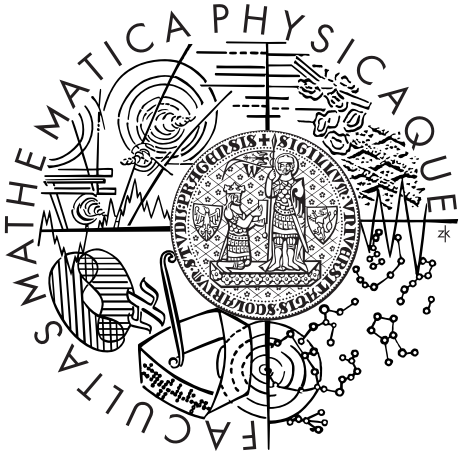
Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Jonáš Vidra

Extending the Lexical Network DeriNet

Institute of Formal and Applied Linguistics

Supervisor of the bachelor thesis:	doc. Ing. Zdeněk Žabokrtský, Ph.D.
Study programme:	Computer Science
Specialization:	General Computer Science

Prague 2015

I dedicate this thesis to my supervisor, Zdeněk Žabokrtský, and I would like to thank him for his support, advice, suggestions and guidance.

I would also like to thank Pavla Wernerová for proofreading the draft of this thesis.

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ……. date ……. signature of the author

Název práce: Rozšíření lexikální sítě DeriNet

Autor: Jonáš Vidra

Katedra: Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: doc. Ing. Zdeněk Žabokrtský, Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt: DeriNet je databáze českých lexikálních derivátů – lexikální síť, ve které uzly odpovídají lemmatům vybraným z Českého národního korpusu a hrany derivačním vztahům mezi nimi (například *práce* → *pracovat* → *vypracovat*). Vybírání lemmat z korpusu s sebou nese dva hlavní problémy: chybovost a chybějící lemmata, která by mohla sloužit jako spojnice mezi dosud nespojenými komponentami slovní sítě. Proto je potřeba najít spolehlivější a bohatší zdroj lemmat. Cílem této práce je rozšířit slovní zásobu DeriNetu pomocí lemmat z českého morfologického slovníku MorfFlex CZ a opravit derivační pravidla, která s novými slovy produkují chyby. Chybovost je měřena porovnáváním vztahů v databázi s ručně anotovanými daty vytvořenými v rámci práce.

Klíčová slova: DeriNet, derivace, lexikální síť, MorfFlex

Title: Extending the Lexical Network DeriNet

Author: Jonáš Vidra

Department: Institute of Formal and Applied Linguistics

Supervisor: doc. Ing. Zdeněk Žabokrtský, Ph.D., Institute of Formal and Applied Linguistics

Abstract: DeriNet is a database of Czech lexical derivates. It is a wordnet in which nodes represent lemmas sampled from the Czech National Corpus and edges represent derivational relations between them (such as *work* → *workable* → *unworkable*). Sourcing the lemmas from a corpus brings two problems: errors and missing lemmas that could link together currently unconnected clusters. Therefore, a more reliable and more complete source of lemmas is needed. The goal of this thesis is to extend the lexicon of DeriNet using lemmas sourced from MorfFlex CZ, a Czech morphological dictionary, and to correct the derivational rules that produce errors with the new lexicon. Error rate is measured by comparing the relations in the database with manually annotated data created as part of the thesis.

Keywords: DeriNet, derivation, lexical network, MorfFlex

Contents

1. Introduction

In the Czech language, there are several different word formation processes. Derivation – creation of new words by attaching morphological affixes to existing words – is the most productive one (Dokulil, 1986). There are hundreds of different affixes and many of them can be used for producing different meanings. For example, the suffix "-ka" can be used both for deriving diminutives, as in *krabička* (small box), and for deriving female variants, as in *řidička* (driver, *fem.*) (Dokulil, 1962; Štícha, 2012).

Derivation, as opposed to inflection, is not modeled by a single base lemma plus a set of derived words based on a paradigm. Derivational rules are much less regular than inflectional rules and they work recursively – a derived word can have its own derived words. Therefore, recursive derivations from a single lemma can form entire trees – so called derivational nests or clusters (Dokulil, 1962; Dokulil, 1986; Komárek, 1986). See figure 1.1 for an example of this behavior.

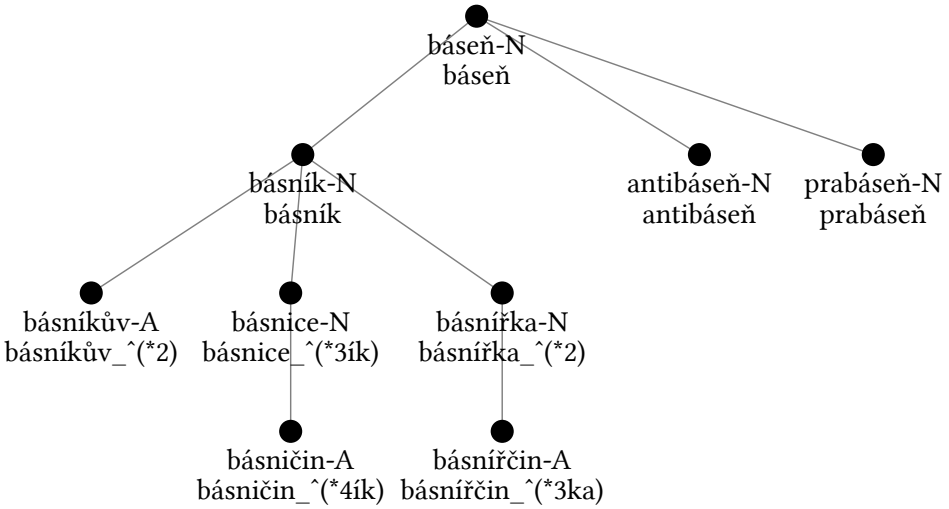


Figure 1.1: An example derivation tree showing recursive derivation of new lemmas.

1.1 DeriNet

DeriNet is a database of derivational relations between Czech words (Ševčíková, 2014b). It is a lexical network where vertices represent Czech lexemes and oriented edges represent derivational relations between them. Each lexeme represents one lemma with its meaning (word sense) partially disambiguated. Several lexemes may represent the same lemma in different senses if they differ in their derivational history, i.e. they are derived from different lemmas (such as *sladit* (to sweeten / to harmonize) from *sladký* (sweet) or *ladit* (to tune)) or they have different descendants, as in the case of *hnát* (vulgar form of limb / to drive) whose child is either *hnátek* (diminutive from hnát) or *hnaný* (the chased one).

In the database, a lexeme is a record identified by a unique numerical ID. It consists of the following fields:

lemma is the base word form (Hajič, 2004). This generally means nominative singular for nouns, nominative singular masculine positive for adjectives and infinitive positive for verbs.[1]

POS stands for Part Of Speech, denoted by a single-letter abbreviation. DeriNet only contains adjectives (A), adverbs (D), nouns (N) and verbs (V) (Ševčíková, 2014b), although some numerals classified as other POSes are also included, for example *čtyřnásobný A* (quadruple).

technical lemma ("techlemma") is a **lemma** plus its technical suffix in the style of the m-layer of the Prague Dependency Treebank (Bejček, 2013; Zeman, 2005; Hajič, 2004). This suffix is included for three reasons: it might contain information useful for our users, it sometimes includes a number useful for distinguishing homonymous lemmas[2] and it can also include information about the derivational parent. Other data encoded in the suffix, such as information about style and category, are currently not used by the build system.

parent is an ID of the derivational parent. Since each lexeme is currently limited to one parent at most, DeriNet does not allow representing compounding. This simplifies the structure – clusters (connected components) are rooted trees[3] instead of general oriented graphs.

See table 1.1 and figure 1.2 for an example of several lexemes. Version 0.9, released in December 2014, contains 305,781 lexemes and 117,327 oriented edges. Aside from version 0.9, there are also versions 0.9.1 and 0.9.2, which contain more derivations. These have not been released yet, but they are available for extending.

Version 0.9's lexemes are sampled from the SYN-2014 subcorpus of the Czech National Corpus (Hnátková, 2014) – the build process selects every adjective, adverb, noun and verb lemma that is mentioned at least twice, does not contain any punctuation symbols and digits, contains a lowecase letter and is at least two characters long.

Basing the dictionary on SYN-2014 has the advantage of only including lemmas that are actually used. It does, however, also include many spelling mistakes, typos and tagging errors. The frequency requirement lets some errors through, because wrong spellings of some common lemmas are more common than many rare lemmas. Furthermore, the process excludes correct-but-uncommon lemmas that could serve as derivation links between pairs of common lemmas – for example, the verb *domnít* is a parent to both *domnívat* and *domněnka*, but since it is not present in the database, these two lexemes are not connected together.

This selection process excludes many erroneous lemmas and abbreviations, but it still includes common spelling mistakes, typos and tagging errors. There exists a superior source of lexemes – the dictionary MorfFlex CZ, formerly known as "Hajič's

[1] Some words in negative are included as well – this is discussed in section 4.1.1.

[2] Note that the homonym number is, according to PDT documentation in (Zeman, 2005, p. 9), part of the lemma proper and not the technical suffix. We consider it to be technical information and strip it from the lemmas.

