Bachelor thesis
Computing Science

Radboud University

Implementation, Evaluation and Comparison of Python Modules implementing Morphological Analysis and Generation

Author:
Tom Aarsen
s1027401

First supervisor:
prof. dr. ir. D. Hiemstra
djoerd.hiemstra@ru.nl

Second assessor:
prof. dr. M.A. Larson
m.larson@let.ru.nl

March 21, 2021

Abstract

The existing `Lingua::EN::Inflexion` Perl module implementing morphological analysis and generation for British English has been ported to Python and improved upon to create the new `Inflexion` Python module. Our `Inflexion` allows the conversion of any noun, verb, or adjective to a specific wordform, such as singular, plural, past, past participle, or present participle. Both a qualitative and quantitative evaluation procedure are introduced to evaluate this new module. The qualitative evaluation procedure classifies predicted outputs of algorithms like `Inflexion` to give details on the kind of errors that are made. On the other hand, the quantitative evaluation procedure allows such algorithms to be compared in terms of the accuracy of their predictions. These evaluation procedures will be used to compare and contrast `Inflexion` to eight competing Python modules with similar functionality. This results in a recommendation of which module should be used for each type of conversion, based on the characteristics of the input data that will be used for the conversion. `Inflexion` outperforms all tested Python modules for morphological analysis and generation of nouns and is very competitive for morphological analysis and generation of verbs. `Inflexion` is open source, and publicly available at `https://github.com/tomaarsen/Inflexion`.

Contents

Chapter 1

Introduction

The English language has undergone drastic changes through its evolution from Indo-European to West Germanic to the English as we know it [1]. Many changes have been introduced throughout its evolution, including the *reduction of inflections* [1], causing the language to devolve into a complicated collection of edge cases. As a result, programmatically converting even the simplest words from e.g. singular to plural is difficult.

Differences in pluralisation depending on grammatical person, number or gender and mountains of edge cases are notable factors in making natural language inflection a challenging topic. The distinction between multiple accepted variants only add to the challenge, e.g. modern and classical plural:

- The modern plural of `penny' is `pennies', while the classical plural is `pence'.

Due to this difficulty, complicated systems would need to be manually developed for the simplest interfaces. Instead, counterintuitive language is often used, e.g.:

- `Search results:  1' as opposed to `1 search result.'
- `3 book(s) found!' as opposed to `3 books found!'
- `Banana, stock:  3' as opposed to `3 bananas in stock.'

Luckily, morphological analysis and generation algorithms are capable of executing these conversions, and many have been implemented in Python modules. This way, developers can use a simple interface for these difficult conversions. However, these algorithms are far from sufficiently reliable.

For example, only 3 of the 8 previously released Python modules discussed within this thesis are correctly able to determine that the plural of the noun `politics' is still `politics'.

However, this poor performance does not take away the usefulness

of these algorithms in natural language processing (NLP) fields. Fields such as *text summarization*, *machine translation* and *question answering systems* frequently have to convert words into different forms. This further stresses the importance of morphological analysis and generation algorithms.

This thesis acts as a state of the art of such morphological analysis and generation algorithms, and introduces a new module to compete with the existing modules. The research question (RQ) for this thesis is:

RQ What is a (better) implementation of a morphological analysis and generation algorithm of British English, validated through evaluation?

With the following sub-questions:

RQ 1 What is a good procedure to evaluate the performance of morphological analysis and generation algorithms for British English?

RQ 2 What is our judgment on the evaluation of our morphological analysis and generation algorithm of British English in comparison to existing modules with similar functionality?

In order to answer these research questions, we have made the following advancements:

- We have ported the Perl module `Lingua::EN::Inflexion` [2] by Damian Conway to Python, where its functionality can be used alongside Python's breadth of Machine Learning and Natural Language Processing modules. See *Chapter 3: Lingua::EN::Inflexion* for information on this Perl module and for the reasoning about why this module was chosen to expand upon.

- We have expanded this port of `Lingua::EN::Inflexion` to create `Inflexion`, with the aim to further improve the performance of the port. See *Chapter 4: Inflexion* for the changes made, and *Chapter 7: Results* to see the performance gains. The highlights are:
  - Large rework of verb conversions.
  - Large improvements in collocation[1] support, especially for verbs.
  - Additions, removals and modifications of morphological analysis and generation rules.
  - Casing and whitespace between inputs and outputs is now preserved, e.g.:
    - `` Mother in Law' `` to plural becomes `` Mothers in Law' ``.
  - Fixes of bugs present in the Perl module `Lingua::EN::Inflexion`.

