User's Handbook for the Icelandic Wordweb

Hjalti Daníelsson
The Árni Magnússon Institute for Icelandic Studies

1. Introduction

1.1 About this handbook

This is a user's handbook for the new version of the Icelandic Wordweb ("Íslenskt Orðanet").

The original Wordweb is a database of words, categorizations and word relations, presented through a web site tailored for specific ways in which to use its data. This new version consists of a single, large RDF file that houses the original's content and is encoded with a standardized vocabulary. Although the new format opens up an array of new options for exploring the Wordweb, its open-ended nature may be challenging at first, particularly for those unfamiliar with the original's data and design.

This handbook thus has the aim of making the Wordweb more accessible for its users. It describes the original Wordweb's architecture, explains how those features have been implemented in the new format, and demonstrates some of the ways it may be used and explored. Some familiarity with OntoLex, SKOS, SPARQL, and RDF triplets might be required, although we do our best to explain the most important facets of each one.

Please note that this handbook is not meant as an exhaustive list of the ways in which the Wordweb may be traversed and used. Rather, it is intended to clarify the primary aspects of the Wordweb's design and possible use, and to point out the ways in which it maintains and expands on the features of the original. In addition, it should be kept in mind that the models in which the new Wordweb was implemented are immensely flexible, and there are often many other ways in which we could have applied them to implement the Wordweb's ceatures. In our design of the new Wordweb, we've generally tried to hew close to how these models were intended to be used, and not break convention any more than necessary. While we do on occasion mention how certain limitations guided that design, we are not necessarily implying that these limitations couldn't have been overcome or circumvented with different solutions, but simply that the particular design path we chose seemed sensible in context.

In chapter 2 we provide the necessary background on the original Wordweb, and on the two models, OntoLex and SKOS, in which the new version was implemented. This is also the last chapter where these entities are discussed separately. Each of the ensuing chapters examines a particular Wordweb feature, demonstrating its design in the original Wordweb and the corresponding implementation and use in OntoLex and SKOS: Chapter 3 explores the Wordweb's basic elements, chapter 4 explains the relationships between these elements, and chapter 5 looks into some of the more specialized functionality.

1.2 Quick start guide

For those who simply want to set up a database for SPARQL queries, please follow these steps. More information on each step may be found in chapter 5 of this handbook. **Please note that the author of this handbook, and The Árni Magnússon Institute for Icelandic Studies, can not assume responsibility for any issues, including software incompatibility or data loss, related to the content in this handbook**.

- Go to https://jena.apache.org/download/index.cgi and download Apache Jena. For Linux users, select the filename ending in ".tar.gz". For Windows, select ".zip". You do *not* need to download Apache Jena Fuseki from the same page.
- Go to https://repository.clarin.is/repository/xmlui/handle/20.500.12537/100, download the wordweb.rdf zip file, and unzip it to the /bin/ subfolder of your Apache Jena software.
- Create a new, empty folder named TDB. Make sure you have several Gigabytes free on that disk.
- Depending on your operating system, refer to the sections "Windows setup instructions" or "Linux setup instructions" in chapter 5.1 on how to proceed.

2. The models

In this chapter we look at the architecture of the original Wordweb, and discuss certain basic features of the models in which the new version was implemented. This is by no means an exhaustive list: The goal here is simply to ensure sufficient understanding of these models so that later chapters will make sense to the reader.

2.1 The original Wordweb

The Wordweb is a semi-categorized collection of lemmas and their relations. To better distinguish the two core Wordweb entities, we'll capitalize them from here on: Lemmas ("Fletta" singular / "Flettur" plural) and Concepts ("Hugtak"/"Hugtök").

Lemmas come in many varieties: Monolexical (single-word) and polylexical (multi-word), unordered and ordered, sourced both from reference works and primary sources, and sometimes accompanied by various types of explanatory and morphosyntactic metadata. Lemmas generally do not have definitions except in cases where it's necessary to differentiate word forms, rather, their meanings are considered to be implicit in the relations they have to other Lemmas.

The relations between these lemmas are likewise a mix of different types. There is a heavy slant toward semantic relations, meaning that lemmas tend to be connected if their respective meanings have some common factor, but there is also a syntactic aspect.

The categorization schema is composed of Concepts. While it is quite extensive, it is not intended to be all-encompassing. Each Lemma may belong to one, none, or multiple Concepts. Their schema has a flat structure: All Concepts have equal priority, there are no Concept hierarchies, and from a semantic perspective their subjects may overlap. Although Concepts exist as separate entities in the Wordweb (not just as properties of Lemmas), there are no direct Concept-to-Concept relations; they are considered to be related only through the Lemmas that belong to them.

One of the Wordweb's greatest strengths is its magnitude. It is an extensive and growing collection, numbering well over 200,000 Lemmas and innumerable relations of various types. It contains a wealth of semantic relations that may seem obvious to the reader but cannot be found in more heavily curated and structured reference works.

But with this magnitude comes a degree of complexity, particularly in terms of encoding all that information in machine-readable fashion. This is compounded by the fact that the Wordweb has been built on and expanded over a development period of many years, through the work of several people employing multiple sources & databases. As a result, there is no single, definitive type of Lemma in the original Wordweb's database tables. Some of the Lemmas may be written or encoded differently from others, some have more metadata, and in certain cases the metadata itself may also be encoded differently between Lemmas.

In short, a direct conversion to an established format is simply not possible. The only way of standardizing the Wordweb while simultaneously maintaining its unique functionality has been to design and implement it in the new format almost from scratch. In addition, we've incorporated as much information as possible - including data that was unavailable in the original - while also trying to establish a more standardized storage.

2.2 OntoLex and SKOS

While there is no single standardized model capable of containing the entirety of the Wordweb's data, each of its two core systems - Lemmas and Concepts - lends itself quite well to a specific model: OntoLex and SKOS, respectively.

OntoLex (previously known as "lemon", short for "The Lexicon Model for Ontologies") is intended primarily to codify grammatical and syntactical information in linked data collections. This includes varying morphosyntactic forms and metadata, of which there is a great wealth in the Icelandic language. An important aspect of OntoLex is that while it is very suitable for storing this kind of information, it is generally neither applicable nor intended for other purposes such as categorization. We use OntoLex to store Lemmas, most of their metadata, and some - but importantly, not all - of their relations.

SKOS, on the other hand, is intended for classification and categorization. It does not store grammatical information, and generally is better suited for handling groups of entities rather than representing each one individually. We mainly use it to store the Wordweb's Concepts, and to represent certain complex relations. (The basic SKOS unit is also called a "concept". To avoid confusion, we'll use "Concept" for the Wordweb entity, and "concept" or "SKOS concept" for the SKOS unit). There are two particular properties of SKOS that have proven very useful when implementing the new Wordweb. Firstly, since it is intended for a more general purpose than OntoLex is, it is also quite a bit more flexible. Secondly, its focus is on the relations between its entities rather than information on the entities themselves. Certain core Wordweb features straddle the gap between semantics and syntax, which made them a difficult fit for OntoLex. In those cases, we were able to leverage SKOS's flexibility and extensive relations vocabulary instead.

We'll now look at each aspect of the Wordweb, go into more detail about its original design, and explain how it was implemented - and should be handled - with the new models.

3. Elements

3.1 Lemmas

Lemmas are the core entities of the Wordweb, and all of its information - including Concepts - derives from them in some way.

Each Lemma always consists of at least three parts: A series of one or more words, with a specific collective meaning, and certain grammatical information. In OntoLex, these are called the Lexical Entry, Lexical Sense, and Form, respectively.

Lexical Entries

Although we will generally use the terms "Lemma" and "Lexical Entry" interchangeably, Lexical Entries are in fact only the word or words that make up each Lemma, shorn of meaning and grammar. Lexical Entries are the absolute core of OntoLex; most of its functionality and data is meaningless unless attached to one of those.

It should be noted - and we will discuss this later in the handbook - that although Lexical Entries don't need to contain relations with other Lexical Entries (nor really any data other than their actual words), OntoLex requires them to be connected to a corresponding ontology entity, via Lexical Senses. A Lexical Entry without this kind of connection is, by OntoLex's definition, invalid. In our case, these ontology entities are the Concepts. In chapter 3.2 we detail how we dealt with this particular restriction.

One of the few OntoLex properties that can be attached directly to Lexical Entries, and which we employ in the Wordweb, is its multiplicity: All Lemmas are identified either as monolexical (single-word), polylexical (multi-word) or affixes. Monolexical Lemmas may generally be considered lexemes, with each Lemma presented in its canonical form. Keep in mind, however, that as with many other languages, Icelandic has its share of words whose standard version may not be in the default form (such as "skæri", which shares the plurale tantum of its English "scissors" counterpart). As a result, monolexical Lemmas can not be assumed to always have the same particular syntactic properties, even in their dictionary form, and need to be accompanied by grammatical marks.