[3] Version 0.9 contains several cycles due to a programming oversight. See section 4.2.1 for details.

Table 1.1: An excerpt from DeriNet version 0.9. Each line represents one lexeme.

ID	lemma	technical lemma	POS	parent
113313	bulvárnost	bulvárnost	N	113314
113314	bulvární	bulvární	A	113312
113315	bulvárně	bulvárně_^(*1í)	D	113314
113316	bulvárový	bulvárový	A	
113317	buly	buly	N	
113318	bulící	bulící_^(*3it)	A	113294
113319	bulík	bulík-1_^(mladý_vůl)	N	113294
113320	bulík	bulík-2_,h_^(polštářek)	N	
113321	bulík	bulík-1_^(mladý_vůl)	N	
113322	bulík	bulík	N	
113323	bulíkovat	bulíkovat_:T	V	
113324	bulíkování	bulíkování_^(*3at)	N	113323
113325	bulíkův	bulíkův-1	A	
113326	bulíček	bulíček-1_,h	N	113319
113327	bulíček	bulíček	N	

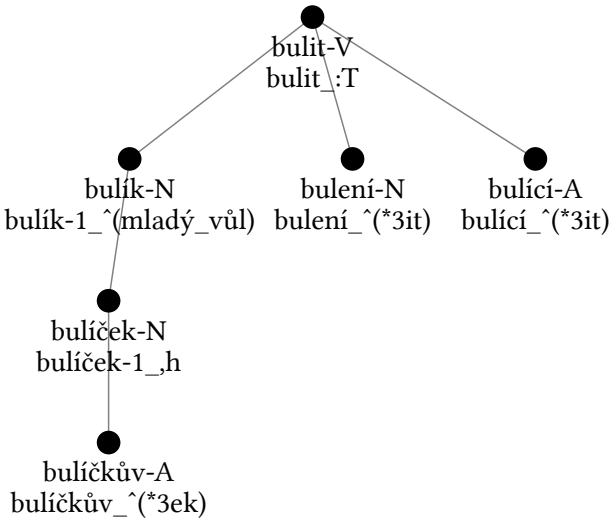


Figure 1.2: A tree from DeriNet 0.9 showing several lexemes from table 1.1.

morphology", (Hajič, 2004; HajičHlaváčová, 2013), which contains 1,001,704 unique lemma + part-of-speech-tag pairs.

The derivational relations are added using rule-based scripts. These come in two general flavors – a general rule with a list of exceptions, or a list of individual instances. When adding new lexemes, these rules would have to be revised by adding new exceptions to increase precision and new instances to increase recall.

The goals of this thesis are:

1. Fix any problems within the existing codebase of DeriNet.

2. Document code that DeriNet is composed of, both existing and new. Currently, documentation is limited to a paper explaining the linguistic background

(Ševčíková, 2014b), and to slides by (Žabokrtský, 2014) and (Ševčíková, 2014a) from a talk in December 2014. The new documentation should give potential DeriNet developers an overview of the structure and point out the key sections of the code.

3. Merge lemmas from MorfFlex into DeriNet and review and correct any errors, with a special attention given to precision.

4. Release DeriNet version 1.0 based on the results.

1.2 Structure of this thesis

Chapter 2 contains documentation of the DeriNet build system and explains how to install and use it and how to modify and write new modules.

Chapter 3 explains how I monitored precision and recall while working on the code, how I made the gold-standard data, and it documents the annotation format and the module used for measuring precision and recall.

Chapter 4 describes the changes I have made to the code to allow using lemmas from MorfFlex without a decrease in precision.

And finally, chapter 5 compares various versions of DeriNet and presents measurements of the quality of my work.

The contents of the enclosed CD are listed in Attachments.

2. Documentation of DeriNet

The DeriNet project is a set of resources used to build the DeriNet database; including Unix shell scripts, Perl scripts, Perl modules connected by a Treex-like API, Makefiles and text files with semi-automatically created derivational rules.

End users are not expected to use or even see the code. Only the end product, the compiled TSV file containing the selected lemmas and their derivations, is published – and even this file is of little use to most users, since its structure does not reflect the structure of the derivations. There are currently two tools that parse the file and present it in a friendlier way: a viewer by Milan Straka[1] and a search engine I have developed myself in spring 2015.[2] The search engine allows you to search for lexemes and clusters using regular expressions and a simple CQL-inspired grammar similar to the one the Czech National Corpus uses in its KonText tool[3] (Hnátková, 2014; Jakubíček, 2010). The viewer only selects lexemes by their techical lemmas, but it shows more statistics, such as an overview of tree shapes.

This chapter presents an overview of all the code and concepts DeriNet is composed of. Sections 2.1 and 2.2 give a quick overview of how to start the build process and get a newly-built database. Retracing these steps is not required of readers of this work, since I have included pre-built databases on the enclosed CD. Section 2.3 goes into detail on how the build system is structured. Section 2.4 explains how to use `derimor`, the main component of our build system, and, finally, section 2.5 shows how to write new modules for it.

2.1 Installation and requirements

The DeriNet build system requires standard Unix utilities, GNU Make, Perl 5, Treex and MorphoDiTa. If you use Linux, Make and Perl are probably included in your distibution. In the computer laboratories of the Faculty of Mathematics and Physics at Malá Strana, you can use preinstalled Treex and MorphoDiTa made by Martin Popel – just execute `source popem3am/preinstall/treex.sh` in your shell. Otherwise follow the instructions at `https://ufal.mff.cuni.cz/treex/install.html` and `https://ufal.mff.cuni.cz/morphodita/install`. Be aware that you have to install the SVN version of Treex, since the standard CPAN distribution does not contain `derimor`.

2.2 Building DeriNet

Start the build process by running `make` in directory *build/*. This will generate a compressed database file called *derinet*-version.*tsv.gz*. It is possible to choose between SYN and MorfFlex as a source of lemmas – the default is MorfFlex, but you can select SYN by passing `LEMMAS_SOURCE=syn` as an argument.

[1] `http://ufallab.ms.mff.cuni.cz/~straka/derinet-viewer/`

[2] `http://jonys.cz/derinet/search/`

[3] `https://kontext.korpus.cz/first_form?queryselector=cqlrow`

2.3 Build system

DeriNet is built by a set of Unix shell scripts and Perl modules for Treex connected by Makefiles. The process starts with extracting lemmas from the base dictionary, which varies depending on whether you use lemmas from MorfFlex or from SYN, continues with culling unwanted duplicates caused by polysemous and homonymous lemmas and finishes with executing several `derimor` scenarios that create the derivational links.

2.3.1 Lemma selection

SYN: The DeriNet distribution contains data sampled from the Czech National Corpus (Hnátková, 2014) – Czech lemmas and their counts in SYN-2014. They are further processed by a script `select_syn_lemmas.pl`, which selects lemmas of adjectives, adverbs, nouns and verbs that are found at least twice, do not contain a digit or punctuation, do not end with a capital letter (excludes some abbreviations) and are longer than two characters.

MorfFlex: The MorfFlex dictionary (HajičHlaváčová, 2013) contains Czech lemmas, word forms and their morphological tags. Script `select_morfflex_lemmas.pl` parses the file and extracts the lemmas and POS-tags for all adjectives, adverbs, nouns and verbs. The lemmas are also partially deduplicated to bring the size of the resulting file down – the script eliminates runs of equal lemma-POS pairs. Lemmas that contain digits are excluded, but lemmas with punctuation are preserved. MorfFlex does not contain punctuation, so this would only exclude useful lemmas such as *O´Connorův* or *e-mailový*. Also, abbreviations are excluded – they are recognized by the `_:B` techlemma flag.

After lemma extraction, the next phase is removing "false homonyms". The base dictionary may contain the same lemma more than once – as long as it differs in the POS or in the technical suffix. These duplicities are linguistically motivated by homonymy and polysemy. Many lemmas contain a so-called homonym number in their technical suffix precisely for the purpose of allowing more variants to coexist in the dictionary. They usually differ in inflectional forms or in their semantical meaning. However, many such duplicities do not differ in their derivation trees, which makes the distinction meaningless for DeriNet – for example, the noun *žebračka* has four different variants in MorfFlex: *žebračka*, *žebračka-3_,n_^(polévka_z_chleba)*, *žebračka-4_,n_^(žudro)* and *žebračka-5_,h_^(nejzadnější_kostelní_lavice_pro_žebráky)*. The non-numbered variant has a derivation *žebračč*in, while the three numbered variants do not, so we want to include it in DeriNet in a lexeme of its own. The numbered variants, however, do not differ in any aspect of their derivation history, which means that they can be coalesced into a single lexeme. Therefore, two of them are removed by checking them against a predefined list of classes of lemmas that share the same derivational history – the first lemma in each class is retained and the rest is deleted. These classes were created by Milan Straka by listing lemmas that share all inflection forms (Žabokrtský, 2014, p. 18).

2.3.2 Calling `derimor`

The rest of the build process is handled by `derimor`, a command-line application that is a part of the Treex framework (Popel, 2010). It is, however, simpler than the

standard `treex` application and not compatible with Treex modules.