  Our `Inflexion` Python module is open sourced and published at `https://github.com/tomaarsen/Inflexion`.

[1] See *Chapter 2: Preliminaries* for a definition.

- We have developed a *qualitative* evaluation procedure for morphological analysis and generation algorithms. It classifies test results into 10 separate categories, useful for better evaluation and comparison of algorithms. See *Section 5.1: Qualitative evaluation procedure* for more information.
  - We also introduce a secondary evaluation procedure in *Section 5.2: Quantitative evaluation procedure*, which is *quantitative* and focuses on accuracy between modules.

- We compared our direct port of Damian Conway's `Lingua::EN::Inflexion` and our own `Inflexion` to 8 other well-known Python modules. Each of these 10 modules support a section of all morphological analysis and generation functionality for the English language:
  - `inflect` [3]
  - `Inflection` [4]
  - `Inflector` [5]
  - `LemmInflect` [6]
  - `NLTK` [7]
  - `Pattern` [8]
  - `PyInflect` [9]
  - `TextBlob` [10]

  These modules have been compared and contrasted with both of our evaluation procedures using data from `CELEX` [11], a lexical database by the Dutch Center for Lexical Information. See *Chapter 6: Existing Python modules* for information on these modules and *Chapter 8: Discussion* for a discussion regarding the performance of these modules.

Chapter 2

Preliminaries

This chapter will introduce some terms that aid the understanding of the remainder of this thesis.
Morphological analysis revolves around splitting a given word up into a root and bound morphemes, which occur as parts of words [12]. Two types of bound morphemes exist [13, 14]:

- **Derivational morphemes**
  These morphemes change the semantic meaning of the given word, for example:
  - The word `unhappy' consists of root `happy' and derivational morpheme `un-', which has inverted the meaning of the root.
  - The word `coexist' consists of root `exist' and derivational morpheme `co-', which has changed the meaning of the root.

  Alternatively, derivational morphemes may change the *part of speech* (POS) category of a word. Examples of such POS categories include nouns, verbs, adjectives, articles, etc.
  - The word `huggable' has the adjective POS and consists of a root `hug' and derivational morpheme `-able'. The root `hug' is a verb and when the derivational morpheme `-able' is added, the result becomes an adjective.

- **Inflectional morphemes**
  For verbs, these morphemes modify the *tense*, *aspect*, *mood*, *person* or *number* of a word. Similarly, for nouns, adjectives or pronouns these morphemes modify the *number*, *gender* or *case*.
  These modified categories are called grammatical features, for example:
  - The word `mice' has root `mouse' and number as plural.
  - The word `theirs' has root `mine', number as plural, gender as neuter and case as genitive (i.e. possessive).

- The word `` `are' `` has root `` `be' `` and either:
  - tense as present, person as 2nd, number as singular, e.g. `` `you are' ``.
  - tense as present, number as plural, e.g. `` `we are' ``.

  The root produced is a simple and irreducible morpheme, while these inflectional morphemes change the form of a word. In linguistic morphology, *inflection* is the process of word formation where a word is modified to express the information from the grammatical features. [13]

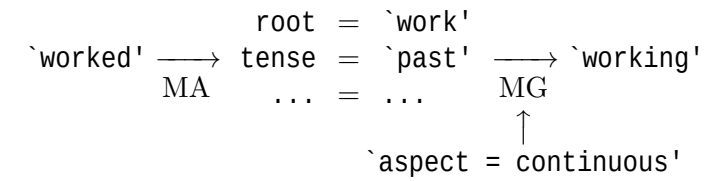
Morphological generation is the counterpart of morphological analysis. It takes a lemma, POS and information on the type of inflection to be applied, and generates the inflected word [15]. A lemma is a word in the canonical form, that other related words are represented by. For English nouns, the lemma is the singular, while for verbs it is the plural, e.g.:

- The lemma `` `go' `` represents the inflected forms `` `go' ``, `` `goes' ``, `` `going' ``, `` `went' `` and `` `gone' ``.
- The lemma `` `penny' `` represents the inflected forms `` `penny' ``, `` `pennies' `` and `` `pence' ``.

And as a result:

- With the lemma of `` `mine' ``, POS as noun, number as plural, gender as neuter and case as genitive, morphological generation produces `` `theirs' ``.