Polylexical (multi-word) Lemmas, meanwhile, are considered phrases, since they stand together as a conceptual unit, and may sometimes be idioms as well. Lemmas do not need to be fully

formed components or subclauses. For example, the Lemma "detta af hesti" translates to "fall off a horse", not "to fall off a horse" nor "he fell off a horse".

Lexical Sense

A Lexical Sense is not a "definition" in the traditional sense, inasmuch that it does not textually define its Lexical Entry. Instead, a Lexical Sense identifies a particular usage of a single Lexical Entry, by linking it with a specific ontological element (i.e. a Concept). In most cases, a Lexical Entry will only have one corresponding Lexical Sense. If the Lexical Entry represents a concept that has multiple definitions or meanings, then each of those meanings will have its own Lexical Sense.

The point about Lexical Senses deriving their meaning from their links with ontological elements is important. Recall that in the Wordweb, not all Lemmas belong to a single Concept. Some belong to many Concepts, others to no Concept at all. But in OntoLex, the function that a Lexical Sense represents, from a Lexical Entry on one end to an ontological element on the other, has to be strictly one-to-one. Because of this issue, along with a separate one that has to do with how the Wordweb models its semantic relations, we created a special "encapsulation layer" between the OntoLex and SKOS implementations that serves as a connecting point for all Lexical Senses. We detail its design in chapter 3.2.

Forms

A Form is a specific written representation of a Lexical Entry, and may contain descriptive metadata to help the user identify its grammatical properties. In the Wordweb, this metadata is mostly morphosyntactic, although the precise contents vary between monolexical and polylexical Lemmas - the former tend to have far more detailed information - and between different grammatical categories. Most of this data is derived from The Database of Icelandic Morphology (DIM), and was stored separately in the original Wordweb's database tables.

For monolexical Lemmas, the new Wordweb distinguishes between the inflectional forms of nouns, adjectives, verbs, adverbs, and pronouns. Noun properties are gender, declension, number, and the definite article. Adjective properties are degree, gender, declension, and number. Verb properties are voice, mood, tense, person, number, gender, declension, and declension strength. Adverbs have only one property, degree. Pronoun properties are gender, declension, and number. For both mono- and polylexical Lemmas the Wordweb also specifically recognizes exclamations, conjunctions, prepositions, numerals and proper nouns, and specifically for polylexical Lemmas it designates set phrases ("orðastæða"), phraseological units ("orðsamband") and idioms ("orðtak"). (In reality, the first two types are very similar. They have their own dedicated marks in the original Wordweb's database tables - a special "x" mark for the former, and a separate "OSAMB" designation in a dedicated database table for the latter - but are effectively almost the same thing. In the Wordweb, phraseological units tend to be complete

phrases or idioms, with a meaning that is clear, definite, and well removed from the meaning of their individual words. Set phrases, meanwhile, generally don't have all the words that would be needed for a fully formed unit, and their meaning tends to be more obscure and less clearly defined.)

Note that for each word in a polylexical Lemma, generally only the grammatical category is specified. There are several reasons for this, but in brief, single-word data for the Wordweb's polylexical entries is quite challenging to establish, implement, and represent. The easiest and most efficient way - particularly in a single-file implementation, where we want to avoid needless repetition of metadata - is to create a link between the individual words in a polylexical Lemma, and their monolexical Lemma counterparts. We've done this wherever possible, and also pointed to the monolexical Lemma's specific morphosyntactic form when appropriate.

An important factor in our design and implementation of Forms is efficient use of space. As can be seen by the list of properties for monolexical Lemmas, Icelandic words tend to have a high number of morphosyntactic variations, each of which has its own written representation. To prevent the Wordweb's file container from absolutely blowing up, we've created standardized Forms for every possible combination of every grammatical category: One Form for all "noun-masculine-nominative-singular-no definite article" words, another Form for all "noun-masculine-nominative-plural-no definite article", and so on. Every morphosyntactic variation of every monolexical Lemma is thus assigned only two properties: Its written representation, and a link to one of these standardized Forms.

In the RDF file, our naming scheme for Lemmas is "[dictionary form]_[grammatical category]_fl". Recall that while the same Lemma may have different meanings, these are distinguished through the use of Lexical Senses. Thus the only real identifier we need for each Lemma is its grammatical information, not its meaning. In most cases the grammatical category is sufficient, although for a few rare cases some additional information has been encoded to ensure each Lemma's identity remains clear. The "fl", meanwhile, is shorthand for "Fletta", Icelandic for the Wordweb's Lemmas, and is intended to help distinguish them from the Concepts and LaCs described in an earlier chapter.

Special characters: [Square] and <angled> brackets

The original Wordweb contains certain special characters whose role is primarily to imply, rather than outright state, certain properties specific to the entry in question. Some of these properties are related to grammar, others to meaning, and yet others to both.

Unfortunately, encoding these kinds of properties in machine-readable format is not a straightforward task. These properties rely strongly on implied meaning and appropriate use, which is nontrivial to codify, and they're being used in a system whose design is in good part

based on representing various degrees of meaning without explicitly defining them or providing examples. To be of any real use these properties generally depend on human interpretation and a good working knowledge of the Icelandic language.

As a result, some of them, particularly those involving <angled brackets>, could not have any particular functionality encoded in the new Wordweb. They have been left as-is. (If they were to be encoded at a later date, the most likely option would be to represent them as so-called "arguments.")

Others, however, were brought over but encoded somewhat differently from the source versions, in order to enhance their functionality in the new system. The most significant type of these is [square brackets], which serve as a kind of wildcard for meaning and tend to play (very informally) the role of an intensional definition: "dökkrauður [hestur, kýr]" ("dark-red [horse, cow]").

For any Lemma whose original form contained square brackets, we dropped the brackets and instead created a new Lexical Sense for the remainder of that Lemma, with the bracket contents attached to said Lexical Sense as Usage examples. If the remainder already existed as its own Lemma, we simply attached the Lexical Sense to it. If we look to the example mentioned earlier, this involved parsing "dökkrauður [hestur, kýr]", creating a new Lexical Sense whose Usage property is "hestur, kýr", and attaching that Sense to the preexisting Lemma "dökkrauður". (If no "dökkrauður" Lemma had existed, the parsing process would have created one.)

3.2 Concepts, LaCs, and the SKOS encapsulation layer

The Wordweb's Concepts are implemented as SKOS concepts. They represent the ontology that OntoLex needs in order for its Lexical Entries and Lexical Senses to work.

Concepts are a much less complex entity than Lemmas. Their only properties are their individual names, and the only formal relations they have in the Wordweb are with the Lemmas that belong to them. The entire Concept structure was constructed by design, rather than from direct sources, and functions as an addendum of convenience rather than an all-encompassing system.

Thanks to the malleability both of the Wordweb Concept system and the SKOS model in which we've encoded it, we were able to extend its functionality to cover two OntoLex prerequisites that would otherwise have proven rather problematic. One - the requirement of one and exactly one Lexical Sense connecting each Lemma/Concept pair - has already been described in chapter 3.1. The other will be detailed in chapter 4, but in brief, we had to implement a specific

type of relation between certain Wordweb entities, where the entities had to be encoded in OntoLex but the relation couldn't be represented in OntoLex.

In both cases, what we really needed was the ability to connect every OntoLex Lexical Entry to exactly one SKOS concept, no more nor less. We therefore created the aforementioned encapsulation layer around the entire Concept system and populated it with what we termed "Lemmas-as-Concepts" or LaCs. Each LaC is effectively a combination Lexical Entry and Lexical Sense, recreated as a SKOS concept. Note that in the actual RDF file, we can then comfortably employ the SKOS "broader" and "narrower" properties to create a two-way link between each LaC and Concept pair.

It may help to think of LaCs as adaptor plugs that ensure all Lemma-type entities can be connected to the Concept system. Thus, instead of the connection between the two systems being represented as follows, where no Lexical Sense can be established for Lemma_B, and thus OntoLex cannot properly represent Lemma_B itself:

Lexical Entry		Lemma_A		Lemma_B		Lemma_C
		↓		↓		↓
Lexical Sense		✓		X		✓
		↓				↓
SKOS concept		Concept_A		[No concept]		Concept_C

We now have fully-fledged connections for all Lemmas, as follows, where all Lemmas are plugged into Concept objects and thus are considered valid entities by OntoLEx:

Lexical Entry		Lemma_A		Lemma_B		Lemma_C
		↓		↓		↓
Lexical Sense		✓		✓		✓
		↓		↓		↓

Lemma-as-Concept		LaC_A		LaC_B		LaC_C
		↓				↓
SKOS concept		Concept_A		[No concept]		Concept_C

We can then use the SKOS "senseOf" keyword to relate one LaC to another without implying precisely what type of relation this is (as opposed to the OntoLex "isSenseOf", for example, which requires the relation to express either synonymy or hierarchical structure). This implementation also has the advantage of confining this rather nonstandard use to a single location in our model, which is much preferable to, say, weaving nonstandard keywords (or worse, standard keywords in a nonstandard fashion) into the general fabric of our OntoLex and SKOS data.

