the respective predicate in the given example. Control is understood as the identification of the covert subject of the infinitival clause with an argument of the matrix predicate, for instance in the case of *gestehen* 'confess' with its subject, and in the case of *auffordern* 'prompt' with its object.

(13) a. Peter$_x$ gestand seiner Schwester$_y$, _ $_x$ ihr Auto benutzt zu haben.
'Peter confessed to his sister that he used her car'.
b. Peter$_x$ fordert seine Schwester$_y$ auf, _ $_y$ sein Auto zu reinigen.
Peter prompted his sister to clean his car.

Controller is specified in terms of the argument variable (taken from the property **arg. structure** described below) that corresponds to the controller:

(14) a. *gestehen*: **controller**:*x*; **arg. structure**:*P-(y)-x*
b. *auffordern*: **controller**:*y*; **arg. structure**:*P-(y)-x*

There are two special cases: if the **controller** does not correspond to an (explicit or implicit) argument of the matrix predicate, it is specified as *v*, as in (15-a). A very specific subcase is found with the verb *anordnen* 'order', which shows control with an obligatorily implicit argument (coded as *i*), as in (15-b).

(15) a. Und Ehebruch basiert ja darauf, _ $_v$ etwas zu verstecken. (ZDB 1359: DWDS TS 2004)
And indeed adultery is based on hiding something.
b. Die Staatsanwaltschaft$_x$ ordnete an, _ $_i$ den Toten zu obduzieren. (ZBD 630: DWDS BZ 2005)
The D.A.'s office ordered that the victim be autopsied.

Apart from the simple cases shown in (14), there are more complex cases in which either two arguments of the matrix predicate jointly function as **controller** (= "split control") or in which the **controller** is understood as a set of referents including the referent of one of the matrix predicate's arguments and further referents (= "partial control"). The former may be illustrated with *vereinbaren* 'agree' as in (16-a), the latter with *zustimmen* 'consent' as in (16-c).

(16) a. Maria$_x$ vereinbart mit Peter$_y$, _ $_{x+y}$ sich vor dem Kino zu treffen.
'Maria agrees with Peter to meet in front of the cinema'
b. **controller**:*x+y*
c. Maria$_x$ stimmt zu, _ $_{x+v}$ sich vor dem Kino zu treffen.
'Maria consents to meet in front of the cinema'
d. **controller**:*x+v*

Sometimes, the control reading cannot be fully resolved from the context. In this case, the coding *x/v* is chosen, as in (17)

(17) _ $_{x/v}$ Kleine Truppen zu finanzieren, finde ich$_x$ okay. (ZDB 4917: DWDS TS 2005)
'I think it's okay to finance small squads.'

The database also encodes whether the control reading is "inherent" to the predicate or one of its readings. The notion of inherent control goes back to Stiebels (2010). Usually, control is only discussed with respect to infinitival complements. However, if the full array of admissible clausal complements of a predicate is taken into account, it becomes obvious that some predicates require the identification of an argument of the embedded predicate (mostly the subject) with an argument of the matrix predicate in all types of clausal complementation, whereas others show control only with infinitival complements. This contrast can be nicely demonstrated with the two factive verbs *bereuen* 'regret, repent' and *bedauern* 'regret'. Inherent control is coded by adding *, inh* to the value for **controller**.

(18) a. Maria$_x$ bereut es, _ $_x$ einen SUV gekauft zu haben.
'Maria regrets to have bought an SUV.
b. Maria$_x$ bedauert es, _ $_x$ einen SUV gekauft zu haben.
'Maria regrets to have bought an SUV.
c. *Maria bereut es, dass Peter einen SUV gekauft hat.
Maria regrets that Peter bought an SUV.
d. Maria$_x$ bereut es, dass sie$_x$ einen SUV gekauft hat.
Maria$_x$ regrets that she$_x$ bought an SUV.
e. Maria bedauert es, dass Peter einen SUV gekauft hat.
Maria regrets that Peter bought an SUV.
f. *bereuen*: **controller**:*x, inh*
g. *bedauern*: **controller**:*x*

The following table summarizes the possible values for **controller**:

x or *y*	exhaustive control by the matrix predicate's argument x or y, respectively
i	control by an obligatorily implicit argument (argument is also specified as such in the argument structure of the predicate)
v	control by a referent which is not an argument of the matrix predicate (= non-local control)
x+y	split control by x and y
x+v	partial control by the matrix predicate's argument x and a further referent v
x/v	control by x or v or x and v
inh	inherent control
a	not really control, but raising: the null subject of the infinitive has raised into the matrix clause, also indicated by an *a* in the **arg. structure**

7.2.13 Control shift

This field is also restricted to infinitival complements. It encodes whether the control relation is shifted from the canonical control reading (coded as +). **Control shift** can be observed with certain clause-embedding predicates, but not with others. For instance, the verb *versprechen* 'promise' allows a shift from subject to object control, whereas the verb *bitten* 'ask' allows a shift from object to subject control. This shift is triggered in these verbs if the embedded clause is passivized or modified with the deontic modal *dürfen* 'be allowed'; also recipient-oriented verbs such as *empfan-*

gen 'receive' in the embedded clause may trigger such a shift. (19) illustrates the **control shift** in *versprechen* and its annotation.

(19) a. Peter$_x$ versprach Maria$_y$, _ $_x$ den Wagen zu waschen.
'Peter promised Maria to wash the car'
b. **controller**:*x*; **control shift**:-
c. Peter$_x$ versprach Maria$_y$, _ $_x$ den Wagen waschen zu dürfen.
'Peter promised Maria to be allowed to wash the car'
d. **controller**:*y*; **control shift**:+

Note that **control shift** is only assumed for those verbs that have an underlying preference for a specific control relation. Verbs such as *vorschlagen* 'propose' are not marked for **control shift** because there is no default tendency for subject, object or split control. Finally, note that we have not attempted to exhaustively document which predicates allow control shift. That is, we have not systematically searched for examples showing control shift with all relevant predicates in the database, but rather simply attempted to code for it in the examples we have found. In other words, while one may draw conclusions based on examples with **control shift**:+ in the database, one may not draw conclusions based on the lack of such examples with particular predicates.

7.3 Arg. structure and arg. realization

7.3.1 Description of arg. structure and arg. realization

The two properties **arg. structure** and **arg. realization** both pertain, as their names suggest, to the argument-taking properties of the clause-embedding predicate in a particular example sentence. Because they are closely tied to each other in the database, it makes sense to discuss them in tandem here.

The **arg. structure** lists the (potentially abstract) arguments that the clause-embedding predicate takes in a particular example, indicates the semantic types of these arguments and shows their relative positions in the argument hierarchy, adopting ideas from Lexical Decomposition Grammar (see e.g. Wunderlich 1997, ultimately going back to Bierwisch 1983 on the idea of the depth of embedding of arguments in SF). For instance, the verb *versprechen* 'promise' has places for three arguments. They can be identified by asking *Wer hat wem was versprochen?* 'Who has promised what to whom?'. Two of the argument places are for expressions denoting individuals, and one is for a propositional expression. Argument places for individual arguments are symbolized by variables like *x*, *y* and *z*, while capital letter variables like *P* and *Q* are used to symbolize propositional argument places. Thus, we get **arg. structure**:*P-y-x* for *versprechen*. The order of the variables in the **arg. structure** mirrors the order of the arguments in the syntactic base structure (again, following the argument hierarchy). *P*, the leftmost argument variable, is replaced by a complement that is closest to the verb, *y* is replaced by the next higher one and *x* is specified by the highest one.

How the abstract argument places of the **arg. structure** are actually realized morphosyntactically in a given example is indicated by the property **arg. realization**. Individual argument places are typically realized by noun phrases, which in German are case marked. For *versprechen*, *y* and *x* will typically be realized by constituents with dative and nominative case, respectively as, for instance in (20)

(20) Der Präsident verspricht seinem Sohn, dass er den Aufsichtsratsvorsitz bekommt.

'The president promises his son that he will get the chairmanship of the Board'.

Propositional arguments can be realized as clauses, as in the previous example, or by DPs, like the Accusative-marked *den Aufsichtsratsvorsitz* 'the chairmanship of the Board' as in (21):

(21) Der Präsident verspricht seinem Sohn den Aufsichtsratsvorsitz.

They can also be expressed by a nominal sentential correlate like *es* 'it', as in (22):

(22) Der Präsident hat es seinem Sohn versprochen, dass er den Aufsichtsratsvorsitz bekommt.

Therefore, the realization of propositional arguments is also specified in terms of case in the database. The **arg. realization** of *versprechen* in the examples discussed here then reads as follows: *ACC-DAT-NOM*, with ACC being the realization of *P*, DAT being the realization of *y*, and NOM being the realization of *x*. With other predicates, arguments of various types may be realized instead as PPs, as expletives, as APs and various other grammatical elements. All of these possibilities are distinguished by the possible values for **arg. realization** and will be explained here.

7.3.2 Possible values

The following two tables give an overview of the elements used to construct the **arg. structures** and **arg. realizations** used in the database. They are ordered alphabetically. Encodings that are require some comment are marked by * and will be discussed in more detail in section 7.3.3. First, we list the types of variables used in **arg. structure**.

var.type	explanation	arg.struc	example
a	argument place for a raised subject	*P-a*	Es dauerte eine Weile, bis *sie* anfing, sich wieder mit mir zu unterhalten.
e	expletive*	*P-e*	Somit bleibt *es* dabei, dass die Länder und die Kommunen darüber entscheiden.
i	variable for an obligatorily implicit argument	*P-i-x*	Die Staatsanwaltschaft ordnete an, den Toten zu obduzieren.
P, Q, R	Propositional variables. They are realized by finite clauses, infinitives, nominalizations and quantified propositional expressions.*	*P-x*	Natürlich wisse sie von *der Schließung des Hauses*, aber von *Untergangsstimmung* spüre sie nichts.
		Q-P	*Eine Reduzierung auf zwei Präsidien* würde *eine Aufgabenverlagerung von Präsidien hin zu Schutzbereichen* bedingen.
		R-Q-P	*Die Übernahme von "know how"*, ... unterscheidet, *Wirtschaftsentwicklung** von *bloßem Anwachsen des Umfanges wirtschaftlicher Tätigkeit* ...
Pr	variable for a non-verbal predicate	*Pr-r-P*	Es hört sich recht *gut* an, kleinere Klassen, ... zu verlangen.
		Pr-y-x	Ein ägyptisches Gericht befand ihn für *schuldig, vom Glauben abgefallen zu sein*
		Pr-P-x	Ich finde *gut*, daß wir den Tieren helfen.
		Pr-r-x	Er äußert sich *glücklich darüber, dass sie kommt*
r	variable for an inherently reflexive pronoun	*P-r-x*	Frank hat es *sich* überlegt zu kommen.
x, y, z	variables for individuals	*P-y-x*	*Indurain* hatte *seinem Herzen* angewöhnt, nur 35mal pro Minute zu schlagen. *Er* richtet zudem *von der Geschäftsleitung* aus, dass Foodwatch gerne per Mail Kontakt aufnehmen könne.
(...)	Brackets indicate optional arguments*	*P-(y)-x*	Er riet *von der Schaffung* eines Einparteiensystems ab.