2.4 Using `derimor`

2.4.1 Prerequisities

If you want to use `derimor` as a standalone application (outside of the Makefiles), you have to setup your `@INC`, so that Perl can find the required modules in the *build/lib* directory. This is done by adding the directory to the `$PERL5LIB` environment variable. In `sh` or `bash`, you can type `export PERL5LIB="$PWD"/build/lib:"$PERL5LIB"` to temporarily set it in the current shell, assuming your current working directory is at the top of the unpacked distribution archive. For a more permanent change, run `echo 'export PERL5LIB="`*`absolute_path_to_build/lib`*`:$PERL5LIB"' >> ~/.bashrc` if you use `bash`. Then try to run `derimor CreateEmpty Dummy Save file=test.tsv`. If you encounter an error similar to the one in Listing 2.1, it means your `$PERL5LIB` does not contain the *build/lib* directory. Correct output should look like the one in Listing 2.2 and the program should create a file called *test.tsv* with contents identical to Listing 2.3 – that file is actually a DeriNet database containing a single lexeme with lemma "testlemma".

Listing 2.1: Error encountered when the `@INC` has not been set correctly.

```
Can't locate Treex/Tool/DerivMorpho/Scenario.pm in @INC (
   ↪ you may need to install the Treex::Tool::
   ↪ DerivMorpho::Scenario module) (@INC contains: /etc/
   ↪ perl /usr/lib/perl5/vendor_perl/5.18.2/powerpc-
   ↪ linux /usr/lib/perl5/vendor_perl/5.18.2 /usr/lib/
   ↪ perl5/vendor_perl /usr/lib/perl5/5.18.2/powerpc-
   ↪ linux /usr/lib/perl5/5.18.2 .) at ./derimor line 6.
BEGIN failed--compilation aborted at ./derimor line 6.
```

Listing 2.2: Output of the command "`derimor CreateEmpty Dummy Save file=test.tsv`".

```
TREEX-INFO:     3.633:  Initializing Treex::Tool::
   ↪ DerivMorpho::Block::CreateEmpty
TREEX-INFO:     3.651:  Initializing Treex::Tool::
   ↪ DerivMorpho::Block::Dummy
TREEX-INFO:     3.677:  Initializing Treex::Tool::
   ↪ DerivMorpho::Block::Save
TREEX-INFO:     3.679:  Applying Treex::Tool::DerivMorpho
   ↪ ::Block::CreateEmpty
TREEX-INFO:     3.680:  Applying Treex::Tool::DerivMorpho
   ↪ ::Block::Dummy
TREEX-INFO:     3.682:  Applying Treex::Tool::DerivMorpho
   ↪ ::Block::Save
scenario: CreateEmpty Dummy Save file=dummy-test.tsv
Saving 1 lexemes
```

Listing 2.3: Contents of file *test.tsv* as created by "`derimor CreateEmpty Dummy Save file=test.tsv`".

```
0 testlemma          Dummy
```

2.4.2 Writing scenarios

A `derimor` scenario is a list of parameters, supplied directly on the command line. A scenario consists of "blocks" — instances of Perl modules — and their arguments. See Listing 2.4 for an example of a scenario. When executing a scenario, all the blocks are first initialized in the order in which they appear, and then executed in the same order.

This is similar to the Unix concept of pipelines and it means that lemmas and derivations created by any block are visible in the following blocks, but not in the preceding ones. Therefore, their order is important, because some modules do not modify existing derivations or create missing lexemes, and some others only reconnect existing derivations and do not create new ones. If you put a "reconnection" block before a "creation" block, the former will have no effect – and switching the order of two "do not modify existing links" blocks may produce widely different results.

Arguments are placed after the block they refer to and have a general format of *`argument=value`*.

The scenario in Listing 2.4 consists of 5 blocks: `CreateEmpty`, `CS::AddLexemesFromList` with two parameters, another `CS::AddLexemesFromList` with three parameters, `CS::AddDerivationsFromLemmaSuffices` and `Save` with one parameter.

Listing 2.4: An excerpt from a real-world scenario used for building DeriNet.

```
derimor \
  CreateEmpty \
  CS::AddLexemesFromList file=sorted_lemmas.tsv \
    dictionary_name=morfflex \
  CS::AddLexemesFromList file=extra_lemmas.tsv \
    dictionary_name=manual verify_lemma_uniqueness=1 \
  CS::AddDerivationsFromLemmaSuffices \
  Save file=derinet-example.tsv
```

2.5 Writing new modules for `derimor`

The API of `derimor` is very similar to Treex; with both utilizing a custom launcher for Moose classes written in Perl. Modules for `derimor` extend Moose class `Treex::Tool::DerivMorpho::Block` and the names you specify on the command line are prefixed by `Treex::Tool::DerivMorpho::Block::`, so any modules you write have to be placed in a *Treex/Tool/DerivMorpho/Block/* subdirectory somewhere in your Perl's `@INC`. If you use the provided infrastructure of Makefiles, you can store your modules directly in *build/lib/Treex/Tool/DerivMorpho/Block/*; otherwise use one of the directories listed in the environment variable `$PERL5LIB`.

You can see an example dummy module in Listing 2.5. This module creates a single lexeme as a hello-world-style demonstration of the API.

A module is executed by calling its function `process_dictionary` with a `Treex::Tool::DerivMorpho::Dictionary` as a parameter. You have to return the modified dictionary after you are done. If you did not override this function, the default implementation from `Treex::Tool::DerivMorpho::Block` calls your subroutine `process_lexeme` on each `Treex::Tool::DerivMorpho::Lexeme` in the dictionary. Therefore, your module has to override at least one of these methods.

Listing 2.5: An example `derimor` module that adds one lexeme to the database.

```
package Treex::Tool::DerivMorpho::Block::Dummy;
use utf8;
use Moose;
extends 'Treex::Tool::DerivMorpho::Block';

sub process_dictionary {
  my ($self, $dict) = @_;
  $dict->create_lexeme({
    lemma          => 'testlemma',
    lexeme_creator => $self->signature
  });
  return $dict;
}

1;
```

If you want to pass arguments to modules, simply add a Moose attribute declaration to the top of the source file, as in Listing 2.6. Pass a value by adding *`attribute_name=value`* directly after the block in the scenario, then access the argument using `$self->`*`attribute_name`*.

Listing 2.6: Example of adding an attribute named `file` to the module.

```
has file => (
  is            => 'ro',
  isa           => 'Str',
  documentation => 'file name to load',
);
```

2.5.1 Working with lexemes

Lexemes are represented in `derimor` by a class `Treex::Tool::DerivMorpho::Lexeme`. You can retrieve the lemma using `Lexeme->lemma`, techlemma using `Lexeme->mlemma`, POS using `Lexeme->pos` and parent lexeme using `Lexeme->source_lexeme`.

There are also functions for retrieving the immediate derivational descendants: `Lexeme->get_derived_lexemes()`, and the ultimate derivational ancestor: `Lexeme->get_root_lexeme()` (useful for walking through the whole cluster).

2.5.2 Finding lexemes

The Dictionary you get as a parameter when overriding function `process_dictionary` allows you to find and add lexemes and add or delete derivations.

The basic way of retrieving lexemes is getting the whole contents of the Dictionary and working with them by yourself. This is done by using function `Dictionary->get_lexemes()`, which returns an array of all lexemes. This is handy if you want to loop over them and find those that satisfy some complicated condition, but if you are looking for lexemes with a specific lemma, you should rather use the more specialized function, `Dictionary->get_lexemes_by_lemma($lemma)`. That one searches for lexemes matching the string `$lemma` using an internal hashtable, so it is more efficient than looping.

Even more specific function is `Dictionary->find_lexeme_pair($source_lemma, $source_pos, $target_lemma, $target_pos)` – it takes four strings (two lemmas and two corresponding parts-of-speech) and returns a pair of Lexemes: `($source_lexeme, $target_lexeme)`. This is the function you should use if you want to create a new derivation between two specific lemmas, because it takes into account existing derivations (prefers connected pairs, which helps with correcting or deleting specific existing derivations) and homonym numbers (prefers pairs with equal numbers). If you do not care about the POSes, just pass `undef`.

2.5.3 Creating new derivations

Derivations are added by passing a hash containing the proper parameters to `Dictionary->add_derivation()`. Supported parameters are `source_lexeme` and `derived_lexeme` for the parent and the descendant lexeme, respectively; then `deriv_type` for a short textual description of the derivation; and `derivation_creator` for the identification of the block that created the derivation. We recommend that you set all the parameters. You can (and should) use `$self->signature` (short name of your module) as the base value of the `derivation_creator`. Currently existing modules usually append the name of the file with rules or instances to the end.

Current modules use the `deriv_type` for an identification of the parent and descendant POS in the format *`parentPOS2descendantPOS`*, i.e. `V2A` for a derivation from a verb to an adjective. You are free to use it for anything you want, though.

Deleting derivations is done by calling `Lexeme->set_source_lexeme(undef)`. Do not use this method for adding derivations – it does not set the `deriv_type` and `derivation_creator` description fields.