In terms of raw OntoLex and SKOS encoding, this means that we need to distinguish between Lemmas, LaCs and Concepts in our RDF file. To do this, we add a shorthand type descriptor to each entry. As we've mentioned, Lemmas have "fl". Concepts have "ht" (short for "hugtak", the Wordweb's Icelandic term for "concept"), while LaCs have "FsH", Icelandic for "Fletta-sem-Hugtak" ("Lemma-as-Concept").

3.3 Element grouping

Before we move on to describing these elements in detail, one final note about their organization. Both OntoLex and SKOS allow the grouping of individual entities. This is an immensely powerful feature for one simple reason: It effectively allows us to assign a common property to any group of elements without having to encode that property with OntoLex or SKOS keywords. As a result, this kind of grouping might be used to extend the models' functionality in a myriad of nonstandard ways.

We have, however, intentionally avoided this kind of grouping, choosing to assign it only to Lemmas, Concepts and Lacs, which respectively belong to the OntoLex-Lexicon type "limeLexicon" and the SKOS-ConceptSet type "conceptScheme_ht" and "conceptScheme_fsh" collections. The reason we've avoided it is simply that these collections can very quickly become rather unwieldy. There is no way to encode any documentation for them in the Wordweb file, and calling on their elements through the SPARQL query language is somewhat more complex than calling on normal element properties. Future Wordweb versions might implement them in greater detail (one obvious option would be to group the word suggestions found inside [square brackets] into discrete sets, where each set might easily be assigned a corresponding Lexicon), but for now, we constrain ourselves to the three groupings listed above.

4. Relation types

4.1 Pairs

Aside from the "Lexical Entry - Lexical Sense - ontological element" tuples, the original Wordweb contains one fundamental semantic relation and three ancillary ones. It also contains a wealth of derived relations that build on these. In the original Wordweb, these derived relations are hardcoded as part of the Wordweb's web presentation. As a result, search functionality for these relations, based on each Lemma, is simple and straightforward - but is also absolutely fixed, which means the search parameters cannot be altered, and the search results cannot be exported for further examination.

In the new Wordweb, we have encoded only the fundamental semantic relations into the system's data. All other relations may be produced through queries written in the SPARQL query language. We will now explain how these semantic relations work, and after that explain the basis of SPARQL and provide queries for all the other types of relations.

Those fundamental relations are called parallel constructions, which we'll refer to hereafter with the shorthand "Pairs" (the Icelandic term employed by the Wordweb is "pör"). In the Wordweb, a Pair consists of two unordered Lemmas that have appeared somewhere in the source texts with the conjunction "and" (Icelandic "og") between them. For example, if we have the preexisting Lemmas "dog" ("hundur") and "cat" ("köttur"), and if somewhere in one of our sources we encounter the words "dog and cat" ("hundur og köttur", or "hundar og kettir" plural), then we add a Pair relation to the two Lemmas: "Dog" now forms a Pair with "cat", and vice versa.

(Without diving too deep into the semantics of synonymy and word relations, we should note that this kind of relation isn't of a single clear type. The Lemmas' pairing indicates they may be used in the same context but that they are not necessarily interchangeable: Their definitions may thus be related, and occasionally even shared, but may not be identical. There is also the implication of syntax, since a Pair relation indicates they have appeared together in a sentence. There is even a slight indication of shared categorization: The two Lemmas may not have the exact same meaning or syntactical purpose, but they clearly *go together* in a fashion. The nebulous nature of these relations allows us to look further than simple synonymy, and - as we'll see below - to build even more extensive and interesting relations on top of the Pairs.)

In terms of implementing Pairs in OntoLex and SKOS, recall from chapter 3.2 that while Lemmas are OntoLex entities, the Pair relation itself is difficult to represent solely with OntoLex functionality, seeing as how a prerequisite for that functionality is that it represent grammar and syntax - not semantics (at least not unless it can be avoided), and absolutely not categorization.

(To avoid any misunderstanding, it should be clear that OntoLex does support a number of Sense-to-Sense relations, and that these relations do cover a fair number of different types of semantic and terminological relations, including synonym and antonymy, printed variants, different emphasis on a specific subproperty of the Entry in question, etc. Doubtlessly these are very useful when required - but unfortunately they don't cover the semantic/syntactic/categorizational mix inherent in Pair relations.)

That's where SKOS comes in. We employ its "senseOf" keyword to encode Pair relations, and since this means we need entities that are SKOS concepts, rather than OntoLex Lexical Entries, we use the encapsulation layer to represent these relations. Let's look again at the encapsulation layer table, but this time indicate where the Pair relations would be encoded, if Lemma A and Lemma B were Pairs (see the sole entry in the table's middle column):

Lexical Entry		Lemma_A		Lemma_B		Lemma_C
		↓		↓		↓
Lexical Sense		✓		✓		✓
		↓		↓		↓
Lemma-as -Concept		LaC_A	↔ Pair	LaC_B		LaC_C
		↓				↓
SKOS concept		Concept_A		[No concept]		Concept_C

4.2 Other encoded relations

Let's now examine the rest of the Wordweb's relations that are encoded into the system itself.

Unlike Pairs, these are not used as building blocks for other relations. They are synonyms, antonyms, "links" (Icelandic term is "stiklur"), and compound words ("samsetningar"). In the original Wordweb, the first three are grouped together under the moniker "assessed relations" ("metin vensl").

The compound words associated with a particular (monolexical) Lemma are instances where the Lemma corresponds either to one of the two stems that form a compound word, or to the compound word itself. Note that these are not exhaustive, and only involve words divided into exactly two stems.

There is always at least one other Lemma present in this relation. For example, if we imagine that "sun" is a Lemma, and that one of the compound words related to it is "sunflower", either the Lemmas "sunflower" or "flower" (or both) would also be part of this relation. Likewise, if "sunflower" were the original Lemma, and if it contained a compound word relation, either the Lemmas "sun" or "flower" (or, again, both) would also be part of this relation.

The assessed relations have usually been derived from older thesaurii and other sources of that type. They form a relatively minor part of the Wordweb. Links are generally considered near-synonymy relations. Since synonyms and antonyms are one of the few relation types explicitly defined in OntoLex, we've opted to encode them using the OntoLex "senseRelation" functionality rather than implementing them in the encapsulation layer.

A senseRelation is a separate entity, rather than just a property. In the Wordweb, a senseRelation links two Lexical Senses. One of these senses is called the "source", the other the "target", and these determine the direction of the senseRelation. (Since assessed relations are bidirectional, we've simply created two senseRelation objects for each, with the source and target swapped.) The senseRelation has a special "vartrans:category" property which determines whether it's a synonym, antonym or "approximateSynonym", with the last of these being the OntoLex equivalent of the Wordweb's links.

Note that the senseRelations are practically unanchored; they can not easily be tracked through Lexical Entries and Senses. This is in accordance with the OntoLex design. To account for this, our Wordweb implementation assigns to each senseRelation two numbers, separated by a dash, corresponding to the source and the target. The Lexical Senses in question have those same numbers in their own titles. As a result, senseRelations can be located by first determining either the source Sense or target Sense, extracting that Sense's number, and looking for senseRelation-type objects containing that number in their titles.

4.3 Lemma subdivision and component relations

Lemma subdivision is a new and powerful feature of the new Wordweb. Simply put, polylexical Lemma are now divided into OntoLex "components".

Components may have many uses, depending on the system in question, and can theoretically hold as many or as few sub-elements as needed. In our case, we use them on polylexical Lemmas to represent every individual word that also exists, somewhere else in the Wordweb, as its own separate monolexical Lemma. The component links to that monolexical entry, and also stores information on that word's particular morphosyntactic form in the polylexical Lemma.

This new functionality allows for a much greater and more thorough level of interconnectedness in the Wordweb. In the original system, single words from polylexical Lemmas could be found only through a separate search for that particular word. In our implementation, the Lemmas can be traversed back and forth like a web.

(As per our earlier discussion of OntoLex's capabilities for element subdivision, it should be noted that OntoLex components may also be used to represent syntactic roles, both for single and multiple words. This kind of functionality was not on offer in the original Wordweb and was thus not given priority during the development process. However, the new format would absolutely support this kind of information, for example if it were added through a language parser.)

When a multiword Lemma is subdivided, we create these so-called components to represent each sub-part. These components point to their single-word counterparts and also include any morphological information specific to that particular multi-word lemma. Additionally, we create a series of "rdf counters" to represent the order in which the components should be read (the components, by themselves, are unordered). In the Wordweb, each such component represents a single word.