Morhological analysis can be used as a preprocessing step before applying a morphological generation algorithm. That way, rather than only allowing roots to be converted to any wordform, every word can be converted. Such an algorithm could be structured like so:



Ideally such an algorithm works on both words and *collocations* [16]. Collocations are a series of words or terms that co-occur more often than would be expected by chance, either separated by spaces or hyphens.

- When given the collocation `` `baby-sit' ``, the verb POS and number as singular, morphological generation should only inflect the `` `sit' `` portion of the collocation, resulting in `` `baby-sits' ``.
- With the collocation of `` `son of a gun' ``, the noun POS and number as plural, the result of morphological generation should be `` `sons of guns' ``.

- With the collocation `mother in law', the noun POS and number as plural, the result of morphological generation should be `mothers in law'.

All of these examples, and many more, differ from merely applying the algorithm on each word individually and then adding the words back together. It is clear that a good morphological analysis and generation algorithm ought to work on collocations too.

Chapter 3

Lingua::EN::Inflexion

The Perl module `Lingua::EN::Inflexion` [2] by Damian Conway implements a morphological analysis and generation algorithm, allowing the conversion between many wordforms.
It acts as the foundation for our Python module `Inflexion` that will be introduced within this thesis. From now on, `Inflexion` will be used to refer to our Python module, which was newly developed by us. `Lingua::EN::Inflexion` will be used to refer to Damian Conway's Perl module.

The most important reasons that `Lingua::EN::Inflexion` was chosen as the basis for `Inflexion` are:

- Wealth of functionality
- Adaptability, extendibility and upkeep
- Reputation

The remainder of this chapter includes elaboration on these reasons, followed by an overview of the bugs and limitations we found in `Lingua::EN::Inflexion`.

3.1 Wealth of functionality

`Lingua::EN::Inflexion` has constricted itself to the following functionality, providing functions for each wordform[1]:

- **Verbs**, detecting and converting to the following wordforms:

[1] `Lingua::EN::Inflexion` supports more functionality such as converting a number to cardinal or ordinal form. However, these are not relevant to morphological generation, so this functionality is not mentioned.

  - **singular** *e.g.*`he flies'
  - **plural** *e.g.*`we fly'
  - **past** *e.g.*`we flew'
  - **present participle** *e.g.*`he was flying'
  - **past participle** *e.g.*`we have flown'

- **Nouns**, detecting and converting to the following wordforms:
  - **singular** *e.g.*`brother'
  - **plural** *e.g.*`brothers'
  - **classical plural** *e.g.*`brethren'

- **Adjectives**, detecting and converting to the following wordforms:
  - **singular** *e.g.*`my'
  - **plural** *e.g.*`our'

The functions `singular` and `plural` allow the grammatical person to be included as a parameter. These functions influence the grammatical number too. Similarly, `past` affects the grammatical tense, and `present participle` and `past participle` affect the grammatical aspect.
The large number of grammatical features that are modified by these functions allows for these five common wordforms to be generated.

In addition, the support for a classical plural is a rare bonus, which is neatly implemented in the morphological analysis and generation rule files. Support for adjective conversion is useful, but not a particularly difficult challenge, as only a fixed number of adjectives differ between singular and plural.
Because adjectives can thus be perfectly converted using a small lookup table, adjective conversions are not tested within this thesis.

3.2 Adaptability, extendibility and upkeep

This section gives readers insight in the structure of the morphological analysis and generation rules that are used within `Lingua::EN::Inflexion`. This information may be useful in understanding the changes that our `Inflexion` makes on top of Damian Conway's `Lingua::EN::Inflexion`, as described in *Chapter 4: In exion*.

In constrast to the other currently existing modules that will be covered within this thesis, `Lingua::EN::Inflexion` is the one module that can be modified and kept up to date by users without any knowledge of programming. The flexibility offered by this implementation is attractive. It allows for easy upkeep, updating and customisation for specific tasks, such as inflecting strictly according to British or American English.

This all is due to how `Lingua::EN::Inflexion` generates its inflection rules from easily readable text files that follow a specific format, that can be edited by anyone with a text editor. Each of the three supported POS (noun, verb and adjective) have their own file with their own format.

One of these files is `` `nouns.lei' ``, which supports many types of rules required for rule-based inflection:

- For uninflected (e.g. tennis) or irregularly inflected nouns there must be rules directly from a singular word to a plural word. This way these rules have priority over regular inflectional rules like appending an `` `s' `` when converting a noun from singular to plural.