Now consider the various **arg. realization** types:

realization	explanation	arg.struc.	arg.real.	example
ACC	Accusative	*P-x*	*ACC-NOM*	Ich hoffe, dass uns solche Probleme im Fall des Kosovo erspart bleiben,
ACC[prof]	P is realized by the sentential pro-form *es*.	*P-x*	*ACC[prof]-NOM*	Riedels Resümee: "Ich habe *es* gehofft, dass das konzentrierte Krafttraining anschlagen wird, aber" ...
		Pr-P-x	*PP[als]-ACC[prof]-NOM*	Ich empfinde *es* als Vorteil, wenn man sich in mehreren Ländern heimisch fühlt,
AP	realization of Pr as AP	*Pr-P-x*	*AP-ACC[prof]-NOM*	Die Kassen fänden es *gut*, wenn eine Begutachtung schneller ginge, sagt er.
DAT	Dative	*y-(x)-P*	*ACC-DAT-NOM[prof]*	Es würde *uns* das Leben erleichtern, wenn wir unsere Vorurteile gegenüber dem Islam abbauen.
DAT[prof]	P is realized by the sentential pro-form *dem*.	*P-x*	*DAT[prof]-NOM*	... und Miller hatte *dem* zustimmen müssen, dass Polen nicht mit einer Minderheitsregierung in die letzte Etappe vor dem EU-Beitritt gehen wird.
EXPL	realization of the expletive argument place *e*	*P-e-x*	*OBL[mit]-EXPL-NOM*	Erneut haben wir *es* hier mit dem Erlangen eines Monopols zu tun,
GEN	Genitive	*x-P*	*GEN-NOM*	Wahrlich, dies zu ergründen, wäre *eines Psychologen* würdig!
		Q-P	*GEN-NOM*	Allerdings sei "nicht jede Kritzelei würdig, festgehalten zu werden".
GEN[prof]	sentential pro-form *dessen.*	*P-x*	*GEN[prof]-NOM*	... "wir werden *dessen* inne, wer wir sind." ...
NOM	Nominative	*P*	*NOM*	Zu Beginn der Woche sickerte durch, dass die Telekom womöglich die Online-Firma Club Internet kauft, ...
		*(x)-P**	*ACC-NOM*	Die Menschen störe, wie Rot-Grün mit Institutionen und Regeln umgehe.

NOM[prof]	sentential pro-form *es*	*x-P*	*ACC-NOM[prof]*	Stattdessen macht er den Eindruck, als würde *es* ihn heute beschäftigen, ob er nicht doch ausgewichen sei.
NULL	optional argument that is not realized	*Q-(x)-P*	*OBL[von]-NULL-NOM*	Diese erfreulichen Tatsachen sollten aber nicht davon abhalten, dass in den einzelnen Kreisen weiterhin Werbung … gemacht wird.
		P-(y)-x	*ACC-NULL-NOM*	Firmen hätten bereits für den Fall von Behinderungen angedroht, dass sie Ausfallgelder verlangen.
OBL[a…z]	realization of P, Q, and R as oblique, that is, as P- object*	*P-r-x*	*OBL[über]-REFL.ACC-NOM*	Ich freue mich vor allem *darüber*, jetzt Alkohol in allen Variationen zu genießen.
OBL[0]	non-overt prepositional adverb*	*P-r-x*	*OBL[0]-REFL.ACC-NOM*	Wir gingen also … und freuten uns, welche Ordnung hier herrschte.
PP[als]	realization of a predicative as PP with *als*	*(Pr)-P-x*	*PP[als]-ACC[prof]-NOM*	Er entlarvt es *als Illusion*, von der linearen Weltzeit zu erwarten, daß sie auf Vergangenem aufbauen kann, …
			als + DP	
			als + AP	Er habe es schon *als verrückt* abgetan, daß sie das Grab für Bello besorgt hat.
PP[a … z]	realization of an individual argument as P-object	*(x)-r-P*	*PP[für]-REFL-NOM[prof]*	*Für die Guerilla* könnte es sich auszahlen, den Konflikt in die Nachbarländer hineinzutragen.
REFL	realization of *r* if *r* relates to *P*	*r-P*	*REFL-NOM[prof]*	Nach einigen aufgeklärten Straftatenserien hat es *sich* auf Seiten der Täter herumgesprochen, den Ballungsraum Berlin besser zu meiden.

REFL.ACC	reflexive with Accusative case	*P-r-x*	*OBL[mit]-REFL.ACC-NOM*	Ich hatte *mich* doch eigentlich längst damit abgefunden, … nie ganz oben auf dem Siegerpodest stehen zu dürfen.
REFL.DAT	reflexive with Dative case	*P-r-x*	*ACC-REFL.DAT-NOM*	Ich maße *mir* keinen Augenblick an, daß wir nach Moskau gehen können, …
REFL.PP[a…z]	reflexive with P-case	*P-r-x*	*OBL[0]-REFL.PP[mit]-NOM*	Ich kämpfe noch eine Weile *mit mir*, doch einen Ansatzpunkt zu finden, …
ZERO	realization of a propositional argument place without case*	*P-x*	*ZERO-NOM*	Demonstranten brüllen, er solle die Christen schützen. Er schwankt ständig, ob er als Chronist die Geschichte der Grünen nachzeichnen oder seine ganz subjektive Sicht liefern soll.

7.3.3 Problematic and controversial cases

Here we discuss a series of encodings for **arg. structure** and **arg. realization** that present special difficulties. Some involve controversial decisions as a matter of principle, others involve cases where it is difficult in practice to decide between possible encodings for specific examples. In situations like this, there are no perfect solutions that will satisfy all concerns and couldn't be questioned for one reason or another. Thus our goal for this section is not to try to defend in detail every individual decision we have made, but to make clear what the issues are and indicate the considerations that have gone into those decisions. In some cases, it is quite possible that specific encodings or even general coding practices will be changed in future releases of the database, though in general we will try to keep codings stable, all other things being equal.

Expletive and proform *es*: The database contains constructions with two types of *es*: expletive *es* and sentential argument *es*. Sentential or proform *es* represents an extraposed complement clause which, depending on the matrix predicate, is either the subject or direct object. Standing in for a complement clause, the proform *es* is thus related to a meaningful constituent. Expletive *es*, on the other hand, is not connected with a meaningful constituent, but exists purely for syntactic reasons. There are 18 expletive-taking predicates in the contemporary German part of the database. Many of these involve clear-cut expletives — cf. (23-a) and (23-b).

(23) a. Beide hätten es darauf abgesehen, die Stabilität … in Nordirland zu gefährden. (ZDB 23945: DWDS BZ 2000)

'Both had the goal of threatening stability in Northern Ireland'.

b. Es geht um Macht, ... (ZDB 10644: DWDS BZ 2004)
'The focus is on power'.

But it is not always certain whether the *es* should really be seen as an expletive. Thus, the quantifier *alles* in (24-b) makes the expletive status of *es* in (24-a) questionable.

(24) a. Vor allem aber kommt es auf eine Regionalisierung der Wirtschaftszonen an. (ZDB 17859: DWDS Zeit 1974)
'It depends above all on the regionalization of economic zones'

b. Heute ist das alles Stimmungssache, alles kommt darauf an, dass der Laden läuft .... (ZDB 540: DWDS TS 2001)
'Today, everything is a matter of morale, everything depends on whether things are running smoothly.'

As for the *es* occurring in the context of a raising verb like *drohen* in (25-a), it disappears in a verb-first sentence — cf. (25-b). Therefore, it should rather be regarded as a positional *es* (i.e. only there to satisfy part of the V2 constraint), and hence not included in the **arg. structure**.

(25) a. Es droht, dass viele kein Geld kriegen. (ZDB 22406: DWDS PNN 2004)
'The threat is looming that many people won't get any money'

b. Drohte etwa, dass viele kein Geld kriegen?
'Was the threat looming that many people wouldn't get any money?'

Propositional DPs: Argument places for constituents that refer to clear-cut individuals are encoded as *x*, *y*, or *z*. Argument places for clauses are encoded as P, Q, or R as in (26-a). Nominalized propositions like *der Entscheid* 'the decision' in (26-b) are also encoded like clauses.

(26) a. Ob beide genehmigt werden, hänge noch davon ab, ob in Hordorf möglicherweise eine integrative Gruppe eingerichtet werde. (ZDB 12046: IDS brz 2006)
'Whether both will be allowed depends on whether an inclusive group will be established in Hordorf'.

b. Der Entscheid wird davon abhängen, ob die Atommächte die Zeit benützt haben, um selber ihre nukleare Bewaffnung zu vermindern. (ZDB 12045: DWDS KK-Ze 1965)
'The decision will depend on whether the nuclear powers have used the time to reduce their nuclear stockpiles themselves'.

However, it is not always easy to decide whether the argument is an individual or a nominalized propositional one. The DP *EU* in (27) does not refer to the EU-community but to an underspecified proposition, something like 'that the EU has a future'.

(27) Bisher hänge die EU davon ab, russische und amerikanische Satelliten mitnutzen zu können. (ZDB 12050: DWDS BZ 1999)
'Up to now, the EU has depended on the ability to use Russian and American satellites'

Additionally, both propositions, 'the EU has a future' and 'the EU is able to use Russian and American satellites' are actually polarity questions when co-occurring with the verb *abhängen* 'depend'.