Let's imagine we have the multiword Lemma "only fools and horses", and that the Wordweb already contains the Lemmas "fool" and "horse". When we parse "only fools and horses", we create two components (plus counters). The first is called "only fools and horses (01c)" and includes two pieces of information: A pointer to the Lemma "fool"; and morphosyntactic information indicating that its form in "only fools and horses" is plural. The second is called "only fools and horses (02c)" (the "c" stands for "component") and points to the Lemma "horse", again with morphosyntactic information indicating it should be pluralized.

5. Using the Wordweb: SPARQL

Lastly, we will go over how to extricate data from the Wordweb, both in terms of the relations available in its original form on the Wordweb web site, and in terms of relations that are fully user-defined. To do any and all of this, we employ SPARQL.

SPARQL is an RDF query language that, thanks to our choice of models, is directly applicable to the data encoded in the Wordweb RDF file. Its queries are usually in the form:

```
SELECT * WHERE {
   ?s ?p ?o .
}
```

The letters "s", "p" and "o" stand for variables that represent subject-predicate-object relationships, often in a kind of "entity s has the relationship of type p with entity o" pattern. (You don't always have to use all three.)

5.1 Getting SPARQL to run

To run SPARQL queries on the Wordweb's data, we need two things: A query program that can parse our SPARQL queries, and a data storage program that supports loading very large files in "RDF triplet" or "triple store" format.

The point about very large files is important in this context. The full Wordweb RDF file is approximately 2 Gigabytes in size. On most computers, any attempt to load it all at once into memory is likely to fail. What we need to do instead is run a program that parses the Wordweb file, then creates and saves to disk a readable database of its contents. Any SPARQL queries should then be run against this database. Query response time will naturally be slower, since we'll be reading from disk rather than directly from memory, but with a file of this size there simply is no alternative.

(Incidentally, please keep in mind that even though the database is not all stored in memory, the results of your queries will be. Queries that return too many results may fail to complete, and instead produce only "out of memory" errors. If this is happening, you may have some luck with increasing the amount of memory available to the query program, or possibly running the query in so-called "chunks", but most likely you will simply have to narrow down either the parameters of your search or place a hard limit on the amount of results it produces.)

There are a number of options for local triple store programs. In the examples below we've gone with Apache Jena TDB, which is free, Linux-compatible, and has a comparatively low barrier to entry. Other triple-store linguistic projects are known to have employed Parliament, RDF4J, GraphDB, and Virtuoso.

Setup instructions for Jena TDB follow. Pleae note that the most current versions of both software and documentation take precedence over the following instructions, particularly in cases where there is a clear disparity. In addition, please note that any issues with or questions about the following instructions should be directed to the Apache Jena support staff. **Neither the author of the Wordweb Handbook, nor The Árni Magnússon Institute for Icelandic Studies, can assume any responsibility nor liability for any issues that may arise, including software incompatibility or data loss.**

Navigate to the Apache Jena homepage at https://jena.apache.org/index.html and download the latest version of Jena for your particular operating system. We only want to download Apache Jena, and not Apache Jena Fuseki, as the TDB module comes bundled with the former. Unpack the downloaded file into a folder of your choice, then open that folder in a terminal window.

What we now want to do is create a persistent (i.e. saved to disk) database, *not* one that gets loaded directly into memory. If you haven't done so already, download and unpack a copy of the wordweb.rdf folder from the Clarin-IS web site (see https://repository.clarin.is/repository/xmlui/handle/20.500.12537/100) and move that copy into a particular subfolder of where you'd unpacked Apache Jena: The /bin/ subfolder if you're using Linux, or the /bat/ subfolder if you're using Windows.

Before proceeding, create a separate folder which will house your TDB-format database. Please note that this folder will require several Gigabytes of storage. This folder may be anywhere, though note that you may have to repeatedly reference it by its full path during your SPARQL query execution later on.

Windows setup instructions

You'll have unpacked wordweb.rdf into the /bat/ subfolder of your Apache Jena installation. Open a command prompt, with administrator rights, into that subfolder.

If any of the following instructions fail, particularly with the message "JENA_HOME not set", there are several steps you can take for problem analysis and resolution. A good place to start is https://jena.apache.org/documentation/tools/ , where "Setting up your environment" gives an indication of how you need to set your environment variables, while "Common issues with running the tools" may help with resolving certain problems (including possibly downloading a

more recent version of the batch tools directly from Github). More specifically - **and please note that we cannot take responsibility for anything that might go wrong in this process** - you can try going to Control Panel -> System -> Advanced system settings -> "Advanced" tab -> Environment variables; where, under System variables, you edit "Path" to add a new line that reads (without quotes) "%JENA_HOME%/bat", and then add a brand new system variable where the variable name is "JENA_HOME" (without quotes) and the variable value is the full system path to your Apache Jena folder.

We strongly recommend that you reboot your computer after performing any of the above steps. If your JENA_HOME variable is set, you should be able to open a command prompt as administrator, execute "cd %JENA_HOME%", and be automatically transported to the /bat subfolder.

Once you're all set up, navigate your administrator-level command prompt to the /bat subfolder and run the following command, where [TDBPATH] is the full system path to the TDB folder you created. It will create a permanent database containing the Wordweb's RDF-encoded data (and may take a bit of time to run):

```
tdbloader.bat --loc=c:\Users\Notandi\Desktop\Wordweb\TDB wordweb.rdf
```

Once you've created the database, you can run SPARQL queries on it either directly or through a text file. If you opt for the latter, you can then run queries with the following command, where [TDBPATH] is, again, the full path to your newly-created TDB database, and [SPARQLFILE] is the filename of your SPARQL query text file (which should be saved to the directory from which you're running this command):

```
tdbquery.bat --loc=[TDBPATH] --query [SPARQLFILE]
```

If you wish you save the output to file, simply append " > [OUTPUTFILE]" to the command, where "[OUTPUTFILE]" is whatever name you chose for the file in which to save the query's output.

Linux setup instructions

You'll have unpacked wordweb.rdf into the /bin/ subfolder of your Apache Jena installation. In that same /bin/ subfolder, you should see scripts with names that include "tbd2.tbdloader" and "tbd2.tbdquery". (TBD2 works on Linux systems, whereas on Windows we have to run regular TDB instead.)

Let's say that the name of the Wordweb file is "wordweb.rdf" and that the full path of your TDB2 directory is [TDBPATH]. Open a terminal window in the /bin/ folder and run the following command (this may take a while):

```
./tdb2.tdbloader --loc=[TDBPATH] wordweb.rdf
```

You will only have to run this TDBloader script once. Now let's assume that you've saved your SPARQL query to a text file, [SPARQLFILE], located in that same /bin/ subdirectory of your Apache Jena software. In order to run that SPARQL query, open a terminal window inside /bin and execute the following script:

```
./tdb2.tdbquery --loc=[TDBPATH] --query [SPARQLFILE]
```

This script (tdb2.tdbquery) reads your SPARQL query and runs it against the database you've created. If you wish to save the output to a file, say [OUTPUTFILE], simply add " > [OUTPUTFILE]" to the end of your query command, where "[OUTPUTFILE]" is whatever name you chose for the file in which to save the query's output.

5.2 Core SPARQL functionality

SPARQL keywords in the Wordweb

SPARQL queries can be made to search either for raw text, or for keywords employed in the data collection in question. Some of these keywords are standalone, while others employ extra keywords to define particular sub-properties.

We do encourage any interested reader to familiarize themselves not only with the OntoLex and SKOS models, but also with the raw contents of the Wordweb RDF file. However, as a convenient index, here is a sorted list of keywords employed in that file. In the following subchapters, we will provide examples that show how keywords and properties may be used in SPARQL queries. Keep in mind that the entities detailed below are described in detail in chapter 3 of this document.

Lemmas have three categorizations: Basic types, part-of-speech categories, and term types. A Lemma will always be of one basic type, and may have a maximum of one term type, but can have zero-to-many entries for part-of-speech.

Types are red:type and can be either ontolex:LexicalEntry (the default type), ontolex:Word, ontolex:MultiWordExpression, or ontolex:Affix.

Part-of-speech categories are lexinfo:partOfSpeech and may be lexinfo:exclamativeDeterminer, lexinfo:conjunction, lexinfo:preposition, lexinfo:numeral and lexinfo:properNoun.

Term types are lexinfo:termtype and may be lexinfo:setPhrase, lexinfo:phraseologicalUnit and lexinfo:idiom.

Multi-word Lemmas, as you may recall, are subdivided into components when possible, and linked to their single-word counterparts. A multi-word Lemma subdivided in this way will contain two lines for each subpart. The first is decomp:constituent, which links to the subcomponent; the second is rdf:_X, where "X" is a numer representing the order that component has in the full list of components for this multi-word Lemma. Both of these lines include a link to the component in question.