2.5.4 Creating new lexemes

Lexemes are created by passing a hash with parameters to `Dictionary->create_lexeme()`. Supported parameters are `lemma`, `mlemma` and `pos`, which are self-explanatory, and `lexeme_creator` for the identification of the block that created the lexeme. As before, we recommend using `$self->signature` and a source file name as a basis for the `lexeme_creator` description.

Setting the creator is not required but highly recommended. The other three basic parameters are required and not setting them may result in strange behavior of other modules.

You should ensure that your lemma and mlemma match each other. If you do not know what to set the mlemma to, just use the lemma as a mlemma.

3. Quality measurement

Immediately after producing the first version of DeriNet using lemmas from MorfFlex (DeriNet 0.10.0) I have sampled 1,000 random lexemes and manually tagged their derivational parents. This was done "blindly with corrections", that is, I have first annotated the lemmas without consulting either the techlemmas or the database and then I ran a sample measurement using the data and re-thought the instances where my annotation differed from the database. Most of them were errors in DeriNet, but some were typos, misunderstandings or inconsistencies on my side – and those were annotated again. This means that the process is not entirely bias-free, but the potentially misannotated instances were seen two times.

The annotation has a simple format: The first two columns in a tab-separated-file are the lemma and its POS. The annotator writes the lemma of the derivational parent(s) into the third column. If there are variants (such as the lemma "Karlův", which can be derived from either "Karel" or "Karl"), all of them are written down, separated by commas. Composition is annotated by separating the parents by spaces. The annotator can also write a dash if the source lemma does not have a parent, a question mark if he does not know the answer, or an exclamation mark if he considers the source lemma to be erroneous.

The database is evaluated against the annotation using module `MeasurePrecisionRecall`, which I wrote for this purpose. It searches for lexemes with the lemmas from the first column and compares every single found lexeme to the values in the third column. Lemmas produced by composition are by default considered to be wrongly derived, since composition is currently not representable in DeriNet. There is an option `ignore_composition` which, when set, makes the module expect no parent in place of multiple parents – i.e. a composite lemma is correctly derived iff it has no derivational parent. I have had this switch enabled when taking all the measurements in this thesis.

After finishing the annotation, I produced a report on precision and recall. The results were underwhelming, because while DeriNet 0.9 had precision of about 0.97, DeriNet 0.10.0 had only 0.84. After reviewing the results of the tests, I have made a summary of the errors. There were 137 wrong derivations, out of which 117 were caused by iterative verbs and their derivates. These lexemes are generally connected to the right cluster, but they are skipping derivational links by connecting to the root instead of a deeper node. (See figure 4.1 for an example of this behavior, section 4.4.1 for the solution and figure 4.2 for an example of what the trees look like now.)

Apart from this set of data, which was used for development, I have prepared a second set for final evaluation, sampled from the final version. I did not review this evaluation set and I did not use the data during development, so the results should be representative. To further rule out my personal biases, I have asked an independent annotator to annotate the evaluation set too. This parallel set was used both for evaluation and for measuring inter-annotator agreement. Inter-annotator agreement is 0.878, when measured according to the following algorithm: lexemes are considered to be annotated differently if none of the annotated parents match or if one of the annotations contains one of (-,?,!) and the other contains a different symbol or a parent.

4. Porting to MorfFlex

This chapter presents a list of all the changes I have made between DeriNet version 0.9.2 and version 1.0.

4.1 Differences between SYN and MorfFlex

The SYN-2014 based dictionary from DeriNet 0.9 contains, after lemma extraction but before homonym removal, 310,117 unique lemma+POS pairs. MorfFlex contains (again, after extraction but before removing false homonyms) 994,659 such pairs. However, MorfFlex is not a strict superset of the SYN-based dictionary – there are 300,924 common pairs, 601 are only found in SYN and 662,206 are only found in MorfFlex. The 601 missing lemmas are nearly all tagging errors (*bělit A*, *ale N*, *podotkl N*, *jako N*), misspellings and errors in capitalization (*západďákč*, *afrodita*, *blančin*, *elektrostrojírenký*, *Ještě*), non-lemmas (*kadeřávkovskou*, *Holubovské*, *krojovaných*, *polická*, *chvíli*) and numerals (*čtyři*, *desetitisící*). There are only 19 correct lemmas in the SYN-based dictionary not found in MorfFlex. I have extracted them, had them tagged by MorphoDiTa and manually verified and injected the results into the DeriNet dictionary during the build process.

4.1.1 Negative adjectives

The amount of new lemmas also means that the module `AddOstLexemesFromCNC`, which was adding extra lexemes whose lemmas end with "-ost", became nearly unnecessary – the lemmas it is adding now are either negative adjectives or erroneous lemmas. I have prevented it from adding the wrong ones, but it is up for debate whether the negatives should be left in or also excluded. I have left them in so far, since even though they are superfluous, they create new derivations, they are well-formed and excluding them is as easy as deleting the appropriate line from *derinet09/Makefile*.

4.1.2 Abbreviation culling

DeriNet 0.9 excluded abbreviations by not including any lemma ending with a capital letter. MorfFlex allowed me to use a better way – a selection process based on the `_:B` techlemma flag that marks abbreviations. This is very reliable – from the 4,546 excluded techlemmas, only several were not abbreviations and none were useful Czech lemmas without other variants in the database.

Some abbreviations have remained in the database, such as *TANAP* or *kW*. There is, however, only a handful of them, so I have decided not to do anything about it.

4.2 Violations of structural constraints

4.2.1 Cycles

DeriNet version 0.9 contained four cycles in its structure, for example *poddaná* → *poddaný* → *poddaná*. This was highly problematic, because the structure is supposed

to be a set of trees and the tools used for working with the database should be able to depend on this.

The cycles seemed to be caused by an erroneous direction of derivation between *poddaná* (vassal, *fem.*) and *poddaný* (vassal, *masc.*), but that turned out to be an unrelated problem – see section 4.5.1 for details. The real cause of the cycles were duplicate lemmas created by remove_false_homonyms.pl, which prompted me to rewrite that script.

The old version relied on a particular ordering of input lemmas (lemmas that were going to be substituted must have appeared after the lemma they were being substituted to) and printed duplicities whenever this constraint was violated, which in turn caused cycles to appear downstream. The new version processes input in two passes, which is slower, but also more reliable, because the script now sees the whole database at once. It is also better at choosing the proper lemmas – I have written a scoring heuristic to help lemmas with the richest technical information pass through. If some of the variants contain derivational information in the technical suffix, it only chooses among those. It favors longer techlemmas in hope that these will be more informative, but penalizes lemmas with style classification and thus favors stylistically neutral ones.

4.2.2 Non-matching techlemmas

Many lexemes created by `AddOstLexemesByRules` and `AddOstLexemesFromCNC` had techlemmas that did not match their lemmas, for example the adjective *oklamatelný* had a techlemma *neoklamatelný* and *necitlivost* had techlemma *citlivost_^(*3ý)*. There were 1,687 such pairs just with the prefix "ne-". The immediate cause, which I have fixed, is using results from morphological analysis without checking that they are correct. The underlying cause is that these "lemmas" are not actually lemmas. MorphoDiTa considers negative words to be inflectional variants of the affirmative ones, so a lemma should always be affirmative.

4.3 Derivation history

The TSV output format of `derimor` includes information about the type of the derivation, name of the module that created the lexeme and name of the module that created the derivation. In version 0.9 it did not include the derivation history of the lexeme, however, which made it hard to discover where or why was a lexeme reconnected and where it belonged before the reconnection. I have added history tracking which records all the former authors and former derivational parents.

Aside from adding the tracking code to `DerivMorpho::Lexeme`, this also required revising existing modules, because some of them did not fill the derivaton-creator information in.

4.4 Errors in MorfFlex

Apart from the 19 missing lemmas, there are more errors in MorfFlex. The one big issue – misleading derivation comments in techlemmas – is easily solvable within

DeriNet. The other issues are minor and I have decided to merely report them to the MorfFlex development team and have them solved upstream.

4.4.1 Derivational comments

729 derivational comments refer to lemmas that do not exist in the lexicon. For a full list see *suffixes_bad_words.tsv*. I have added the correct ones manually via *extra_words.txt*; the rest is ignored. Also, one techlemma links to itself: *dovotvírat_:T-_,h_ˆ(ˆGC*0)*; this is ignored.

Furthermore, many derivational comments do not point to the immediate derivational parent, but skip it and connect to a lexeme higher in the cluster. I have found two big systematic examples of this behavior – iterative verbs and their derivates and feminine possessives. Then I wrote a new module, `CS::ReconnectVerbalDerivatives`, that would reconnect them.

The module is, by default, strict – all its rules contain a suffix of the current (incorrect) parent, which is checked before reconnecting. The module therefore does not reconnect lexemes that have already been reconnected by a different block and by default only reconnects lexemes within their cluster, which helps to ensure that the rules will not affect semantically unrelated lexemes that just happen to be orthographically close. There is, however, an "aggressive mode" which relaxes the checks somewhat and allows connecting to lexemes outside the cluster. This aggressive mode is enabled by default, because all of its 2,714 extra results have been manually checked and were found to be sound.