Non-overt prepositions OBL[0] and ACC: There is a minor class of matrix predicates (about 100), the **arg. realization** of which can vary while the **arg. structure** remains unchanged — e.g. *(es/darüber) abstimmen* 'decide', (*es/damit*) *abwarten* 'await', *(es/darüber) diskutieren* 'discuss', *(es/darauf) hoffen* 'hope', and *(es/davon) hören* 'hear'. These predicates either exhibit an *es*-correlate as in (28-a), or a prepositional adverb as in (28-b), or they occur without any correlate as in (28-c).

(28) a. "Wir haben es abgestimmt, dass sie in diesem Jahr ins Rennen gehen", sagte Becker. (ZDB 11116: IDS rhz 2006)
'We've voted that they will take part in the race next year, Becker said.'
b. Die Mieter haben darüber abgestimmt, dass ihr Haus hundefrei bleibt. (ZDB 11112: DWDS TS 2000)
'The renters have voted on whether their house will remain dog free.'
c. Die Menschen im Südsudan haben abgestimmt, dass sie einen eigenen Staat wollen. (ZDB 11115: IDS nun 2011)
'The people of South Sudan have voted that they want their own state.'
d. Am heutigen Mittwoch muß das Parlament beschließen, ob die Arbeit der Richter in der Korruptionsaffäre untersucht werden soll. (ZDB 2339: DWDS BZ 1998)
'Today, Wednesday, the parliament must decide whether the the judges' work in the corruption affair should be investigated.'

If there isn't any correlate as in (28-c) and (28-d), we have encoded the **arg. realization** of P as *ACC* (where we would have expected an *es*-correlate) or as *OBL[0]* (where we would have expected a prepositional adverb correlate). The latter is quite rare.

OBL and PP: A propositional argument that is linked to the predicate by a prepositional adverbial like *daraus* in (29-a) is encoded as OBL[...] if its relating clause is extraposed. If the propositional adverb is anaphoric as in (29-b), it is also encoded as OBL[...], but the relating clause is not considered in the annotations. If a phrase like *Mietvertrag* as in (29-c) represents a propositional argument and is the complement of a preposition, it is encoded as PP[...].

(29) a. Deutschland und Japan leiten ihre Forderung unter anderem daraus ab, dass sie nach den USA die wichtigsten UN-Geldgeber sind. (ZDB 53: DWDS TS 2004)
'Germany and Japan motivate their demand in part based on the fact that they are the most important principal donors to the UN.'
b. Daraus leite ich ab, sie können auch in Landtagen nicht mit fiktiven Mehrheiten operieren, ... (ZDB 55: DWDS Zeit 1982)
'From this I conclude that they cannot operate with fictitious majorities even in state parliaments, ...'
c. Anders verhält es sich, wenn sich aus dem Mietvertrag ergibt, dass die angegebene Quadratmeterzahl nicht bloße Beschreibung ist, ... (ZDB 4179: DWDS BZ 2005)
'The situation is different if it becomes clear from rental agreement that the specified square footage is not a simple description, ...'

Quantified propositional expressions: Propositional variables can be realized by quantified propositional expression as shown in (30-a) and (30-b). As for (30-a) with the predicate *wissen* 'know'

and its **arg. structure**:*Q-P-x*, P is realized by *mehr* 'more'. The latter quantifies over propositions and could be replaced by a proposition. In (30-b), *wenig* 'little' and *nichts* 'nothing' are also quantifying over propositions, but additionally encode negation.

(30) a. Nach diesem Sommer der Biennale und der documenta werden wir mehr darüber wissen, ob der Katalog oder die Ausstellung recht hat. (ZDB 24704: DWDS Zeit 1982)
'After this summer of the Biennala and the Documenta, we will know more about whether the catalogue or the exhibition is correct.'

b. Denn ihnen bringt die Verkürzung der Arbeitszeit wenig oder gar nichts. (ZDB 2896: DWDS Zeit 1988)
'Because the reduction of working hours doesn't do them any good.'

ZERO: Predicates like *ächzen* 'groan' exhibit *ZERO* as **arg. realization** — cf. (31-a). A *ZERO* argument is a clause that cannot be replaced by a sentential correlate (31-b) (hence we cannot associate it with a particular case), nor can it be in the middle-field (31-c) or undergo CP-movement (31-d).

(31) a. So geärgert muss ihn dieser Name haben, dass er ... als Erstes ins Mikrofon ächzte, wie gut er sich beim Marathon "amüsiert" habe. (ZDB 13248: DWDS BZ 2000)
'This name must have made him so angry that he first groaned into the microphone what a good time he had had at the marathon.'

b. * ..., dass er es ... als Erstes ins Mikrofon ächzte, wie gut er sich beim Marathon "amüsiert" habe.

c. *Als Erstes hat er dass er sich beim Marathon "amüsiert" habe, ins Mikrofon geächzt.

d. *Dass er sich beim Marathon "amüsiert" habe, hat er als Erstes ins Mikrofon geächzt.

As for (32), one could suggest that the **arg. realization** of P be *OBL[0]*. But since *damit* doesn't really exist as an overt prepositional adverb, *OBL[0]* is perhaps inadequate.

(32) Jeden von uns, der ihn irgendwie ansprach, maulte er an, auf den Ton zu achten, so nicht mit ihm zu reden und so weiter. (ZDB 13296: IDS rhz 2007)
'He growled at every one of us who tried to talk to him in any way that they should watch their tone, couldn't speak to him that way and so forth.'

Predicates with a multiple-place argument structure and a propositional subject: Predicates like *sich auszahlen* 'pay' and *frustrieren* 'frustrate' in (33-a) and (33-b) are multiple-place predicates and have their propositional argument place in the highest position. Specifically, *sich auszahlen* has *x-r-P* and *frustrieren* has *x-P* as **arg. structure**. As for object experiencer predicates like *frustrieren* 'frustrate', it is often assumed that the experiencer argument is higher than the propositional argument, thus retaining *P-x* as **arg. structure**. We decided on the encoding *x-P* in order to make clear their experiencer verb status as in (33-a) or to indicate that the embedded clause is in the subject position (33-b).

(33) a. Es frustriert mich, mit KollegInnen zu sprechen, deren Sprachkenntnisse limitiert sind. (ZDB 5050: DNB 2007 SA4 990100162)
'It frustrates me to talk to colleagues with limited language ability.'

b. Die Förderung von Transfereffekten … zahlt sich bereits heute für die Stadt aus. (ZDB 12677: DWDS TS 2003)

'The faciliation of transfer effects pays for the town already today.'

Optional arguments: Brackets indicate optional arguments. This means that there generally will be at least two examples in the database to demonstrate the optionality, one where the optional argument is realized and one where it is not — cf. (34-a) and (34-b) (aside from when the optional argument is the only propositional one, and leaving it off would amount to an example without embedding, which thus wouldn't belong in the database).

(34) a. Eine Frau verzeiht alles, aber sie erinnert uns oft daran, dass sie uns verziehen hat. (ZDB 4249: IDS brz 2007)

'A women forgives everything, but she often reminds us that she has forgiven us.'

b. Hans-Olaf Henkel, hat bei seinem Besuch in Havanna daran erinnert, dass es Deutsche waren, die den Sozialismus und den Kommunismus erfunden haben. (ZDB 4251: DWDS Zeit 1999)

'During his visit in Havana, Hans-Olaf Henkel called to mind that it was Germans who invented socialism and communism.'

One could also have annotated such predicates with two **arg. structures**, with *P-y-x* for (34-a) and *P-x* for (34-b). But this does not indicate the existence of the other argument structure the way that the bracketing notation — in this case *P-(y)-x* — does.

8 Using the search interface

8.1 Exploring the two database tables

8.1.1 Examples and predicates

The database consists of two main tables:

- A table of corpus examples and their annotated properties, henceforth example table, characterized by an orange color scheme;
- and a table of clause-embedding predicates and their annotated properties, henceforth predicate table, characterized by a blue color scheme.

The two tables are tightly linked to each other in a "1:n"-relationship. This means that

- each example in the example table contains exactly one predicate from the predicate table;
- for each predicate in the predicate table, there is at least one, but usually several, examples containing this predicate in the example table.

The user may freely switch between the two tables at any time using the radio buttons at the top of the page. The tables, which adapt to the browser viewport size, support virtual scrolling. This means that the user can scroll through tens of thousands of examples as if the complete dataset were present in the browser. Behind the scenes, however, the browser successively loads chunks of data from a server during scrolling.

8.1.2 (In)visible columns

Both tables have a large number of columns, each of which represents a certain property of an example or predicate. By default, only a small number of these columns is actually visible. By clicking on the **Column visibility** button, users may freely choose to include or exclude columns. Each column heading displays an **(i)** symbol (as in 'info') linked to the appropriate section in this documentation. Note that the color schemes are continued here: example properties are printed at the tops of their columns in orange, predicate properties in blue.

8.1.3 Inherited properties

Examples and predicates are said to 'inherit' each other's properties. This is obvious for examples: each example contains a clause-embedding predicate whose properties can be assigned, albeit indirectly, to the example. The case of predicates is more involved: Given a certain example property **P**, we will say that a predicate inherits value x of **P** if and only if at least one example with this predicate has value x. If a predicate has examples with **P**:x, other examples with **P**:y and yet other examples with **P**:z, then the predicate inherits all three values x, y, z for property **P**; all three values appear in the predicate table cell. Please note that the order of values in the predicate table is simply alphabetical.

As a consequence of inheritance, the two tables feature, in a certain sense, the same set of properties. In a strict sense, this creates redundancy in the data presentation; but, as will become apparent shortly, it makes advanced searches much easier. Another consequence of inheritance is that, normally, the predicate table is simply a "collapsed" version of the example table, with all examples sharing a certain predicate being represented in one single row. We will refer to this fact by saying that the tables are **in sync**. Certain advanced options cause the tables to get out of sync, however; see below.

8.1.4 Detail view for table rows

By double-clicking on a table row, the user may open a detail view of a table record that essentially contains the data for all columns, visible or invisible, belonging to the present row.

8.1.5 Sorting

Each table can be sorted according to any of its columns by clicking on the column heading. Repeatedly clicking on a heading toggles between ascending and descending sort order. The tables support sorting by **multiple columns**: After clicking on the 'primary' column heading for sorting, the user may SHIFT-click (= click while keeping the SHIFT key pressed) on other column headings to define them, in the sequence of mouse clicks, as secondary, tertiary, ... sorting criteria. This means that if two rows show the same value in the primary column, their order is determined by the values in the secondary column; if these values are also equal, the tertiary column is considered for ordering, and so on.