The component linked by decomp:constituent will have a name that ends in two digits and a "c", the digits representing that same order in the full list. It uses the property decomp:correspondsTo to link directly to the Wordweb single-word Lemma entry it corresponds to.

A *Lexical Sense* will have a unique ID in its URI. It will always contain a ontolex:isSenseOf which links to its Lemma, and a ontolex:reference which links to its LaC. In addition, if the meaning attached to this Sense was either explicitly written out in the original Wordweb, or if it was derived from square brackets (see earlier chapters), the Sense will also contain a ontolex:usage line that contains the definition itself.

Certain Lexical Senses also have particular kinds of relations directly with other Senses, in the form of antonyms, synonyms and approximate synonyms. This Sense-to-Sense relation has its own entry in the Wordweb, and will be named "senseRelation(sensenumber1)_(sensenumber2)" (there will likely also be another sense relation entry with the two numbers switched). This kind of relation is considered as being directed from the former number and to the second, and therefore will contain a vartrans:source entry pointing to the first Sense, and a vartrans:target pointing to the second Sense. In addition, it will contain a vartrans:category property which may be lexinfo:synonym, lexinfo:antonym or lexinfo:approximateSynonym.

Forms are encoded both within Lemmas and as discrete entities. A Lemma will usually contain at least an ontolex:canonicalForm property that links to its Form counterpart. If the Lemma has multiple morphosyntactic variants, it will also contain a sizeable number of ontolex:otherForm properties, each of which links to a specific Form counterpart.

Recall that the Form entry for a particular Lemma will contain only two properties: the text of that lemma, and a link to a standardized Form of that particular type. This helps us avoid reduplication of Form information. The text has the property ontolex:writtenRep, while the link to

the standardized Form use the same ontolex:canonicalForm keyword as we saw in the Lemma itself.

As an example, the Fletta "affaradrjúgur_(lo)" includes, among many others, a link to the Form "LO-FSB-KK-ÞFET_affaradrjúgur_(lo)". That particular form has two properties: The written form of that morphosyntactic variant, which is "affaradrjúgan"; and a link to the standardized "LO-FSB-KK-ÞFET" form which represents every single Lemma in that particular morphosyntactic variant.

That standardized Form will generally contain a wealth of grammatical information, as the reader might have inferred from the listing in the Forms section of chapter 3. In keyword terms, they are as follows:

- lexinfo:partOfSpeech may be lexinfo:noun, lexinfo:adjective, lexinfo:verb, lexinfo:adverb or lexinfo:pronoun
- lexinfo:degree and lexinfo:definiteness sometimes go together. By itself, lexinfo:degree may be lexinfo:positive, lexinfo:comparative, or lexinfo:superlative. With both lexinfo:degree and lexinfo:definiteness, they may respectively have lexinfo:positive and lexinfo:indefinite; lexinfo:positive and lexinfo:definite; lexinfo:comparative and lexinfo:indefinite; or lexinfo:comparative and lexinfo:definite
- lexinfo:verbFormMood may be lexinfo:infinitive, lexinfo:indicative, lexinfo:subjunctive or lexinfo:imperative. Additionally, lexinfo:verbFormMood may be paired with lexinfo:tense for the respective values of lexinfo:participle and lexinfo:present; or lexinfo:participle and lexinfo:past
- lexinfo:case may be lexinfo:nominativeCase, lexinfo:accusativeCase, lexinfo:dativeCase or lexinfo:genitiveCase
- lexinfo:number may be lexinfo:singular or lexinfo:plural
- lexinfo:voice may be lexinfo:activeVoice or lexinfo:middleVoice
- When applied to the definite article of nouns, lexinfo:definiteness may be lexinfo:definite or lexinfo:indefinite
- When applied to the declension strength of verbs, lexinfo:definiteness also takes on either lexinfo:definite or lexinfo:indefinite
- lexinfo:gender may be lexinfo:neuter, lexinfo:masculine or lexinfo:feminine
- lexinfo:tense may be lexinfo:present or lexinfo:past

Basic SPARQL queries

Each SPARQL query usually begins with a long list of prefixes that provide a kind of shorthand for the multitude of standardized terms employed in these queries. To maintain readability and clarity, we will only list these once, but please note that they should be included with every SPARQL query.

A very simple SPARQL query would thus be as follows:

```
PREFIX cc: <http://creativecommons.org/ns#>
PREFIX grddl: <http://www.w3.org/2003/g/data-view#>
PREFIX void: <http://rdfs.org/ns/void#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dcam: <http://purl.org/dc/dcam/>
PREFIX vartrans: <http://www.w3.org/ns/lemon/vartrans#>
PREFIX owl2xml: <http://www.w3.org/2006/12/owl2-xml#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX synsem: <http://www.w3.org/ns/lemon/synsem#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX lexinfo: <http://www.lexinfo.net/ontology/2.0/lexinfo#>
PREFIX dcr: <http://www.isocat.org/ns/dcr.rdf#>
PREFIX lemon: <http://lemon-model.net/lemon#>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ontolex: <http://www.w3.org/ns/lemon/ontolex#>
PREFIX vann: <http://purl.org/vocab/vann/>
PREFIX semiotics: <http://www.ontologydesignpatterns.org/cp/owl/semiotics.owl#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX skosxl: <http://www.w3.org/2008/05/skos-xl#>
PREFIX decomp: <http://www.w3.org/ns/lemon/decomp#>
PREFIX lime: <http://www.w3.org/ns/lemon/lime#>
PREFIX core: <http://www.w3.org/2004/02/skos/core#>
SELECT *
WHERE {
        ?a ?b ?c
}
```

Note that this query will produce a large number of results. If you ever want to limit your results - such as when verifying whether a newly constructed query will run at all - we suggest you place a hard limit on the result count, like so (don't forget to include all the PREFIX lines, too!):

```
SELECT *
WHERE {
        ?a ?b ?c
} LIMIT 500
```

Keep in mind that some queries depend on the language in which the data is encoded. This means that you may have to affix "@is" to certain queries. A simple search for the standard form

of the Lemma "vindur" (Icelandic for "wind") would thus be as follows (note that every WHERE condition except the last must end with a full stop):

```
SELECT ?lemma
WHERE {
        ?lemma ontolex:canonicalForm ?lemma_form.
        ?lemma_form ontolex:writtenRep "vindur"@is
}
```

The search produces the single entry <vindur_fl_(no)>. More specifically, it finds all ontolex Forms whose written representation is "vindur", and then finds all Lemmas with a corresponding canonical Form. Note that we cannot find the Lemma directly from a particular search condition; instead, we have to trace our way through intermediate results by first finding a Form and then finding the Lemma through that. This is going to be a familiar pattern in most searches. A Lemma isn't just a single entity (and corresponding single entry) of names and properties, but rather a nucleus surrounded by several other discrete entries, each of which represents a particular facet of that Lemma.

If we now want to use this particular Lemma in order to find some other property, we can use that single entry result for other queries. The following search produces an entry corresponding to the canonical Form for the Lemma in question:

```
SELECT ?lemma_form
WHERE {
        <vindur_fl_(no)> ontolex:canonicalForm ?lemma_form.
}
```

For a good example of how we need to weave our way through multiple layers, let's look at how to find out whether a given Lemma belongs to any particular Concept. (Recall that in the Wordweb, the designations for Lemmas, LaCs and Concepts respectively are "fl", "fsh" and "ht".) An added complication is that a single Lemma may have more than one Lexical Sense - i.e. more than one definition. We'll want to ensure that we also bring up the definition for each of these Lexical Senses; these may be found as rdfs:label properties under each corresponding LaC. So, in brief: From Lemma we find Lexical Senses. From each Lexical Sense we find a single LaC along with the LaC's label. From each LaC we then find a Concept, if one exists. A SPARQL query for all of this will then be as follows:

```
SELECT ?label ?concept
WHERE {
        <vindur_fl_(no)> ontolex:sense ?sense.
        ?sense ontolex:reference ?lac.
        ?lac rdfs:label ?label.
        ?lac skos:broader ?concept.
```

```
}
```

Of course, when we start searching, we're not likely to know the precise entry for any given Lemma in the Wordweb. Instead, we'll probably be starting with a word, and possibly not even the canonical form of that particular word. Let's say that we wanted, again, to find any concepts to which "vindur" belongs, but that now the only information we have is that particular word in the genitive case, plural, with the definite article: "vindanna". This might be the case if we've picked the word out of a written source, particularly if we're analyzing those words without direct human oversight. In fact, we might not even know the specifics of the word's particular morphological form - its case, gender, and so on - and much less be able to search for them.