The module reconnected 111,725 lexemes. See figures 4.1 and 4.2 for an example of what the clusters looked like before and after the reconnection, respectively.

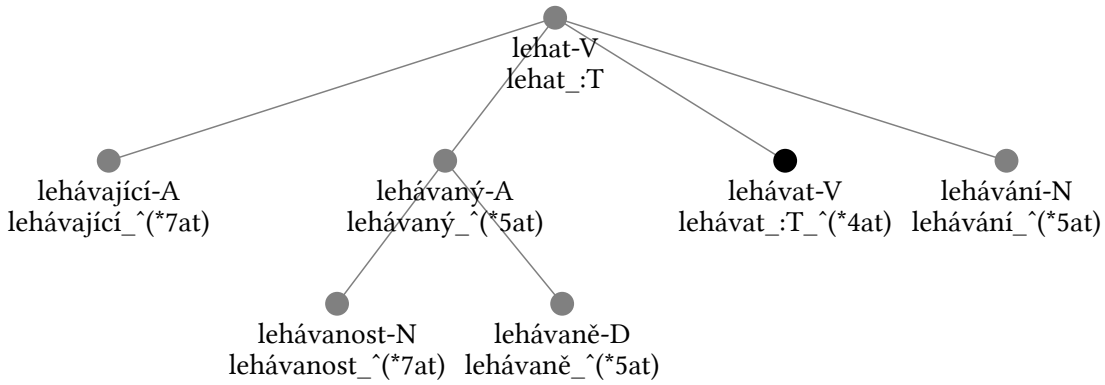


Figure 4.1: An excerpt from the tree of *lehat* within DeriNet 0.10.0. Notice that the actual children of *lehávat* are incorrectly placed directly below *lehat*.

4.4.2 Homonym numbers

Whenever MorfFlex contains one lemma several times, either because of polysemy or because of homonymy, these lemmas are distinguished by having different "homonym numbers" in their technical suffixes. However, the homonym numbers are sometimes inconsistent. According to the manual (Zeman, 2005, p. 10), when

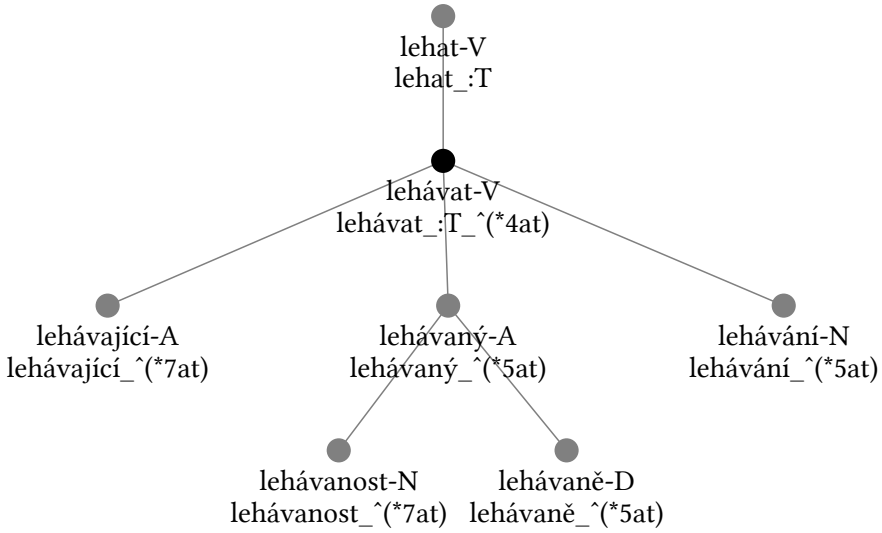


Figure 4.2: An excerpt from the tree of *lehat* within DeriNet 1.0. The children of *lehávat* are now positioned properly.

a lemma has more than one variant, all of them should be numbered and, if possible, distinguished using comments (_ˆ(...)). This is often not the case, 1,793 lemmas in DeriNet 1.0 have inconsistent numbering where the lemma has both numbered and unnumbered variants. The unnumbered variant is often a "miscellaneous" version without any other meaningful tags, but sometimes it is also the main sense of the lemma: *železnice* (railway) × *železnice-1_,s_ˆ(výrobce_železa)_(*5ík-1)* (ironsmith, *fem.*). Therefore, the unnumbered variants cannot be easily deleted.

There are also some stranger cases: *čápek-1_ˆ(malý_čáp)* × *čápek-2_,n_ˆ(malý_čap)* (little stork). There seems to be no difference between those two variants.

4.4.3 Wrong lemmas

MorfFlex contains an occasional misspelling or error. Examples: *zjudaizoovaný*, *opajcnnou*, *nabotnání*, *adenosinmonofosát*, *Vávrůův*. I have done nothing about these with the hope that a future version of MorfFlex will remove them. There is only a handful of them anyway.

Apart from that there are also lemmas that are colloquial or vernacular variants of lemmas in offical usage. They are not "wrong" per se – the issue is that they are introduced irregularly. Some of the lemmmas do have such variants listed, others do not. For example, *vdýchnutelně* (inhalably) has a variant *vdejchnutelně*, but an analogous variant of *vdýchnout* (inhale) – *vdejchnout* – is not there. Because of the missing lemmas, the variants often do not form clusters.

4.4.4 Wrong POS tags

There are at least 95 lemmas with wrong part-of-speech tags; for example *zestetičtění V*, *zcitlivěný V* and most adjectives ending with "t".

4.4.5 Non-lemmas

There are also items in the dictionary that are real Czech words, but are not lemmas. It is not obvious how to find these, but I have encountered at least 36 such verbs; for example *znovuzasunu* (slip in again, *masc. sg. fst.*), *zklevetěn V* (gossiped, *masc. sg. past*) or *odkřemič V* (desiliconize, *imperative*); by searching for verbs that end with neither "t" nor "ci", the standard Czech infinitive suffixes.

4.5 Revised derivational rules

4.5.1 Male and female variants of nouns

Some nouns in the database had an opposite direction of derivation filled in; for example *Železná → Železný*. This error was fixed by modifying a single rule in *Block/-CS/manual.AddManuallyConfirmedAutorules.rules.tsv*: the line

```
*     61 N-á N-ý # prosim zpracovat s obracenym smerem odvozeni!!!
```

was processed with the opposite direction of derivation.

4.5.2 Reconnected possessives of names

In DeriNet 0.9.2, many possessives of male names are connected to the female variants; such as *Petrův* to *Petra*. Others are unconnected, but they were connected in DeriNet 0.9; for example *Františkův*. This is a problem with the annotation – the meaning of many annotation instances was probably "this variant is also possible", but the module which uses the annotation interprets them as "reconnect these unconditionally". This results in good derivations being deleted in favor of less probable ones – see figure 4.3 for an example. The problem does not have an easy solution, all the rules would have to be revised and split between two different modules – an "eager" one and a "lazy" one. Some of the lexemes would also have to be duplicated to satisfy all possibilities.

I have fixed several instances of this problem in the annotation, but many other remain.

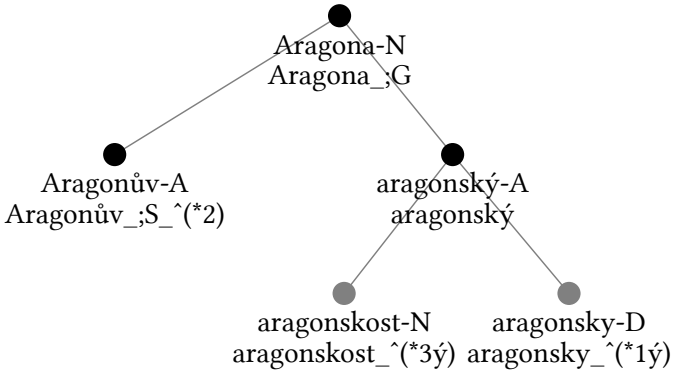


Figure 4.3: A tree showing less-than-optimal placement of *Aragonův* and *aragonský* below *Aragona*. They would be better placed under *Aragon*.

4.5.3 Missing common rules

Many derivations in DeriNet are added by using lists of instances – that is, by explicitly stating which lexemes to connect. This is error-proof, but adding new lemmas means that more instances have to be added to match the enlarged database. I have reused 18 rules from *manual.AddManualyConfirmedAutorules.rules.tsv* that were marked as reliable and generated 27,062 new derivative relations, which were then manually verified and cleaned.

I have also edited `AddManuallyConfirmedAutorules` to support loading rules and instances from non-default locations.

4.6 Assorted improvements

4.6.1 Techlemma parsing

Module `AddDerivationsFromLemmaSuffices`, which parses derivational information in the suffixes of techlemmas, did not recognize all the instances. This was due to an erroneous regular expression, which I have fixed. This change has added 8,394 new derivations. I disabled adding derivations from some variants of these derivational suffixes, because some, such as `_^(^HN**…)`, add strange links. For example, lemma *Honza* has a techlemma *Honza_;Y_^(^HN**Jan)* and tries to connect to *Jan*, but the actual (etymological) connection between these two names should go through *Johann*.