8.1.6 Filters

Each column footer contains a dropdown menu or a text input field that allows the user to search for certain data configurations in real time. It is very important to understand that **filters operate**

only on the current table. (Filtering both tables simultaneously is possible with the advanced search options discussed below.) Note that filters on columns subsequently set invisible are still operational — you'll get a warning if you make a column invisible that has an active filter on it. To remove all existing filters set on the currently displayed table, click on the button **remove all filters from this table**. If you're confused about the results you're getting from a search, it may be because you forgot about something you entered in previously, so it can be a good idea to click this button and re-enter your query to see if the problem resolves itself. Some text input fields have autosuggest functionality, offering a few basic choices, followed by a list of all relevant options, as soon as the user clicks on the input field, and updating as text is entered to only show options that match with what has been entered. See Section 8.1.8 for advanced techniques you can use with filters on text fields.

8.1.7 Search semantics

The search semantics of the predicate table is a little bit complicated:

- Filters on inherited example properties search for predicates for which there is **at least one example** that obeys the filter constraint.
- If there are filters on more than one inherited example property, all filter constraints are, by default, required to hold for **one and the same example**. In other words, the predicates to appear in the table must have at least one example for which all given example filters hold simultaneously. The reason for this default behavior, which we will dub **single example semantics** here, is explained below in Section 8.2.2. You can change this behavior by checking the checkbox **independent example criteria (table filters)** below the predicate table. If this option is selected, the individual example property constraints may hold for different examples belonging to the predicate: there must be at least one example conforming to the first example property filter; at least one example — possibly, but not necessarily identical to the first one — conforming to the second one; and so on. In what follows, we refer to this special behavior as **independent example semantics**.

Search semantics is explained more fully in Section 8.2.2 below, and there is an extended tutorial that walks through concrete examples demonstrating how the semantics work in Section 9.

8.1.8 Advanced techniques for filtering strings

In the case of a property **P** with a text input filter, the default behavior of the filter is to only display those rows whose P value contains the string of the text input field. The filters are not case-sensitive. Two wildcards may be used: ? denotes an arbitrary character within a word; * denotes an arbitrary sequence (possibly empty) of characters within a word. If you really want to search for an asterisk or a question mark, you must add a backslash before the character. By enclosing the search string in double quotes, like so: "Hilfe", a verbatim search is forced such that only rows are displayed where the P value is (apart from case) strictly **identical** to the search string, rather than just being contained within it. So e.g. a search for `"wo"` in the **complementizer** field will only return examples with **complementizer**:*wo*, whereas a search for `wo` would also return ones with *worüber*, *wozu* etc. In addition, for extremely powerful string-filtering possibilities, **regular expressions** may be entered by enclosing them in slashes like this: `/ex.mple/`. The syntax of

the regular expressions follows standard Java conventions. Note that the usage of wildcards in regular expressions is very different from the system used in simple string search: For single arbitrary characters, a dot (.) is used; for an arbitrary sequence of characters a dot followed by a star (.*) is used.

When doing searches on strings (both with and without regular expressions) in the predicate table, make sure not to be confused by the way that values from multiple examples are presented together in the table. If, for example, a predicate has some examples with **complementizer**:*that* and others with **complementizer**:*wo*, this will be indicated in the **complementizer** column for that predicate as `dass|wo` in the predicate table. But this `dass|wo` is just an abbreviated way to display this information in the table. It is not a real value for the example property **complementizer**, i.e. it is not something you can do a string search on, so typing `dass|wo` or `das*wo` into the search box below the **complementizer** column will not work. If you do want to search for all predicates that have some examples with *dass* and some with *wo*, you will need to do an advanced search, with independent example criteria, with "**complementizer** contains `dass`" and "**complementizer** contains `"wo"`". Note that the double quotes around *wo* are necessary in this case to indicate that you mean exactly *wo*, and not something that just contains *wo*, like *wozu* or *worüber*.

8.2 Advanced search

8.2.1 Using the query builder

By checking the **use advanced search** checkbox, an advanced system of building complex, hierarchical search queries is activated. This additional system can be de- and reactivated at any time with the checkbox. The advanced search is tightly integrated with both tables; any change in the query is immediately reflected in the tables. The advanced search differs from the table filtering options in several fundamental ways:

- The search conditions formulated with the advanced search query builder apply to both tables simultaneously. This is possible due to the inheritance of properties discussed above – by default, the tables keep in sync (in the sense defined above), that is, both tables are a valid search result for the query. This nice behavior necessarily breaks when the user chooses independent example semantics (as defined above). See Section 8.2.2 below for more explanation on advanced search semantics.

- The system allows the user to formulate an arbitrary number of criteria, even multiple criteria concerning the same property. Further criteria can be added at any time using the + buttons. By default, each criterion selector offers all available search condition types; by clicking on the caret symbol next to the + button, the user may choose to add a criterion selector that only offers predicate-related criteria or only example-related ones. Note the color coding here again: criteria for example properties have an orange background, while those for predicate properties have a blue one.

- Arbitrary boolean combinations are supported. All conditions have a built-in option for negation; there is a special type of search condition called "group of conditions" that opens up a subgroup of (possibly negated) conditions connected by logical "or" or "and", which yields four types of logical connectivity: all/none/at least one/not all condition(s) is/are true.

The advanced search does not replace the filters of the two individual tables. Instead, the search result for an advanced query, as presented simultaneously in both tables, can be further filtered on a per-table basis, as if the advanced query were an additional "super-filter" present on both tables. A typical usage scenario would be to first formulate an advanced query and then investigate the subset of data obeying the constraints of this query by filtering and sorting the results in the two tables. See the tutorial in Section 9 for discussion of concrete examples of advanced searches.

8.2.2 Advanced search semantics

In what follows, we will refer to a filter set on a table as a **table constraint**, and a condition defined in the advanced search as an **advanced constraint**. When we unconditionally talk of constraints, we refer to either kind of constraint. By default, the two **constraint types** just mentioned are treated alike. This means that, when working with a specific table, the user can freely choose between adding a filter to the table or adding the exact same condition as an advanced constraint on the highest level – the result shown in this particular table will not change. Furthermore, in the predicate table, all constraints on (inherited) example properties are treated as referring to the same example, which means that the result set in the table shows all predicates P for which there is at least one example E that fulfills the table and the advanced constraints. This **single example semantics** is what keeps the two tables **in sync** such that the user can treat the two tables as two alternative views on the same dataset.

There are three checkboxes for advanced search options concerning the predicate table, to be found below the table. They provide users with powerful additional search possibilities, but the exact search semantics can be difficult to understand. Section 9 provides a tutorial on using these possibilities with several examples, but here are the basics:

- The **independent example criteria (table filters)** checkbox enforces "independent example semantics" for table constraints on (inherited) example properties, as already explained above.
- Analogously, the **independent example criteria (adv. search)** checkbox enforces "independent example semantics" for advanced constraints on (inherited) example properties. **This setting causes the two tables to get out of sync**, which essentially implies that the meaning of the advanced query is now different for the two tables — the predicate table is not a collapsed version of the example table any more, but rather presents the answer to a different question (which is why you will get a warning from the system if you try to switch from the predicate table to the example table view when it is checked). To give an example: We may formulate an advanced query with the following three conditions:
  - **verb mood** is *KONJ I*
  - **example type** is *zeroDecl*
  - **predicate contains** `/^anh.*/` (i.e., it begins with 'anh')

  In the example table, this gets you seven entries for the three verbs 'anhalten', 'anherrschen', 'anhören'. All examples have both *KONJ I* and *zeroDecl*. Correspondingly, the predicate table gives the collapsed version of this result, showing the three verbs — as long as independent example semantics for advanced search is deactivated. If you now switch on

the independent example semantics, you get a fourth verb, 'anheben', because the meaning of the advanced constraints has changed for the predicate table. The system now searches for verbs that have at least one example with *KONJ I* and at least one example with *zeroDecl*. Now, 'anheben' indeed has an example with *KONJ I* and an example with *zeroDecl*, but these are two different examples — which is why it does not appear by default. The example table is not affected by this advanced setting; so the two tables now answer different questions or, in other words, switching between the tables is, in general, not useful with this setting turned on.

- The **adv. search is separate query** checkbox also makes the tables get out of sync. It lets the system treat the advanced constraints as one query and the table constraints as another, independent query, and intersects the results of these two queries. This breaks the default rule stated above that "the two constraint types are treated alike". If, for example, you want all predicates that have an example with properties **A** and **B** and an example (possibly, but not necessarily identical to the first) with properties **X** and **Y**, you may formulate **A** and **B** as advanced constraints (with standard single example semantics, since **A** and **B** are required to hold of the same example!) and **X** and **Y** as table constraints (again with standard single example semantics) and turn on **adv. search is separate query** to get the verbs that fulfill your request. Without this option, all four constraints **A, B, X, Y** would be taken to hold for one and the same example. Normally, the only case where checking this option alters the result set displayed in the predicate table is when both other advanced settings are turned off, i.e. when single example semantics is chosen for both table and advanced constraints. The only exception to this rule stems from the special status of search constraints on **example text**, to which we now turn.

If the user chooses to include more than one (table/advanced) constraint on the **example text**, then all of these constraints are always taken to refer to the same example. This glaring exception to the general way the search system works has been introduced to avoid very long response times and may lead to queries that are hard to understand. We therefore recommend to use at most one such constraint. – If there is both a table constraint and an advanced constraint on **example text**, the setting of the **adv. search is separate query** checkbox has a bearing on the query results in the predicate table even if independent search semantics is turned on for table and/or advanced constraints.

8.3 Export function

This function is disabled during the public beta stage. It will be enabled in the first full release, once we have finished testing and can be confident in the general quality of the data and how they are presented.

You can get the output of a search as a file in .csv and .xls formats. Simply run the query you are interested in, then click the **download table data** button. After you enter the correct password (see Section 10.3 for how to obtain the password), the data currently displayed in the table will download in both formats, which you can then work with on your local machine, using R, a spreadsheet application, or whichever other tools you like. Note that the .csv file produced is not designed to be used with Excel, whereas of course the .xls file is. At present, exports are limited to a maximum of 200 lines from the table. If you export a larger table, it will still work, but you will only get the first 200 lines.