A quick side note before we can run the search. If we've picked the word right out of a source, we may not know whether it represents the canonical form of that particular Lemma, or some other form. We don't want to search solely for one and exlude the other. In order to get both options, we need to use the UNION keyword, which (in very simplified terms) combines the outcome of two comparable search parameters. A search for all the forms of vindur, given its Wordweb entry, would then be:

```
SELECT ?form
WHERE {
        {
        <vindur_fl_(no)> ontolex:otherForm ?form }
        UNION
        {
        <vindur_fl_(no)> ontolex:canonicalForm ?form}
}
```

Going back to our word search, let's start with the most basic search option - which we can then, naturally, expect to provide the least specific results - where all we have is the word form and nothing else. If we only look to Lemmas for now, this gives us the following (note again the full stop right after the UNION condition):

```
SELECT ?lemma
WHERE {
        ?form ontolex:writtenRep "vindanna"@is.
        {?lemma ontolex:otherForm ?form }
        UNION
        {?lemma ontolex:canonicalForm ?form } .
}
```

This produces two results: "vinda_fl_(no)" and "vindur_fl_(no)". The former is a female-gendered noun which translates to "winch", while the latter is a male-gendered noun that translates to "wind", the word we were looking for. Disambiguating these terms can be a complex task

requiring either a human reader or specially designed software. We'll leave out the UNION keyword from now on, so that we can better focus on the actual purpose of each search, but keep in mind that you might have to include it in your search in order to capture all possible variables.

Let's say that we now start with two additional pieces of information: Aside from our word form being "vindanna", we know that it's a noun, and that it's male-gendered. (We'll be using Ontolex keywords for this particular search, but note that it's perfectly possible to search directly for specific form versions in the Wordweb by using grammatical category designations. For example, nouns have the designation "NO", and male-gendered ones have the additional designation "KK", so their forms will always begin with "form_NO-KK-*". For extensive lists of standardized designations, refer to the online documentation for The Database of Icelandic Morphology.) These two properties are encoded as lexinfo:partOfSpeech being lexinfo:noun, and as lexinfo:gender being lexinfo:masculine.

What's important to keep in mind is that every Lemma has a unique set of forms associated with each of its morphological variant, and that each of these unique forms *derives from a template form for that type of morphological variance*. It's through these template forms that we can search for properties such as gender and grammatical category. (The reason for this implementation is so that we have to don't constantly duplicate these properties for every single Lemma in the Wordweb, but rather just reference them each time using ontolex:canonicalForm).

The resulting search then becomes:

```
SELECT ?lemma
WHERE {
        ?lemma ontolex:otherForm ?form.
        ?form ontolex:writtenRep "vindanna"@is.
        ?form ontolex:canonicalForm ?templateForm.
        ?templateForm lexinfo:partOfSpeech lexinfo:noun.
        ?templateForm lexinfo:gender lexinfo:masculine.
}
```

With the added morphological constraints, we now get only one result, for the correct "vindur" masculine-gendered noun.

5.3 SPARQL queries for Wordweb-specific legacy relations

We now move on to the derived relations; that is, SPARQL queries that reproduce those relations which were hardcoded into the Wordweb's original form on the corresponding Wordweb website. Please note that these relations may not necessarily be the ones most useful to the average user; we encourate you to read section 5.4 for a list that may be more applicable.

Again, it should be noted that most of these relations are not a fixed part of the new Wordweb's design. Rather, they are simply convenient ways in which to recall and handle the Wordweb's data. The only relations that are fixed are Lemmas and Concepts, and Lemma Pairings.

Since these relations are tied to web site functionality, most of them rely on the user already having searched for and selected a particular Lemma. While the original Lemma's existence may not be explicitly mentioned, it's usually implied (for example, in the relation called "Lemmas sharing the same Concept", "the same" clearly implies "as the Lemma the user has already selected). When we diagram these relations, we will use the term "Lemma" to indicate the original entity whose relations are being inspected, enumerate other terms as required to distinguish their hierarchies, and underline those entities that would appear to the user in a web site search through the original Wordweb. (It might help to think of these entities as a list of results from a search query. Note that the user would not be presented with the full hierarchy of entities; only the underlined ones.)

Concept-based relations: Lemmas sharing a Concept ("Flettur undir hugtakinu")

The original Wordweb offers three types of relations that specifically involve Concepts. This effectively means that unlike most other relations, the user must not only select a particular Lemma, but also a particular Concept to which that Lemma belongs. Lemmas that do not belong to any Concepts are not considered for these types of relations.

The first is Lemmas that share the same Concept. This relation might be modelled as:

- Lemma
    - Concept
        - Lemma_1
        - Lemma_2
        - ...

Once a Concept has been selected, we can use the following pattern for a SPARQL query to list its LaCs:

```
SELECT * WHERE {
   [CONCEPT] skos:narrower ?o .
}
```

As an example, if we've chosen the concept "þoka" (for "fog"), our query would be:

```
SELECT ?lac WHERE {
   <þoka_ht_(52874)> skos:narrower ?lac .
}
```

This produces a myriad of LaC results, ranging from "álfalæða" to "þokusamur".

(In fact, it produces the Wordweb links to these LaCs. We will omit, from this and most future examples, the code required to specifically call forth Lemmas from LaCs, or written representations from each. To add this functionality, please see the preceding chapter.)

Concept-based relations: Pair-related Lemmas ("Venslaðar flettur í gegnum pör")

These are similar to the previous relation, except instead of looking for Lemmas that directly belong to the same Concept, we're looking only for Lemmas that form Pairs with those Lemmas:

- Lemma
    - Concept
        - Lemma_A
            - "and" LemmaA_1
            - "and" LemmaA_2
            - ...
        - LemmaB
            - "and" LemmaB_1
            - "and" LemmaB_2
            - ...
        - ...

Note that the original Lemma is effectively meaningless, other than as a way to trace our way to a particular Concept. In the original Wordweb, Concepts could only be found through a Lemma-based search. In this version, Lemmas are no longer strictly necessary.

Once the user has decided on a given Concept, they would then call on all LaCs belonging to that Concept, and then for each one list every LaC with which it has a Pair relation.

As an example, say we've chosen "hestur" ("horse") as our Concept. The search then becomes:

```
SELECT ?firstlac ?secondlac
WHERE {
        ?firstlac skos:broader <hestur_ht_(17255)> .
```

```
        ?secondlac skos:broader <hestur_ht_(17255)> .
        ?firstlac skos:related ?secondlac .
}
```

Concept-based relations: Related Concepts ("Tengd Hugtök")

This is the last of the three relations directly involving Concepts. By this point, the relation may start to look a little labyrinthine, but it has the same basic structure as the one previously described.

Recall that we selected a Lemma, then its Concept, got a list of all the Lemmas under that Concept, and for each of those Lemmas inspected all its Paired Lemmas.

Now, for each one of those Paired Lemmas, we return not the Lemma itself, but all the Concepts (if any) that it belongs to:

- Lemma
    - Concept
        - Lemma_A
            - "and" LemmaA_1
                - ConceptA1_a
                - ConceptA1_b
                - ...
            - "and" LemmaA_2
                - ConceptA2_a
            - ...
        - LemmaB
            - "and" LemmaB_1
                - ConceptB1_a
            - "and" LemmaB_2
                - ConceptB2_a
                - ConceptB2_b
            - ...
        - ...

The relations that link the original Concept to the eventual Concept can thus be laid out as:

Concept ↔ LaC ↔ "skos:related" ↔ LaC ↔ Concept

If we again select "hestur" as the original Concept, the SPARQL query could be:

```
SELECT DISTINCT ?Concept WHERE {
        ?LaC1 skos:broader <hestur_ht_(17255)> .
        ?LaC1 skos:related ?LaC2 .
        ?LaC2 skos:broader ?Concept .
   FILTER (?Concept != <hestur_ht_(17255)>) .
}
```

Pairs ("Parasambönd" or "Pör")

As previously mentioned, this relation is the cornerstone of the Wordweb. Luckily, it is easy to represent and query. The model would simply be:

- Lemma
    - "and" Lemma1
    - "and" Lemma2
    - ...

The corresponding SPARQL query for the Lemma "hestur" then becomes:

```
SELECT DISTINCT ?PairLemma WHERE {
        <hestur_fl_(no)> ontolex:sense ?Sense .
        ?Sense ontolex:reference ?LaC .
        ?LaC skos:related ?PairLaC .
        ?PairLaC ontolex:isReferenceOf ?PairSense .
        ?PairSense ontolex:isSenseOf ?PairLemma .
}
```

A few aspects of this query bear mention. Firstly, we use "DISTINCT" to ensure we don't have a long list of duplicate entries. Secondly, this query fully implements Lemma results, rather than just LaCs; its code may be re-used in other query examples as needed. Lastly, note that we don't use any SPARQL "FILTER" options, which means that "A to A" relations are possible. This reflects the Wordweb's design, where reflexive relations are allowed. If the user wants to avoid reflexivity, additional query restrictions should be used.