4.6.2 Homonym number removal

The module `AddDerivationsFromLemmaSuffices` contained another error – in an algorithm for homonym number removal. This caused many possible derivations to go unnoticed. It was only discovered thanks to manual reviewing of the output of `ReconnectVerbalDerivatives` – many lexemes did not reconnect, because the module is conservative and did not attempt to fix an incomplete cluster.

A similar error was found in `AddAdj2AdvByRules` before – there, the homonym numbers stayed behind and proliferated into lemmas in the resulting database. There were 1,326 extant lemmas with homonym numbers in DeriNet, fixing this error had removed all of them.

4.6.3 Conversion to MorphoDiTa

Modules `AddAdj2AdvByRules`, `AddOstLexemesFromCNC` and `AddOstLexemesByRules` require a morphological tagger and lemmatizer to derive techlemmas from lemmas when adding new lexemes. DeriNet 0.9 used CzechMorpho for this task, but CzechMorpho is no longer being developed and has been deprecated in favor of MorphoDiTa (Straková, 2014). I have therefore converted these modules to the new tagger. This change was seamless and did not result in any reconnections.

4.6.4 Choosing from homonymous variants

Many modules in DeriNet have to cope with the fact that lexemes may have identical lemmas. Previously, this was mostly ignored – whenever a module searched for a

lexeme with a particular lemma and got more than one result, it simply chose the first one on the list. This had obviously lead to many strange connections: in DeriNet 0.9, *balíkův-2_ˆ(člověk)_(*4-2)* (hillbilly's) was connected to *balík-1_ˆ(předmět)* (package), while it should have been connected to *balík-2_ˆ(člověk)* (hillbilly).

I have written a procedure, `find_lexeme_pair`, which tries to find two lexemes matching a pair of lemmas and POSes. Since it has control over both ends of the derivation link, it can choose the best pair, taking into account existing derivations and homonym numbers. This rectifies several hundreds of these errors. Moreover, converting every applicable module to use this new procedure had another advantage of unifying the code – previously, every module had had its own version, each slightly different.

This does not fix all instances of similar errors, however, because not all derived lemmas have the same number filled in. For example, *stanice* (station) and *stojatý* (upright or stagnant) are still connected to *stát-2_ˆ(něco_se_přihodilo)* (happen) instead of *stát-3_ˆ(někdo/něco_stojí,_např._na_nohou)* (to stand), because their techlemmas lack a homonym number altogether.

4.6.5 Miscellaneous fixes

I have fixed the module `RevertDerivationDirection`, which refused to initialize, by adding `1;` to the end. Modules have to return a true value after they are done initializing.

I have restructured the Makefiles. Previously, each step had to be called manually; this was especially tedious since the build system is split into three directories. Now, you can simply type `make` in the main directory and wait for the result.

The module `AddOstLexemesByRules` is supposed to connect adjectives to nouns, but it did not contain a check for the POS of the adjective and thus it sometimes incorrectly added derivations to other parts of speech. Thus, *komornost A* (intimity) was connected to *komorný_ˆ(osoba) N* (butler) instead of *komorný A* (gentle). I have added this check.

In `Dictionary`, there was a hard-to-discover bug with saving the database. A derivation for a particular lexeme is stored primarily in the field `source_lexeme` in the lexeme itself. But there is a second place – the source lexeme keeps a backreference in a hash called `derived_lexemes`, which is updated every time the `source_lexeme` of the linked lexeme is changed. The fact that it is a hash and not an array is an optimization and a source of the aforementioned bug – the lexemes there are kept using `$derived_lexemes{$lexeme} = $lexeme`. The first occurrence of `$lexeme` in this assignment is cast to string, which depends on its location in memory. And when saving the database to a `.slex` file and reloading it, these locations change, making those backreferences impossible to delete from the linked lexemes. I have fixed this by giving every lexeme an additional unique ID.

5. Statistics

For a comparison of basic statistics between different versions od DeriNet, see table 5.1 and a corresponding graph in figure 5.1.

Description of the version numbers:

0.9 is the version from December 2014, without any of my changes.

0.9.2 is the unreleased version from January 2015 – it differs from 0.9 in having additional rules. This was the basis of my work.

0.9.3 differs from 0.9.2 in including all the fixes and improvements I have made while writing this thesis, but it is still made using lemmas from SYN.

0.10.0 differs from 0.9.2 in utilizing lemmas from MorfFlex. These are plugged directly into the old 0.9.2 infrastructure.

1.0 is the final version with both lemmas from MorfFlex and all the fixes and improvements I have made while writing this thesis. It differs from 0.10.0 in having the improvements and from 0.9.3 in having the new lemmas.

Table 5.1: Side by side comparison of selected statistics for various versions of DeriNet.

Version:	0.9	0.9.2	0.9.3	0.10.0	1.0
Lexeme count:	305,781	305,210	305,471	973,702	968,967
Unique lemma count:	303,174	303,174	303,432	970,194	965,535
Derivational link count:	117,327	124,577	137,090	686,803	715,729
Cluster count:	188,462	180,643	168,381	286,899	253,238
Singleton cluster count:	129,672	122,048	105,372	132,812	101,311
Maximum lexemes per cluster:	38	40	42	82	82
Maximum cluster depth:	7	6	7	7	8

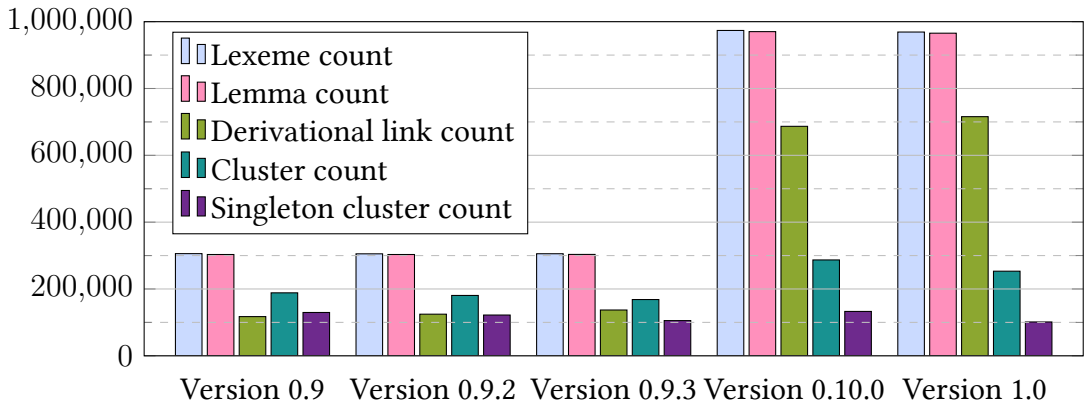


Figure 5.1: Contents of the table 5.1: Side by side comparison of selected statistics for various versions of DeriNet in graphical form.

As evidenced by this data, a lot of the improvements stem just from using lemmas from a better source; this alone allowed creation of new derivational links. See table 5.2 and graph in figure 5.2 for counts of derivations added by various creators. The increase for the three most productive modules (`AddAdj2AdvByRules`, `AddOstLexemesByRules` and especially `AddDerivationsFromLemmaSuffices`) between 0.9.3 and 0.10.0 is caused directly by the new lemmas.

On the other hand, my own additions are visible, too: The decrease for module `AddDerivationsFromLemmaSuffices` between 0.10.0 and 1.0 is caused by the module `ReconnectVerbalDerivatives`, which I wrote to correct the systematical errors in `AddDerivationsFromLemmaSuffices`. These errors have not been so severe in versions based on lemmas from SYN – `ReconnectVerbalDerivatives` reconnects only very few lexemes in 0.9.3. Also, the higher numbers for `AddManuallyConfirmedAutorules` in versions 0.9.3 and 1.0 are caused by the new derivation instances I have generated and verified.

Version 0.9 is not included because the distribution TSV does not contain the required data and I have been unable to replicate the build of that version exactly. Also be aware that the numbers for the modules in the old-code versions (0.9.2 and 0.10.0) may be a little bit off because of because of errors in their codebase – and additionally, some of them are zero because `ReconnectVerbalDerivatives` did not exist yet and `AddOrDeleteLinksInClusters` did not report its name when creating the derivations.