9 A tutorial on building complex searches

In this section we will walk through different types of complex searches to see how they can be constructed and especially how example properties and predicate properties can interact and be treated in different searches. Their behavior depends on the type of search being carried out (i.e. an example search or a predicate search), the way that the constraints on them are entered in, and the use of certain clickboxes in the search interface that affect how queries are interpreted. We will start with the simplest cases, working our way up gradually to quite complex advanced searches. Wherever possible, we will use concrete examples of actual searches to guide the discussion, often with screenshots.

9.1 Example table search

The simplest case is a basic search on the example table. This is the default view you will be given when first opening the database. You can recognize that this is the kind of search you're running when the mark next to **example table** is checked, and the one next to **use advanced search** is not. Conceptually, what you are doing here is searching for a list of examples that meet the various criteria that you set. In this case, example properties and predicate properties are treated essentially the same. The predicate properties are simply interpreted as applying to the predicate contained in the relevant examples. So for example entering `dass` in the **complementizer** field (an example property) will narrow things down to examples that contain the string *dass* in their **complementizer**. Analogously, selecting *Pt-V* in the **morphology** field (a predicate property) will narrow things down to examples in which the clause-embedding predicate is a particle verb.

9.2 Advanced example search

We can move up in complexity by checking the box next to **use advanced search**, while leaving things in **example table** mode. This gives us the ability to combine together constraints on example and predicate properties in far more flexible ways, including different logical connections and negation. Nonetheless, what you are doing here conceptually is not really any different from what you are doing in the basic search. You are still searching for a list of examples that meet a set of criteria — it's just that you have more power in specifying those criteria. As a result, example properties and predicate properties are still treated essentially the same way as each other. Figure 2 shows a search for examples in which the predicate is *bringen* or something derived from it, like *abbringen* or *beibringen*, and in which the embedded clause is either an infinitival or in the subjunctive I. Note that while **example type** and **verb mood** are example properties, **predicate** is of course a predicate property, but all of them are being treated essentially the same — examples are filtered out according to whether or not they fit the constraints placed on these properties. Things work out this way because there is exactly one predicate per example, so it is straightforward to treat the properties of the predicate as though they were properties of the example.

9.3 Predicate table search

Things start to get tricky when we do searches on the predicate table. We are of course doing something different here, in that we are looking for a list of predicates that fit certain criteria. Again, those criteria include constraints on predicate properties and constraints on example properties,

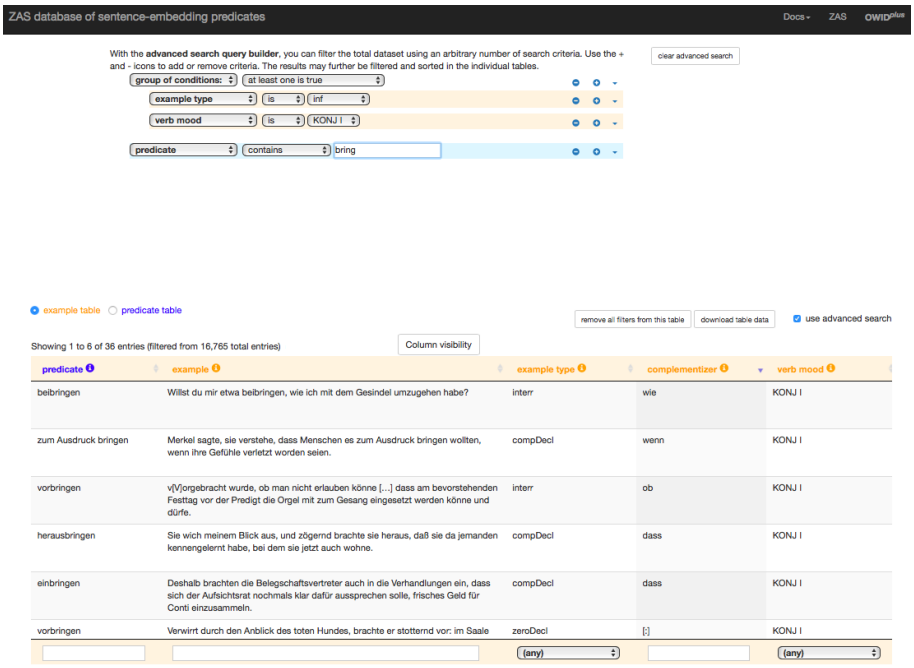

Figure 2: Examples **bringen* with *inf* or *KONJ I*

but conceptually the example properties are playing a rather different role here than anything we saw with the example searches. Ultimately this is because of an asymmetry in how predicates and examples relate to each other. As noted above, there is exactly one predicate for each example, but for each predicate there will usually be several examples, demonstrating various different kinds of embedding that it can do. This means that, while we can look for examples where the predicate is a particle verb, it doesn't make sense to look for predicates where the **complementizer** is *dass*. Instead, we have to look for predicates which are associated with some number of examples where the **complementizer** is *dass*. But this means that we have some choices to make, and we can do rather different kinds of searches, depending on what we intend by 'some number of examples' and on how we combine multiple conditions on example properties. The default and simplest thing would be to say that we are interested in all predicates that have at least one example that satisfies a particular criterion. So in the search in Figure 3, we are looking for all predicates that have at least one example with **verb mood**:*KONJ II*. It is relatively straightforward to understand how the example property **verb mood** plays a role in that search.

But now let's imagine something slightly more complicated, where we bring in a second example property. What if we add on the condition that the example have a *w*-**complementizer**, something we can search for with `w` in the **complementizer** field? That means we're looking for predicates that have examples in the subjunctive II and with *w*-**complementizers**, which sounds simple enough, but note that there are actually two very different ways to interpret this combination. We can't simply interpret it as 'show me every predicate where **the** example has an embedded clause in the subjunctive II and a *w*-**complementizer**', precisely because there isn't a unique example associated with each predicate, but usually several. Instead, we have to decide whether the multiple conditions on example properties are meant to apply to examples simulta-

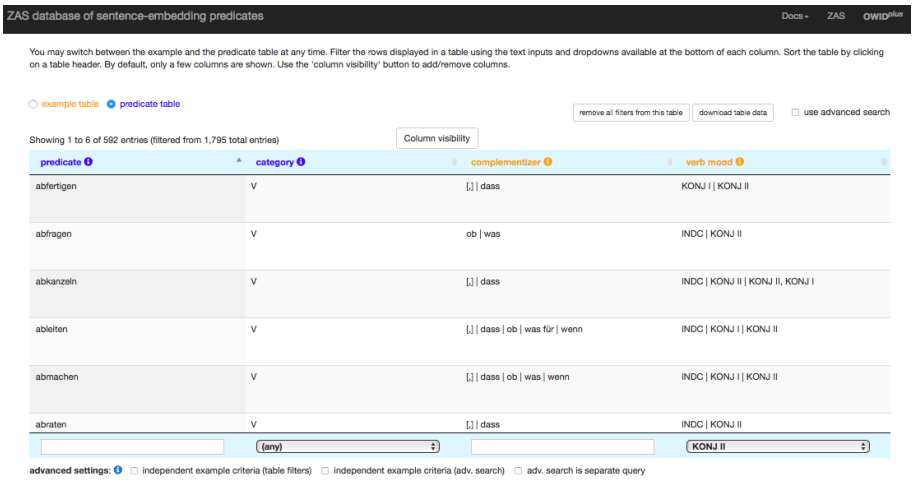

Figure 3: Predicates with *KONJ II*

neously or independently. That is, are we looking for predicates that have at least one example meeting all criteria, or are we looking for predicates that, for each criterion, have at least one example that meets the criterion? The first possibility would mean in the case at hand that we only want predicates that have at least one example that has a subjunctive II clause introduced by a *w*-**complementizer**, something like example (35):

(35) Erstmals hatte sich angedeutet, wo das noch hinführen könnte mit Rudolf Völler. (ZDB 337: DWDS BZ 2003)

'For the first time there was an indication of where this could be headed with Rudolf Völler'

This is what we refer to this as **single example semantics**, because all of the constraints on example properties are interpreted as applying to a single example. The second possibility would be satisfied by any predicate that has at least one example with a subjunctive II clause and at least one example with a *w*-**complementizer**, where crucially these need not be a single example. So for example this is satisfied for the predicate *absehen*, because while it has no single example displaying both properties, it has example (36-a) with subjunctive II, and example (36-b) with a *w*-**complementizer**:

(36) a. Deshalb konnte man schon im Voraus absehen, dass die Reform keine ungeteilte Freude hervorrufen würde (ZDB 22703: DWDS BZ 1995)

'Therefore one could anticipate in advance that the reform would not call forth undivided joy.'

b. So war es wohl abzusehen, was in den 80er Jahren passierte. (ZDB 22701: DWDS BZ 1998)

'Thus what happened in the 80s was predictable'

This second possibility is what we refer to as **independent example semantics**, because the constraints on the example properties are interpreted so that they can each apply independently, po-

tentially to different examples.

Now, these aren't just logical possibilities of how one **could** interpret a query. Both of them are legitimate types of searches that we might realistically be interested in carrying out, depending on what kind of research we are doing. Thus the search interface of the database is designed to let you specify what you want, and do either one. By default, constraints on example properties in predicate searches are interpreted with single example semantics, so we get the first scenario described above, where a single example must display all of the relevant properties. Figure 4 shows how we could do the search for predicates that have at least one example that has a subjunctive II clause introduced by a *w*-**complementizer** described above. Note that we only get 148 hits here, as this is

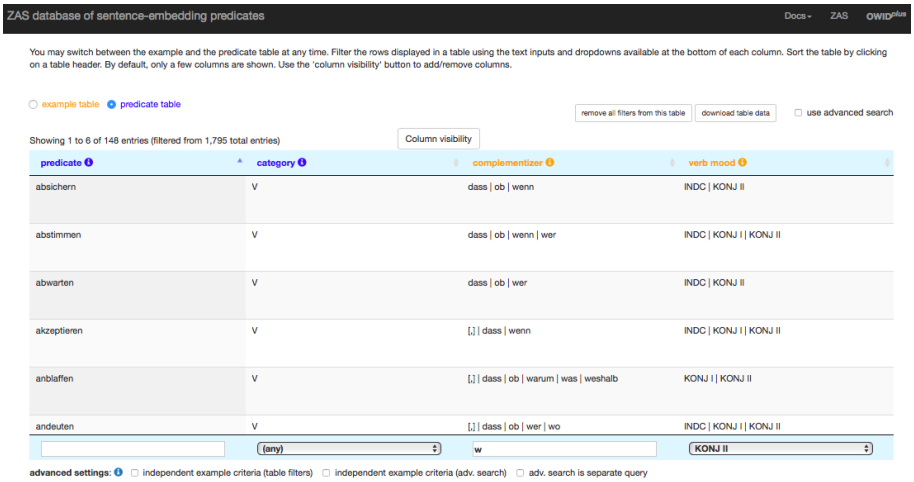

Figure 4: Predicates with *w*-**complementizer** with *KONJ II*

a rather specific restriction we've placed on our predicates. If instead we want the constraints to be interpreted with independent example semantics, we just need to tell the interface this by clicking the box next to **independent example criteria (table filters)** at the bottom of the table. This can be seen in Figure 5, which runs the query described above for predicates that have at least one example with a subjunctive II clause and at least one example with a *w*-**complementizer**, but not necessarily the same example. Note that now we get 455 hits, as this search is not so restrictive as the previous one.