Relationals ("Skyldheiti")

Two Lemmas are considered Relationals if both are Pairs with the same, third Lemma.

In the original Wordweb, lists of Relationals may be ordered in two ways: By the raw count of how many third Lemmas they're Paired with, or the ratio of how strongly those third Lemmas are Paired with them rather than other, unrelated Lemmas. Representing these two orderings as a bulletpoint model would likely be rather confusing, so we'll employ set notation instead.

The raw count of common partners ("fjöldi sameiginlegra félaga") is likely the easier of the two to represent and explain. If A is the Lemma we originally selected, C is a Lemma with which A has a Relational relationship, and the set Bx = {B1, B2, B3, ...} is the set of all Lemmas that have a Pair relationship with both A and C, then the raw count of common partners is the number of Lemmas in B, or |B|

Let's now look at the ratio of relational pairs ("hlutfall af parasamböndum skyldheitis"). As before, A is the Lemma we originally selected, C is a Lemma with which A has a Relational relationship, and the set Bx = {B1, B2, B3, ...} is the set of all Lemmas that have a Pair relationship with both A and C. Let's now define the set Dx = {D1, D2, D3, ...} as the set of all Lemmas with which C is a Pair. (Note that this implies Bx is a subset of Dx.) The ratio of the relational pairs that C has with A is then |B| / |D|

Let's show how we could produce these results through SPARQL. We'll do so in three queries. (As with other query languages, it is of course perfectly possible to merge multiple queries into a single one, but at the cost of readability.)

Let's first find the set of Bx. We'll assume the original Lemma was "hestur" again, and start our search from its corresponding LaC:

```
SELECT ?B WHERE {
        <hestur_fsh_(17255)> skos:related ?C .
        ?B skos:related ?C .
        ?B skos:related <hestur_fsh_(17255)> .
}
```

This is sufficient if we're only looking for the base list of common partners. Moving on to the ratio of relational pairs, let's say that these results - the entire set Bx - turned out to simply be {"asni"} ("donkey"). For every element in Bx, we'll need to run the following, which will give us its Dx:

```
SELECT ?D WHERE {
        <asni_fsh_(1776)> skos:related ?D .
}
```

Let's say that for this particular Bx, the query returned the Dx {"hestur", "múldýr"} (note that we could use "COUNT" to return numerical results if we're not interested in the Lemmas

themselves but only the ratios). If we now want to determine which elements in Dx relate back to A ("hestur") through Pair relations, we can run the query:

```
SELECT ?D WHERE {
        <asni_fsh_(1776)> skos:related ?D .
        ?D skos:related <hestur_fsh_(17255)> .
}
```

The New Neighbors ("Grannheiti")

Neighboring Lemmas are a unique case in that they have effectively been superseded by a certain kind of different-but-similar functionality in the new Wordweb.

In the original Wordweb version, Neighbor relations depended on a separate storage of idioms. Two Lemmas were considered to be Neighbors if they were interchangeable inside one or more of these idioms. As an example, if there existed the idioms "grown man" and "grown woman", then the Lemmas "man" and "woman" (if they existed in the Wordweb) would be considered Neighbors, bound by the "grown [*]" idiom pattern.

A fair amount of these idioms were not Lemmas themselves, nor were they intended to be. For this and various other reasons, they were not valid candidates for parsing and coding into the new Wordweb. Keep in mind that the only entities that should be stored in our RDF file are either Lemmas (or Concepts) or particular properties pertaining to them.

However, those idioms that happened to also be multiword Lemmas were parsed normally. Moreover, as you may recall from previous chapters, these were automatically divided into components, so long as those components also corresponded to (single-word) Lemmas. As a result, we can produce a new type of Neighbor relation: One that isn't based on a separate store of out-of-system idioms, but on the Lemmas themselves.

There are various ways in which the new Neighbor relation may be approached. If we want a wealth of results, we could simply choose an idiom that has been subdivided into components, and query for the synonyms of any of those components. This would give us examples with very similar meanings, albeit not necessarily ones where every entry would be a perfectly valid candidate for that idiom.

A more narrow and far more precise approach would be to compare the texts of two idioms and look for the odd word out. Admittedly, this would likely need to involve some additional text parsing of query results. A first step would be to run a SPARQL query for all idioms, something along the lines of:

```
SELECT ?multiwordexpression
WHERE {
	?multiwordexpression lexinfo:termType lexinfo:idiom .
}
ORDER BY ?multiwordexpression
```

If the reader were to run this (alphabetically ordered) query on the Wordweb, they would quickly see that there are many multiword Lemma idioms that differ only slightly in presentation and are perfectly worthy candidates for Neighbor relations.

At this stage we would likely employ some kind of parsing code in order to compare these Lemmas and see if any of them fulfil the Neighbor pattern. It is possible, however, to do a little bit of exploratory legwork in SPARQL beforehand, by which single-word Lemmas may be extracted from the multiword ones:

```
SELECT ?multiwordexpression ?singlelemma
WHERE {
	?multiwordexpression lexinfo:termType lexinfo:idiom .
	?multiwordexpression decomp:constituent ?decomp .
	?decomp decomp:correspondsTo ?singlelemma .
}
ORDER BY ?multiwordexpression
```

In SPARQL we cannot trivially inspect the remainder of a multiword Lemma minus a particular single-word entry. A text parser, however - possibly one which supports so-called regular expressions - could be used to identify idioms of the form "A B [Cx]", where "Cx" could be any number of single-word Lemmas that appear in the Wordweb within this particular pattern: "A B C1", "A B C2", etc.

5.4 Other SPARQL queries

Of course, SPARQL may be used to retrieve an effectively infinite combination of entities and properties from the Wordweb, far beyond those offered by the relations encoded in the original website form.

What follows is a brief list of queries that the reader may find useful. This list is expected to grow over time, as interesting new queries come to light from the Wordweb's userbase.

As with earlier examples, please note that all queries need to start with the standard list of prefixes described under "Basic SPARQL queries" in chapter 5.2. We will reproduce that list in the first example, but leave it out thereafter. Note also that queries will generally return full

entities ("<þoka_ht_(52874)>"), not just entity names ("þoka"). In the first query, we include search parameters to explicitly produce the names of the entities in question. This same code will be left out of other examples for brevity.

To find all synonyms, and order them alphabetically:

```
PREFIX cc: <http://creativecommons.org/ns#>
PREFIX grddl: <http://www.w3.org/2003/g/data-view#>
PREFIX void: <http://rdfs.org/ns/void#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dcam: <http://purl.org/dc/dcam/>
PREFIX vartrans: <http://www.w3.org/ns/lemon/vartrans#>
PREFIX owl2xml: <http://www.w3.org/2006/12/owl2-xml#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX synsem: <http://www.w3.org/ns/lemon/synsem#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX lexinfo: <http://www.lexinfo.net/ontology/2.0/lexinfo#>
PREFIX dcr: <http://www.isocat.org/ns/dcr.rdf#>
PREFIX lemon: <http://lemon-model.net/lemon#>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ontolex: <http://www.w3.org/ns/lemon/ontolex#>
PREFIX vann: <http://purl.org/vocab/vann/>
PREFIX semiotics: <http://www.ontologydesignpatterns.org/cp/owl/semiotics.owl#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX skosxl: <http://www.w3.org/2008/05/skos-xl#>
PREFIX decomp: <http://www.w3.org/ns/lemon/decomp#>
PREFIX lime: <http://www.w3.org/ns/lemon/lime#>
PREFIX core: <http://www.w3.org/2004/02/skos/core#>
SELECT ?synonym1 ?synonym2
WHERE {
        ?senserel vartrans:category lexinfo:synonym .
        ?senserel vartrans:source ?sense1 .
        ?senserel vartrans:target ?sense2 .
        ?sense1 ontolex:isSenseOf ?Lemma1 .
        ?sense2 ontolex:isSenseOf ?Lemma2 .
        ?Lemma1 ontolex:canonicalForm ?Form1 .
        ?Lemma2 ontolex:canonicalForm ?Form2 .
        ?Form1 ontolex:writtenRep ?synonym1 .
        ?Form2 ontolex:writtenRep ?synonym2 .
}
```

```
ORDER BY ?synonym1 ?synonym2
```

The same query again, except now we take a shortcut by finding the entity names directly through LaCs, rather than tracing them through Lexical Entries. This makes for shorter queries and a slightly faster search, which may come in handy if used as part of a larger, more complicated query.

```
SELECT ?synonym1 ?synonym2
WHERE {
        ?senserel vartrans:category lexinfo:synonym .
        ?senserel vartrans:source ?sense1 .
        ?senserel vartrans:target ?sense2 .
        ?sense1 ontolex:reference ?LaC1 .
        ?sense2 ontolex:reference ?LaC2 .
        ?LaC1 rdfs:label ?synonym1 .
        ?LaC2 rdfs:label ?synonym2 .
}
ORDER BY ?synonym1 ?synonym2
```