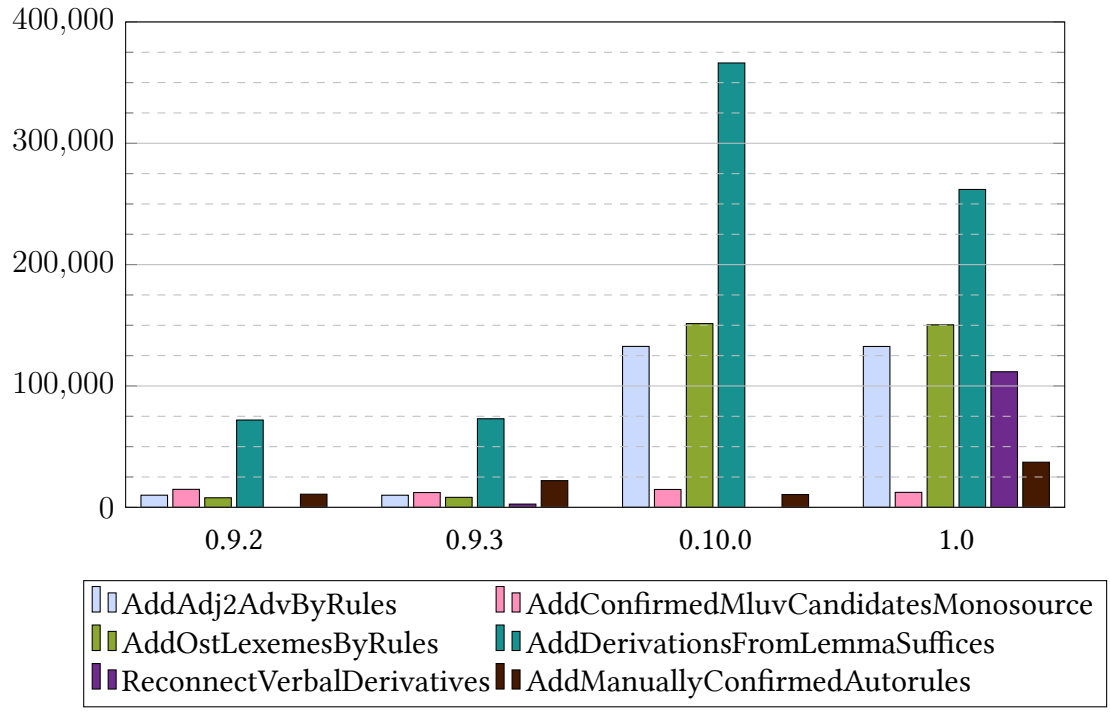


Figure 5.2: A comparison of productivity for selected modules. Other modules that are not shown either did not change their output between versions much or they are too unproductive to be visible on the chart.

Table 5.2: A comparison of productivity for all DeriNet modules. The numbers are total derivations made by this module that remain in the finished database. (Derivations later reconnected by a different module do not count.)

Version:	0.9.2	0.9.3	0.10.0	1.0
AddAdj2AdvByRules:	9,986	9,961	132,603	132,550
AddConfirmedMluvCandidatesMonosource:	14,808	12,181	14,701	12,331
AddDerivationsFromLemmaSuffices:	71,937	72,981	366,180	261,948
AddDerivationsFromList:	2,853	2,697	2,841	2,693
AddManuallyConfirmedAutorules:	10,801	21,977	10,543	37,172
AddManuallyConfirmedAutorules2:	3,523	3,519	3,582	3,592
AddOrDeleteLinksInClusters:	0	1,547	0	1,561
AddOstLexemesByRules:	7,857	8,209	151,399	150,389
Prefixes:	1,462	1,458	2,934	2,929
ReconnectVerbalDerivatives:	0	2,643	0	111,731
RevertDerivationDirection:	9	9	9	9
RevertDerivationDirectionSpecial:	619	619	623	623

What is not visible in these graphs is the improvement in precision and recall that has been directly caused by my improvements – the sole amount of derivations does not say anything about their correctness. See table 5.3 for measurements of precision and table 5.4 for measurements of recall based on various annotations. There are three sets of data – my own development set, my evaluation set *Eval 1* and an evaluation set *Eval 2* made by an external annotator, Kristýna Merthová. Be aware that the statistics consider lemmas created by composition to be connected correctly iff they have no derivational parent, because these lemmas are outside the scope of DeriNet.

Table 5.3: A report on precision.

Version:	0.9	0.9.2	0.9.3	0.10.0	1.0
Devel:	0.97	0.98	0.98	0.84	0.99
Eval 1:	0.97	0.98	0.98	0.87	0.99
Eval 2:	0.93	0.92	0.92	0.82	0.94

Table 5.4: A report on recall.

Version:	0.9	0.9.2	0.9.3	0.10.0	1.0
Devel:	0.65	0.68	0.73	0.72	0.88
Eval 1:	0.69	0.72	0.75	0.75	0.88
Eval 2:	0.69	0.72	0.74	0.72	0.85

The singleton cluster counts are similar for all five versions, and there indeed is quite a large common base of singletons, but a lot of them are also different. The figure 5.5 shows that there is a big difference between singleton clusters in SYN-based and MorfFlex based versions, with a third of them disappearing and about the same amount of new ones appearing. On the other hand, nearly no new singleton clusters have emerged between versions 0.10.0 and 1.0 and a lot of them has been eliminated.

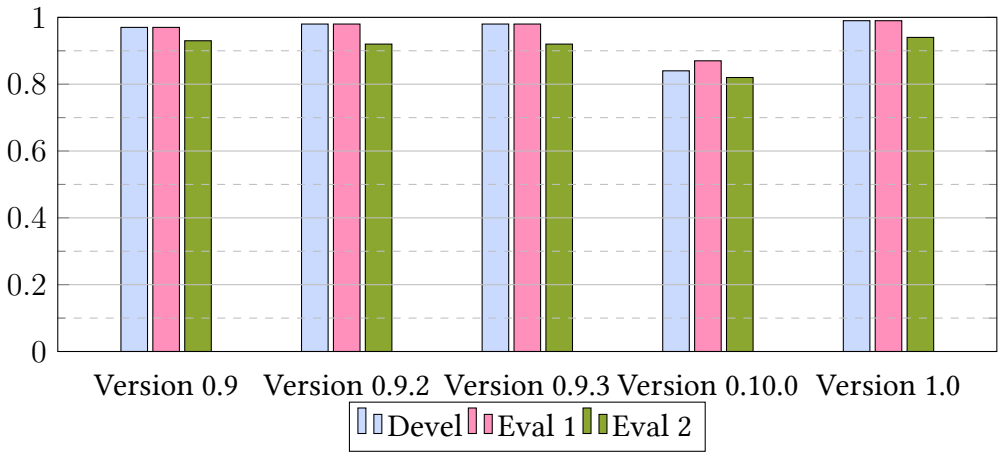


Figure 5.3: A graph of precision on the three data sets.

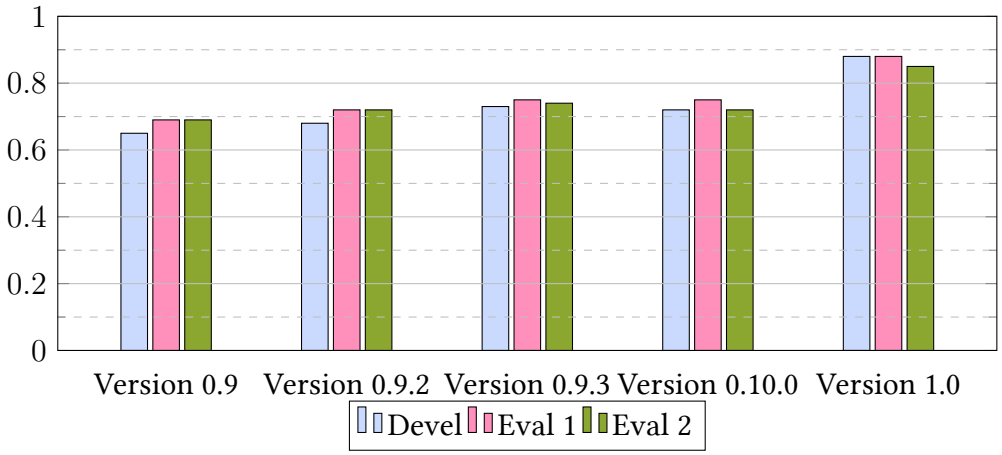


Figure 5.4: A graph of recall on the three data sets.

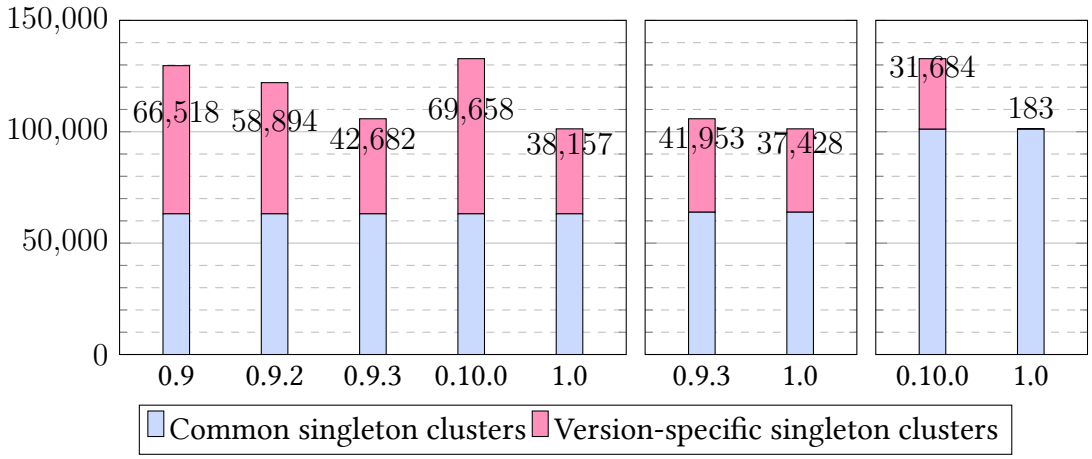


Figure 5.5: Comparison of singleton cluster counts and their composition. The common clusters are computed within each subgroup – i.e. in the leftmost plot the blue part represents clusters common to all five versions, while in the rightmost one it represents clusters common to the two MorfFlex-based versions.