9.4 Advanced predicate search

Now we are ready to consider the most complex type of search, an advanced search on the predicate table. Again, conceptually what we are doing here is not really different from what we are doing in the simple search. We are searching for lists of predicates that meet certain criteria which constrain both properties of the predicates themselves and of the examples that are associated with them. The difference is simply that we again have at our disposal the means to combine together such criteria using a number of different logical connectors. When we add this to the different possible semantics for example criteria, we end up with the ability to craft extremely sophisticated and powerful searches. But of course, this power comes from the ability to create complexity, and that complexity is not always easy to understand. First, just to show you that we can mimic any table filter search, figures 6 and 7 show ways to implement the two searches

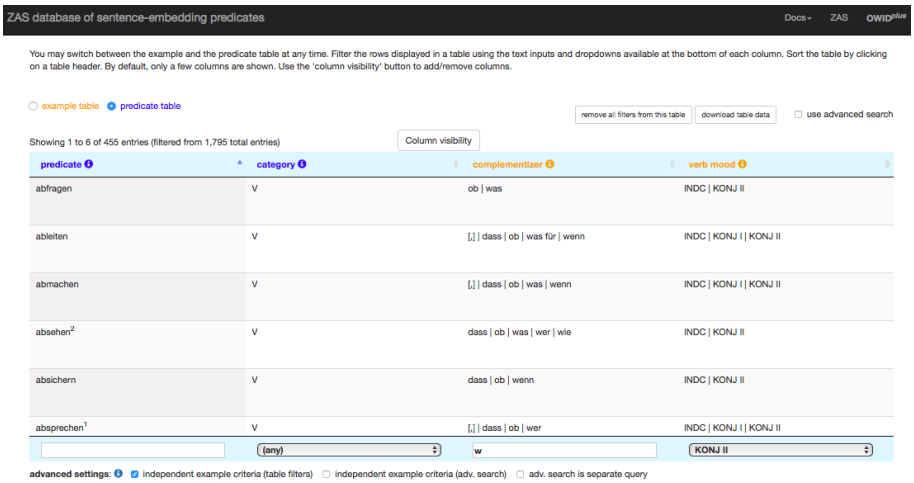

Figure 5: Predicates with *w*-**complementizer** with *KONJ II*, independent semantics

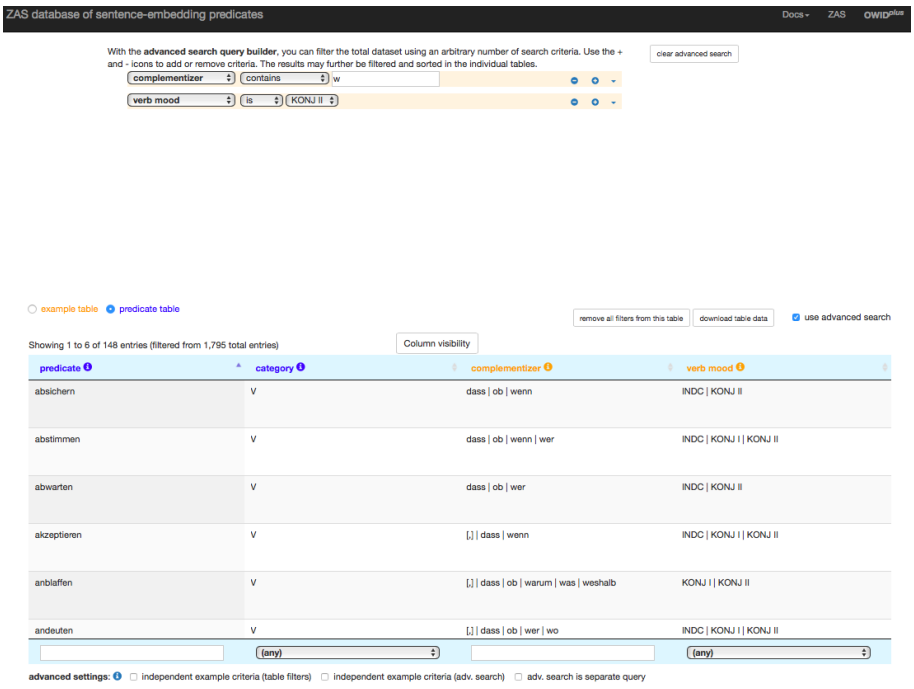

Figure 6: Advanced predicate search, *w*-**complementizer** with *KONJ II*

discussed immediately above using advanced search criteria. Note that we get the same numbers here, 148 and 455, as it should be. And note also that single example semantics is the default with the advanced search as well, and independent example semantics is achieved in Figure 7 by checking the box next to **independent example criteria (adv. search)**.

So what else can we do with the advanced search that wouldn't have been possible otherwise? One thing is that we can have two distinct constraints apply to the same example property (or the same predicate property, for that matter), potentially with distinct logical connectors. Imagine

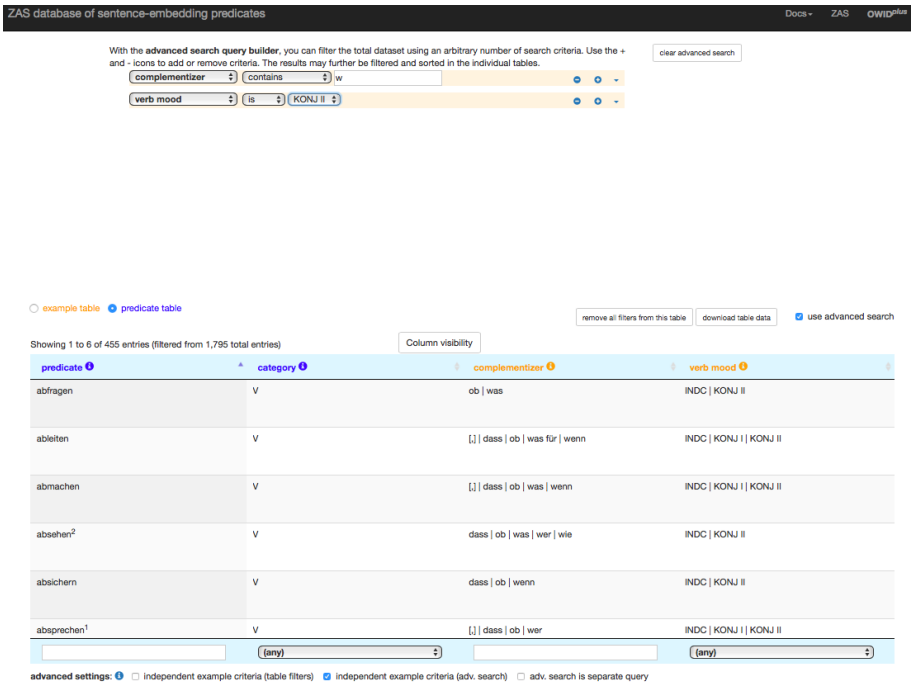

Figure 7: Advanced predicate search, *w*-**complementizer** with *KONJ II*, independent semantics

that you want to search for all predicates that take examples with a *w*-**complementizer** other than *wenn*. Figure 8 shows how you can do that. So we have one constraint that says we want predicates that have at least one example with a **complementizer** that starts with *w*. Then we have another constraint that says we want predicates that do **not** have an example where the **complementizer** is *wenn*. In order to satisfy the query, a predicate must meet both of these criteria, and so we end up with all of the predicates that embed some *w*-**complementizer** but do not embed *wenn*.

Note that we get the particular semantics of the constraint mentioning *wenn* because we've again checked the box **independent example criteria (adv. search)**. If we hadn't checked that box, the interpretation would have been that we want predicates that have at least one example where the **complementizer** is something that starts with *w* but is not *wenn*. It's okay if the predicate has an additional example with **complementizer** *wenn*, but it crucially needs to have something with *wer* or *was* or something like that.

Now, there is something important here that may not be so obvious and requires a bit of discussion. This is the way that negation of example criteria is handled in these advanced predicate searches, which changes a bit depending on whether we're using single example semantics or independent example semantics. Let's say that we're interested in verb-final embedded clauses where the **complementizer** is not *dass*. We can set up an advanced search with two criteria: '**word order** is *VLast*', and '**complementizer** does not contain `dass`'. The important thing here is the negation in 'does not contain', and the question is how this is applied. With the default single example semantics, things are relatively straightforward. The query, given in Figure 9, looks for all predicates, which have at least one example which is **word order**:*VLast* and has a **complementizer** that is not *dass*. So this means that a predicate can perfectly well have additional examples that have *dass* as the **complementizer**, and they will still satisfy this query, as long as they have one ex-