To find all words where the written form - disregarding any differences in grammatical categorization - has more than one explicitly mentioned definition (This would for example bundle all versions of "jump" into a single entry, irrespective of whether the definitions pertain to the verb or noun):

```
SELECT ?term ?usage1
WHERE {
                {?Lemma rdf:type ontolex:Word}
        UNION
                {?Lemma rdf:type ontolex:MultiWordExpression} .
        ?Lemma ontolex:sense ?Sense1 .
        ?Sense1 ontolex:usage ?usage1 .
        ?Lemma ontolex:sense ?Sense2 .
        ?Sense1 ontolex:usage ?usage2 .
        ?Lemma ontolex:canonicalForm ?Form .
        ?Form ontolex:writtenRep ?term .
FILTER (?Sense1 = ?Sense2 ).
}
ORDER BY ?term ?usage
```

Instead of searching for similar words through Pair relations, we may want to find words whose meaning stays related despite their use in varying contexts. This searches for all words that belong to at least two Concepts. Note that for brevity, we now stop using the extra query code specifically to produce the written forms of the words - instead, we settle for either Fletta or LaC entries. (Warning: This is a heavy query that may take some time to run. Make sure to either set a proper LIMIT on results, or add further conditions to narrow down the pool of entities):

```
SELECT ?lac1 ?lac2 ?concept1 ?concept2
WHERE {
        ?lac1 skos:broader ?concept1 .
        ?lac1 skos:broader ?concept2 .
        ?lac2 skos:broader ?concept1 .
        ?lac2 skos:broader ?concept2 .
}
ORDER BY ?lac1 ?lac2
LIMIT 100
```

Since the Concept part of the Wordweb was created by its administrators, and is thus inescapably more subjective than the Pair relations - which are derived directly from source texts - we may want to add to this query some conditions on the connectivity between the two words. Here, we've added the condition that they share the same Pair partner:

```
SELECT ?lac1 ?lac2 ?concept1 ?concept2
WHERE {
        ?lac1 skos:broader ?concept1 .
        ?lac1 skos:broader ?concept2 .
        ?lac2 skos:broader ?concept1 .
        ?lac2 skos:broader ?concept2 .
        ?lac1 skos:related ?lac3 .
        ?lac2 skos:related ?lac3 .
}
ORDER BY ?lac1 ?lac2
LIMIT 100
```

You may simply want to check how equally Concepts manage to cover Lemmas. One way of looking into this is listing the Concepts by their Lemma counts. Recall that every Lemma has a corresponding LaC, and that LaCs and Concepts are related through skos:broader:

```
SELECT ?con (count (?con) as ?concount)
WHERE {
        ?lac skos:broader ?con
```

```
}
GROUP BY ?con
ORDER BY DESC (?concount )
```

If we want to focus on direct, objective relations between Lemmas, we may want to find entities that are relationally important within the scope of the entire Wordweb. One simple way to do this is by finding those Lemmas that have the highest overall number of distinct Pairings. To avoid being flooded by a cascade of low-Paired Lemmas, let's also limit the results to the absolute top:

```
SELECT ?lac1 (COUNT (distinct ?lac2) as ?pairpartnercount)
WHERE {
        ?lac1 rdf:type core:Concept .
        ?lac1 skos:related ?lac2 .
}
GROUP BY ?lac1
HAVING ( ?pairpartnercount > 100 )
ORDER BY desc(?pairpartnercount)
```

Words can be related to other words in a number of ways other than Pairings. A Lemma may literally form part of another Lemma, by being a single word inside a larger multi-word entity. Let's first find those single-word Lemmas that have the highest occurrence inside of multi-word Lemmas, and list them alongside the occurrence counts:

```
SELECT ?Lemma (count (?Lemma) as ?num)
WHERE {
        ?mwe rdf:type ontolex:MultiWordExpression .
        ?mwe decomp:constituent ?component .
        ?component decomp:correspondsTo ?Lemma .
        ?component rdfs:label ?label
}
GROUP BY ?Lemma
HAVING ( ?num > 100 )
ORDER BY desc(?num)
```

What if we want to inspect some of these results? Adding more values to the above query would soon get rather lengthy and messy, but if we decide on some criteria by which to filter out the results we want, we could take this a step further. Let's say we only want to look at the top ten Lemmas with the highest occurrences inside multi-word entries, and for each of those ten Lemmas we want to see all the Concepts to which they belong. Notice that we're effectively describing two separate searches here - finding Lemmas with occurence numbers, and finding

Concepts for Lemmas - where the first search feeds into the next. We can resolve this by *nesting* the first query inside the second. SPARQL will execute the innermost query first, then feed it to the outermost query:

```
SELECT ?Concept ?Lemma
WHERE {
        ?Lemma ontolex:sense ?Sense .
        ?Sense ontolex:reference ?LaC .
        ?LaC skos:broader ?Concept .
        {
                SELECT ?Lemma (count (?Lemma) as ?num)
                WHERE {
                        ?mwe rdf:type ontolex:MultiWordExpression .
                        ?mwe decomp:constituent ?component .
                        ?component decomp:correspondsTo ?Lemma .
                        ?component rdfs:label ?label
                }
                GROUP BY ?Lemma
                HAVING ( ?num > 100 )
                ORDER BY desc(?num)
                LIMIT 10
        }
}
```

Instead of queries based on properties, you may at some point want to search based on the textual contents. This may be the case when the information you want isn't encoded or easily available through formal property names and values - or when you simply don't have the time or inclination to trace through multiple property links.

In these cases, you'll want to rely on the FILTER function. A simple search for any Lemma containing the substring "maður" ("man") would run as follows:

```
SELECT ?Lemma
WHERE {
        ?Lemma ontolex:canonicalForm ?Form .
        ?Form ontolex:writtenRep ?Label .
        FILTER CONTAINS (?Label, "maður") .
}
```

The FILTER function can, in fact, be a remarkably powerful tool, in good part because it allows you to enter regular expressions (sometimes called "regex"). If you are planning on any kind of

text or text-pattern search, we strongly encourage you to familiarize yourself with these. Regular expressions may be used to match patterns of characters and strings, rather than the strings themselve. For example, if you want to narrow down the previous search, you could use a regex to find not all Lemmas *containing* "maður", but only those that *end* in the word (essentially running a search for a suffix):

```
SELECT ?Lemma
WHERE {
        ?Lemma rdf:type ontolex:Word .
        ?Lemma ontolex:canonicalForm ?Form .
        ?Form ontolex:writtenRep ?Label .
        FILTER REGEX (?Label, "maður$") .
}
```

In the Wordweb, the names of multiword Lemmas will usually also contain shorthand grammatical categorizations: "falla af hesti" (e. "fall off [a] horse") thus contains "so_fs_no-d" indicating a verb, a preposition, and a noun in the dative case. While efforts have been made to relate multiword lemmas to their single-word counterparts, these kinds of subdivisions do not include words that are not themselves entries in the Wordweb, including many prepositions. Instead, if you are cognizant of the grammatical shorthand for the phrase you're after, you can search for that particular categorization using regular expressions:

```
SELECT ?mwf
WHERE {
        ?mwf rdf:type ontolex:MultiWordExpression .
        ?mwf ontolex:canonicalForm ?can.
        FILTER regex(str(?can),"so_fs_no-d").
}
```

Some Wordweb entries contain pronoun stand-ins, often encased with angle brackets. There is no guarantee that these pronouns will be encoded separately in the Wordweb: Angle bracket contents aren't encoded no matter what, and pronouns aren't encoded unless the pronoun already exists as a standalone entry in the Wordweb.

However, some users might still want the ability to search for entries that house these pronouns. Doing so requires a raw text search for a particular inflectional form of a particular pronoun each time. We won't cover the obvious equivalents of "him", "her" or "it", but we do note that another standard word the user might want to search for is "einhver" ("someone").

An added complication is the angle brackets themselves, which are encoded in HTML with the %3C and %3E codes. Understandably, we may not want to deal with these in our results. To get

around that condition, we can use the BIND and REPLACE commands to find and replace these brackets. If we then wanted to find all entries containing the neutral gender of "someone" in the dative form ("einhverju"), encased in angle brackets, we would run the following:

```
SELECT ?canfinal
WHERE {
	?mwf rdf:type ontolex:MultiWordExpression .
	?mwf ontolex:canonicalForm ?can.
	FILTER regex(str(?can),"%3Ceinhverju%3E").
	BIND(REPLACE(STR(?can), "%3C", "<") AS ?can2).
	BIND(REPLACE(STR(?can2), "%3E", ">") AS ?can3).
	BIND(REPLACE(STR(?can3), "form_", "") AS ?canfinal).
}
```