6. Conclusion

In this thesis I have documented DeriNet's internals and enlarged DeriNet with lemmas imported from MorfFlex, a large dictionary of Czech word forms. The documentation is intended for developers, not end-users, since our users interface with DeriNet through helper programs.

I have corrected a key issue that prevented DeriNet from interfacing cleanly with these programs, namely cycles in the database. Besides that, I have fixed many smaller issues, including wrong calculation of statistics and erroneous numbers at the end of some lemmas.

After porting DeriNet to using lemmas from MorfFlex, which in itself was an easy task, I have also corrected hundreds of existing derivational rules and added tens of thousands of new ones. A new module, `ReconnectVerbalDerivatives`, was written to amend systematical errors in MorfFlex.

I have been monitoring the results of every change I have made and I have ensured that only changes with net positive impact make it through. To aid me with these checks, I have manually annotated 1,000 lemmas as gold-standard data for development and a further 1,000 for evaluation. The evaluation set was annotated in parallel by a second person. The measured precision and recall is summarized in chapter 5 – I have exceeded the precision and recall of version 0.9.2, which served as the basis of my work.

6.1 Future work

As already mentioned in Section 4.1.1, some of the lemmas of adjectives have both their affirmative and negative form in DeriNet. The forms with negative polarity are superfluous, because negation is considered to be an inflectional process and it is marked in the morphological tags. Several are generated by AddOstLexemesFromCNC and AddOstLexemesByRules, but many others are included in MorfFlex itself and getting rid of those is not easy.

Also, DeriNet contains mistagged numerals. This is an issue I will try to solve in the upcoming months.

We should continue solving problems with homonymy and polysemy. Many lexemes contain semantically different variants of the same lemmas and it is hard to programmatically discern which variant to choose when adding a derivation. My attempts described in section 4.6.4 have been only partially successful. Moreover, this issue is nearly invisible because homonyms are unmarked in the gold-standard data – it is impossible to mark them without consulting the database itself, which is likely to introduce biases. However, since there are only several hundreds of homonymous lemmas, the issue is not particularly pressing.

Perhaps the biggest issue is a lack of on-demand creation of new lexemes, especially duplicating existing ones. For example, "Karlův" can be derived both from "Karel" and "Karl", but right now we can only select one of them as a parent. We therefore either need a way of connecting a single lexeme to two parents and marking one of the connections as "alternative", or an automatic way of copying the lexeme to the second position. The latter version is probably cleaner and easier, but we would have to categorize and describe the cases where duplication is applicable by

using special annotation, which is a lot of work. Currently, only two modules create new lexemes when they do not find a viable candidate – `AddOstLexemesByRules` and `AddOstLexemesFromCNC`, but others should be able to do this too. However, we cannot allow duplication for every rule, because many lexemes should be simply reconnected without copying.

Acknowledgements

Data for measurement of inter-annotator agreement have been prepared by Pavla Wernerová and Kristýna Merthová.

This work has been using language resources developed and/or stored and/or distributed by the LINDAT/CLARIN project of the Ministry of Education of the Czech Republic (project LM2010013).

Bibliography

BEJČEK, Eduard; HAJIČOVÁ, Eva; HAJIČ, Jan, et al., 2013. *Prague Dependency Treebank 3.0*. LINDAT/CLARIN digital library at Institute of Formal and Applied Linguistics, Charles University in Prague. Available also from WWW: ⟨http://hdl.handle.net/11858/00-097C-0000-0023-1AAF-3⟩.

DANEŠ, František; DOKULIL, Miloš; KUCHAŘ, Jaroslav, 1967. *Tvoření slov v češtině 2: Odvozování podstatných jmen*. Prague: Academia.

DOKULIL, Miloš, 1962. *Tvoření slov v češtině 1: Teorie odvozování slov*. Prague: Nakladatelství Československé akademie věd.

DOKULIL, Miloš; HORÁLEK, Karel; HŮRKOVÁ, Jiřina; KNAPPOVÁ, Miloslava; PETR, Jan, 1986. *Mluvnice češtiny 1*. Prague: Academia.

HAJIČ, Jan, 2004. *Disambiguation of rich inflection: computational morphology of Czech*. Prague: Karolinum. ISBN 978-80-2460282-0.

HAJIČ, Jan; HLAVÁČOVÁ, Jaroslava, 2013. *MorfFlex CZ*. Available also from WWW: ⟨http://hdl.handle.net/11858/00-097C-0000-0015-A780-9⟩.

HNÁTKOVÁ, Milena; KŘEN, Michal; PROCHÁZKA, Pavel; SKOUMALOVÁ, Hana, 2014. The SYN-series corpora of written Czech. In. *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. pp. 160–164.

JAKUBÍČEK, Miloš; KILGARRIFF, Adam; MCCARTHY, Diana; RYCHLÝ, Pavel, 2010. Fast Syntactic Searching in Very Large Corpora for Many Languages. In. *PACLIC*. pp. 741–747.

KOMÁREK, Miroslav; PETR, Jan; KOŘENSKÝ, Jan, 1986. *Mluvnice češtiny 2: Tvarosloví*. Prague: Academia.

POPEL, Martin; ŽABOKRTSKÝ, Zdeněk, 2010. TectoMT: modular NLP framework. In. *Proceedings of IceTAL, 7th International Conference on Natural Language Processing*. Pp. 293–304.

ŠEVČÍKOVÁ, Magda; ŽABOKRTSKÝ, Zdeněk, 2014a. *DeriNet: Lexikální databáze českých derivátů* [online]. [Visited on 2015-07-15]. Slides of a talk given by Magda Ševčíková at Linguistic Mondays, December 15, 2014, Charles University in Prague. Available from WWW: ⟨http://ufal.mff.cuni.cz/~zabokrtsky/derinet/derinet-Sevcikova.pdf⟩.

ŠEVČÍKOVÁ, Magda; ŽABOKRTSKÝ, Zdeněk, 2014b. Word-Formation Network for Czech. In CHAIR), Nicoletta Calzolari (Conference; CHOUKRI, Khalid; DECLERCK, Thierry, et al. (ed.). *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. Reykjavik, Iceland: European Language Resources Association (ELRA). ISBN 978-2-9517408-8-4.

ŠTÍCHA, František; JANEČKA, Martin; KOUTOVÁ, Marta, et al., 2012. Miloš Dokulil and his theory of productivity in word-formation. *Korpus – gramatika – axiologie*. Vol. 6, pp. 3–9.

STRAKA, Milan; STRAKOVÁ, Jana, 2014. *MorphoDiTa: Morphological Dictionary and Tagger*. Available also from WWW: ⟨http://hdl.handle.net/11858/00-097C-0000-0023-43CD-0⟩.

STRAKOVÁ, Jana; STRAKA, Milan; HAJIČ, Jan, 2014. Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition. In. *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Baltimore, Maryland: Association for Computational Linguistics, pp. 13–18. Available also from WWW: ⟨http://www.aclweb.org/anthology/P/P14/P14-5003.pdf⟩.

ŽABOKRTSKÝ, Zdeněk; ŠEVČÍKOVÁ, Magda, 2014. *DeriNet: Lexikální databáze českých derivátů* [online]. [Visited on 2015-07-15]. Slides of a talk given by Zdeněk Žabokrtský at Linguistic Mondays, December 15, 2014, Charles University in Prague. Available from WWW: ⟨http://ufal.mff.cuni.cz/~zabokrtsky/derinet/derinet-Zabokrtsky.pdf⟩.

ZEMAN, Dan; HANA, Jiří; HANOVÁ, Hana, et al., 2005. *A Manual for Morphological Annotation*. 2nd ed.

Attachments

1. Structure of the enclosed CD:

   - *derinet-*.tsv* – prebuilt databases of various versions.
   - *annot-*.tsv* – manually annotated data. *Devel* was made from DeriNet 0.10.0, *Eval 1* and *Eval 2* are based on DeriNet 1.0.
   - *annot-eval-diff.txt* – a list of lemmas which are annotated differently in *Eval 1* and *Eval 2*.
   - *build/* – the build system directory.
   - *build/Makefile* – the main Makefile that handles the build process by recursively calling sub-Makefiles in *derinet09/*, *derinet091/* and *derinet092/*.
   - *build/data/* – data files required for building DeriNet.
   - *build/data/pro-zdenka-syn-lemata-od-ondreje.gz* – data sourced from the Czech National Corpus: lemmas, POSes and their counts in SYN-2014.
   - *build/data/morfflex-cz.2013-11-12.utf8.lemmaID_suff-tag-form.tab.csv.xz* – distribution package of MorfFlex CZ.
   - *build/data/straka_same_lemma_sense_classes.txt* – equivalency classes for removing superfluous homonymous and polysemous lemmas.
   - *build/lib/* – modules for `derimor`.
   - *build/derinet09*/* – build directories for the three stages of the build process. They do not correspond to DeriNet 0.9, DeriNet 0.9.1 and DeriNet 0.9.2, since I have made changes to all of them, but the historical structure is preserved.