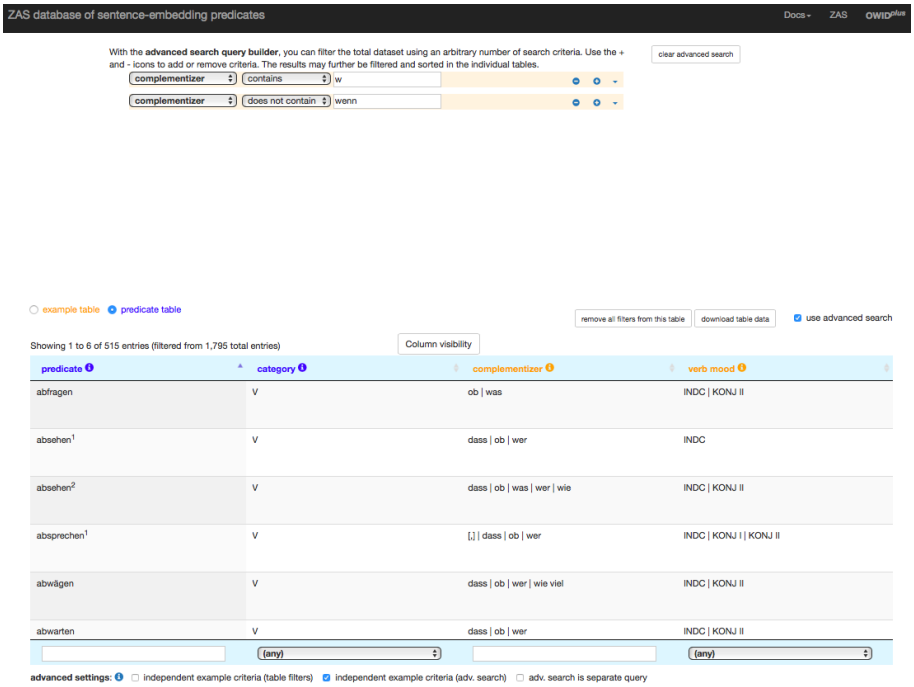

Figure 8: Predicates with *w*-**complementizer**, but not *wenn*

ample with a different **complementizer**. This a very weak restriction which is satisfied by the vast majority of predicates in the database, and the query gets 1710 hits. For the semantically-minded out there, we say that, with single example semantics, negation takes narrow scope relative to the examples, i.e. 'there is an example, such that it is not the case that…'

With independent example semantics, things are different. If we run the same query, but this time with **independent example criteria (adv. search)** clicked, we only get 66 hits, as in Figure 10. The way this is interpreted is that we look for all predicates where '**word order** is *VLast*' holds of an example, and it is not the case that '**complementizer** contains `dass`' holds of an example. We've had to word this a bit oddly to get it right, but hopefully it is clear what is intended here. When we have independent example semantics, negation on a criterion takes wide scope relative to the examples. That is, it is interpreted as 'there is no example that has **complementizer** *dass*', whereas with single example semantics, it was 'there is at least one example that does not have **complementizer** *dass*'.

We can think about things like this. With single example semantics, you go through the examples of a predicate, looking for one that can satisfy all of the example criteria. With independent example semantics, you go through the example criteria, and check whether each one is satisfied by the collection of examples for a predicate. If the constraint is positive, this is fairly easy. To satisfy the constraint 'there is an example with property **X**', you just need one example that matches. But if the constraint is negative, something rather different happens. To satisfy the constraint 'there isn't an example with property **X**', you need to go through all of the examples and make sure that none of them matches.

Let's consider one more example to see how things work. Say we want to find all of the predicates that can embed interrogatives and verb-second clauses, but **not** finite declaratives with

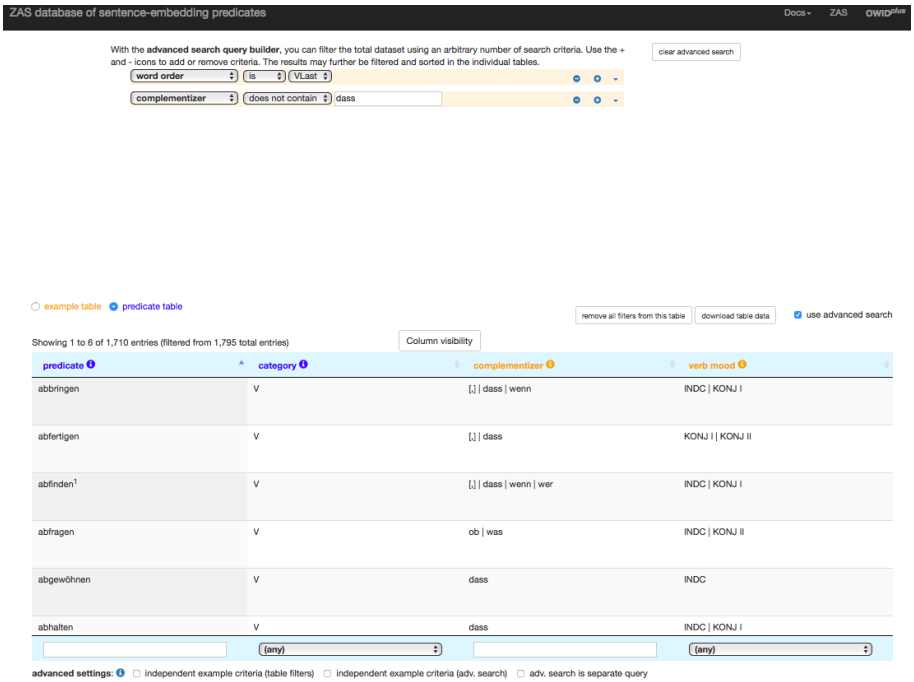

Figure 9: *VLast*, not *dass*

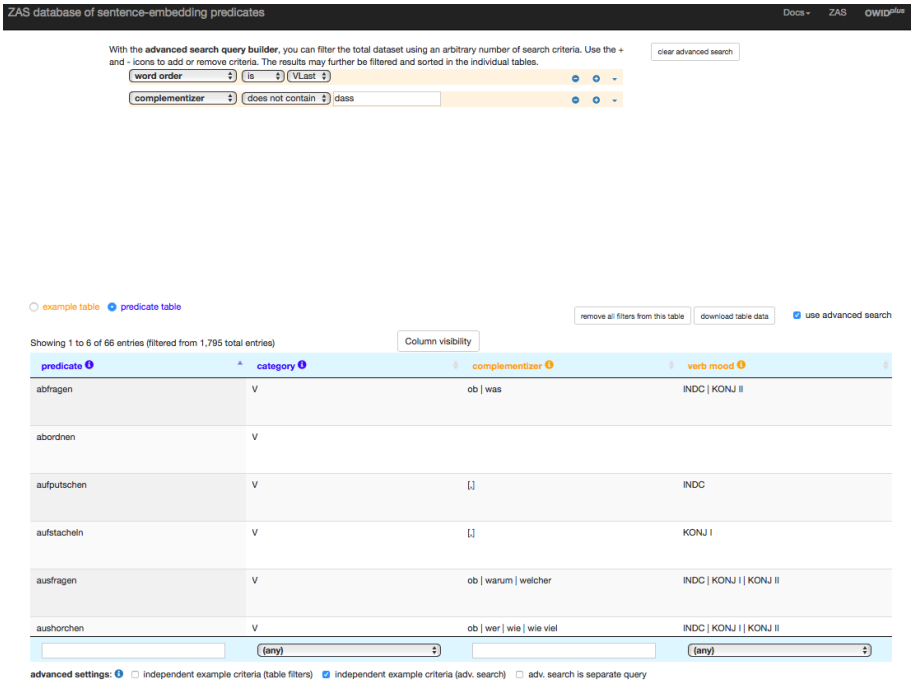

Figure 10: *VLast*, not *dass*, independent semantics

a complementizer. Figure 11 shows how we do it. We start with two simple '**example type** is' constraints, encoding that we want only predicates that have both *interr* examples and *zeroDecl*

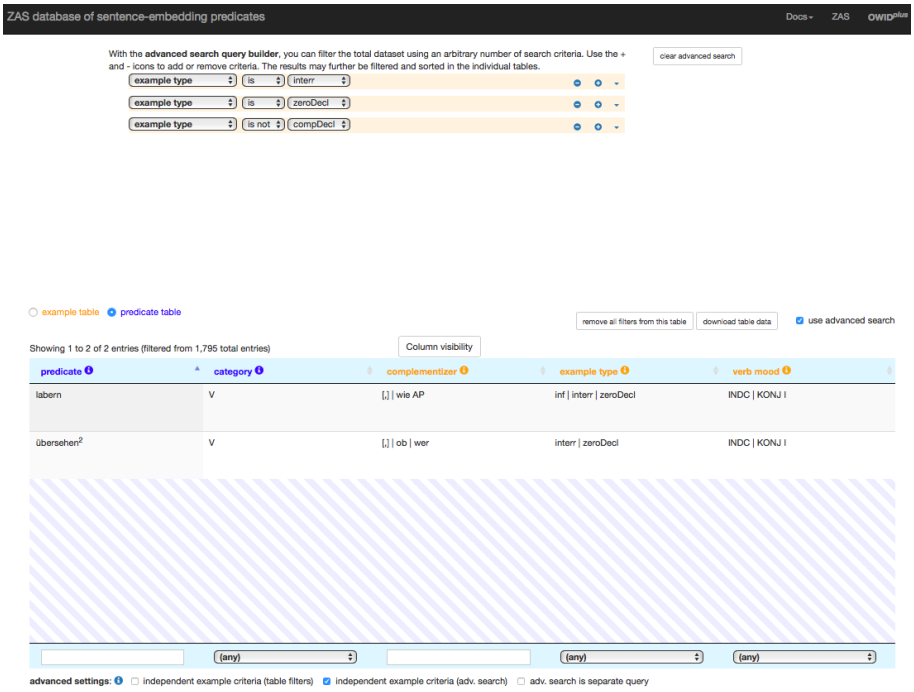

Figure 11: *interr* and *zeroDecl*, not *compDecl*

examples. Then we have the constraint '**example type** is not *compDecl*'. Because we have clicked **independent example criteria (adv. search)**, this gets interpreted to mean that we want predicates which have **no** examples with **example type**:*compDecl*. So putting it all together, we get what we want: predicates that embed interrogatives and verb-second clauses, but no declaratives with a complementizer. Note that things would have come out rather differently if we hadn't checked **independent example criteria (adv. search)**. We would then be looking for all predicates which have at least one example where the **example type** is *interr*, and the **example type** is *zeroDecl*, and the **example type** is not *compDecl*. Of course, the first two constraints aren't consistent with each other, since a single example can't simultaneously have two different **example types**, so this search would return zero hits.

There is one additional way we can play around with how example properties are treated in an advanced predicate search. Note that there are two separate checkboxes for **independent example criteria**, one applying to the table filters and the other applying to the advanced search. This means that you can be using single example semantics in one place and individual example semantics in the other. Let's say that for some reason you are interested in predicates that can take *interr* complements in *KONJ I*, but can also take *inf* and *zeroDecl*. Figure 12 shows how you would search for that. Note that we haven't checked **independent example criteria (table filters)**, because we crucially want to pick out single examples that have **example type**:*interr* and **verb mood**:*KONJ I*. At the same time, we **have** checked **independent example criteria (adv. search)**, because the two criteria there looking for *inf* and *zeroDecl* examples obviously cannot apply simultaneously (with each other or with the constraints in the table filters) to a single example, or they would lead to a contradiction. The independence of the two checkboxes lets us build clever queries with mixtures of single example semantics and independent example semantics.

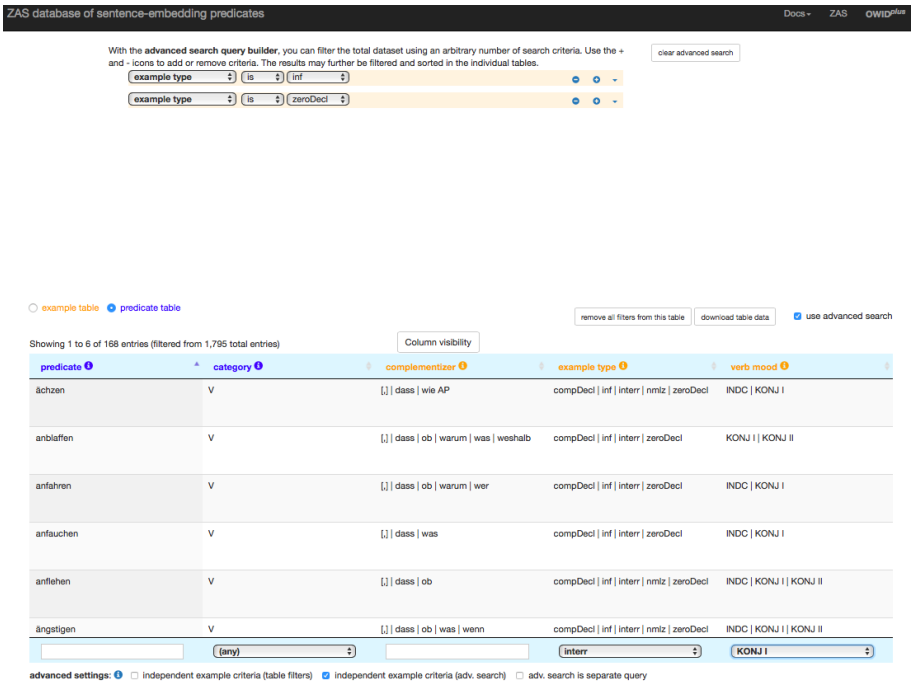

Figure 12: *interr KONJ I*, *inf* and *zeroDecl*

That's all well and good, but what if we want to have two sets of constraints that apply to examples independently, but where the constraints within each set apply to single examples simultaneously? Let's say that what you really care about are predicates that can embed *KONJ I interr* and *KONJ II zeroDecl* clauses. You need single example semantics twice, because you care about examples that are *KONJ I* **and** *interr*, as well as examples that are *KONJ II* **and** *zeroDecl*, but they need to be combined together via independent example semantics, because you obviously can't have a single example that is *KONJ I* **and** *KONJ II*, or interrogative **and** *zeroDecl*. For this, there is a further special checkbox, which doesn't turn on independent example semantics in either the advanced search or the table filters, leaving them both with single example semantics. Instead, it simply makes them independent of each other, so each is treated like a separate query with single example semantics, and then you put them together and get as a result all of the predicates that satisfy both queries individually. How it looks for our specific example can be seen in Figure 13.

10 Correct use of the database

This is release 0.2 (Public Beta) of the ZAS database of clause-embedding predicates, © 2017, Leibniz-Zentrum Allgemeine Sprachwissenschaft, Berlin (http://www.zas-berlin.de). It is made available here, in collaboration with the Institut für Deutsche Sprache, Mannheim (http://www.ids-mannheim.de), for private and research use. You may freely search the database using the interface on this website, and you may reproduce data obtained via such searches, or report statistics or other analytical results based on those data, as long as you follow the guidelines laid out below for citation and attribution. You may also download and make use of data via the pro-

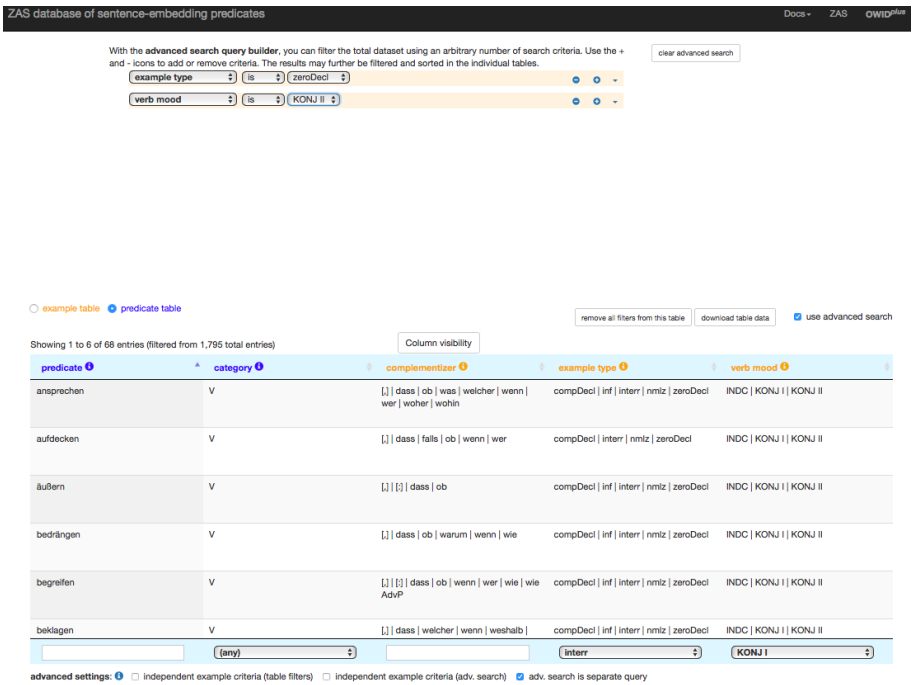

Figure 13: Advanced search is separate query

vided 'download table data' function and carry out your own processing and searches of those data on your local machine **once this function is activated in the first full release**. Any such data reproduced in your own presentations and publications or distributed in any other way must be accompanied by appropriate citation and attribution, as described below.

10.1 Citation

If you use or reproduce any of the data in the database in a publication or presentation, you should cite the database as follows:

Stiebels, Barbara, Thomas McFadden, Kerstin Schwabe, Torgrim Solstad, Elisa Kellner, Livia Sommer and Katarzyna Stoltmann. 2017. *ZAS Database of Clause-embedding Predicates*, release 0.2 (Public Beta) in: OWID^{plus}, hg. v. Institut für Deutsche Sprache, Mannheim, http://www.owid.de/plus/zasembed2017.

When doing so, please always double-check that you have the current release number. We aim to keep the data in the database as stable as possible across releases, and things like example **ID**s will not change from one release to the next, but we will be adding new data, and of course fixing any errors that we become aware of. Indicating the correct release number will ensure that people using a later release are not confused by mismatches in the data available to them.

10.2 Example source information

If you reproduce any example sentences from the database, in whole or in part, they should always be accompanied by **example ID** and **source** information, so that it is always clear what the original

corpus source was. We suggest a format with the abbreviation ZDB (for 'ZAS Database'), followed by the **example ID** and a colon, and then the full **source** tag from the example sentence. If it is made clear in the body of the paper or presentation that an example or series of examples are taken from the database, you can save space by leaving off ZDB and just giving the **ID** number followed by the **source** tag. So the following would be appropriate renderings with the long and short version of the source information, respectively:

(37) a. Sein Arzt riet ihm dringend von *einer Teilnahme am Super Bowl* ab. (ZDB 17688: DWDS TS 2005)
b. Sein Arzt riet ihm dringend von *einer Teilnahme am Super Bowl* ab. (17688: DWDS TS 2005)

10.3 Registering for the 'download table data' function

*****This function is disabled during the public beta stage. It will be enabled in the first full release, once we have finished testing and can be confident in the general quality of the data and how they are presented.*****

As described in section 8.3, it is possible to have the search interface export data output as the results of a search query so that you can save it for offline use. In order to enable this functionality, we request that you complete a free registration, by sending an email with the subject 'ZAS Database registration' to `mcfadden@zas.gwz-berlin.de` with the following information:

- Your name
- Your institutional affiliation, if any
- A note on how you heard about the database
- A sentence or two describing why you are interested in the database and what you intend to use it for

We will then send you a password for the export function. Note that this is not a real security mechanism or an effort to restrict access to the data. Rather, it is a simple way for us to collect some information about who is using the database and what for, and there are no wrong answers to the last question. We thus request as a professional courtesy that you complete the registration individually and do not share the password with others, even within a single institution. After all, the registration is free and relatively painless. Any information you send us as part of your registration will of course be kept private and will not be shared outside of the ZAS Database Team.

10.4 Feedback

The database is a work in progress and will be periodically updated, both with new data and with corrections of the old. We would therefore very much appreciate it if you report to us any errors or problems, either with the data or with the search interface. Feedback can be sent via email to Thomas McFadden at the following address:

`mcfadden@zas.gwz-berlin.de`

References

[1] Bierwisch, Manfred. 1983. Semantische und konzeptuelle Repräsentation lexikalischer Einheiten. In *Untersuchungen zur Semantik*, ed. Rudolf Růzička and Wolfgang Motsch, 61-99. Berlin: Akademie Verlag.

[2] Karttunen, Lauri. 1977. Syntax and semantics of questions. *Linguistics and Philosophy* 1. 3-44.

[3] Levin, Beth. 1993. *English verb classes and alternations: a preliminary investigation*. Chicago: University of Chicago Press.

[4] Stiebels, Barbara. 2010. Inhärente Kontrollprädikate im Deutschen. *Linguistische Berichte* 224. 391-440.

[5] Stiebels, Barbara. 2011. Von den Herausforderungen des lexikalischen Reichtums. *Geisteswissenschaftliche Zentren Berlin: Bericht über das Forschungsjahr 2010*. 51-72. http://www.zas.gwz-berlin.de/fileadmin/material/jahresberichte/Reflexionen_2010.pdf

[6] Wunderlich, Dieter. 1997. Cause and the Structure of Verbs. *Linguistic Inquiry* 28:27-68.