  The uninflected or irregularly inflected nouns have the following syntax in the format:
  - `` `singular => (modern) plural' ``, e.g. `` `tooth => teeth' ``.

  Note that all of these rules go both ways. The example states that the plural of `` `tooth' `` is `` `teeth' ``, and also that the singular of `` `teeth' `` is `` `tooth' ``.

- For singular nouns with multiple inflections there must be rules with multiple plurals. These will have the following syntax in the format, for example:
  - `` `singular => modern plural | classical plural' ``, e.g. `` `brother => brothers | brethren' ``

  Like before, this also states that the singular of `` `brethren' `` is `` `brother' ``.
  The addition of a classical form by adding a part of a rule after a `` `|' `` is allowed for all rules.

- For singular nouns ending in a specific suffix there must be rules that replace its suffix by a new suffix, producing a plural. It is required that there is at least one character before this suffix. These will have the following syntax in the format:
  - `` `-singular_suffix => -plural_suffix' ``, e.g. `` `-s => -ses' ``

  As such, this rule will match `` `lens' `` and convert it to `` `lenses' ``, and match `` `virus' `` to convert it to `` `viruses' ``, but will not match `` `s' `` because there is no character before the suffix.

  This `` `-' `` must be present as the first character on both the singular form and the plural form(s) of the noun for the rule to be a suffix-rule. Otherwise, the rule is simply about inflection of a noun with a dash in it, such as `` `break-away' `` → `` `break-aways' ``.

- Alternatively, the format for singular nouns ending in some phrase, such that there does **not** need to be any characters before this phrase have the following syntax in the format:
  - `` `*singular => plural' ``, e.g.
    `` `*child => children' ``

  So, this rule will match `` `grandchild' ``, `` `brainchild' `` and also `` `child' ``.

  Like with the `` `-' `` suffix, this rule that matches any characters in place of the `` `*' `` only works if the `` `*' `` is the first character.

- For convenience, it is helpful to have constructions for merging similar rules:
  - `` `-ao => -aos' ``
  - `` `-eo => -eos' ``
  - `` `-io => -ios' ``
  - `` `-oo => -oos' ``
  - `` `-uo => -uos' ``

  These can be merged into the following construction:
  - `` `-[aeiou]o => -[aeiou]os' ``

  For this rule, the brackets indicate that exactly one of the enclosed characters must be in that position. The content of the brackets must be identical on both the singular and plural side.

  On a similar note, the following notation can be used to indicate that the rule matches as long as *no* enclosed character is in that position.
  - `` `-[^ns]sis => -[^ns]ses' ``

- Recall that these rules are used both for converting between wordforms and for detecting whether a word is of a specific wordform. Some rules are useful for transforming between plural and singular or vice versa, but do not work to indicate whether a given noun is in a specific form. Two rules in particular fit this description:
  - `` `-s => -ses' ``
  - `` `- => -s' ``

  These need to exist as the most general, last-attempt rules, but cannot be used to determine plurality. For example, `` `-s' `` is not a good indicator for whether a noun is plural. `` `virus' ``, `` `bass' ``, `` `cyclops' ``, `` `genius' ``, etc. are all singular, despite ending in `` `-s' ``.

  So, we need a tag to show that a rule should not be used for the purpose of identifying whether a word is of a particular form:

  - `<nonindicative> - => -s'

  So, `book' may match this rule from singular to plural, converting it to `books', but the rule `-' (which matches any non-empty word) will not be used to identify whether a word is singular, as it would consider all non-empty words to be singular.

- In order to inflect nouns that require sub-nouns to be inflected, some abbreviations need to be introduced:

(PREP)	Matches any preposition. E.g. `over', `on', `in', `out', etc.
(SING)	Matches any word. This word is deemed to be a singular noun.
(PL)	Matches any word. Corresponds to the plural form of the singular noun from (SING).
*	Matches any number of characters (including 0).

  With these abbreviations in place, rules for inflecting nouns that require sub-nouns can be written:

  - `(PREP) it => (PREP) them'
  - `son-of-a-(SING) => sons-of-(PL)'
  - `(SING)-(PREP)-* => (PL)-(PREP)-*' (e.g. `mother-in-law')

  Whenever an abbreviation is used on one side of the `=>', a corresponding abbreviation must be used on the other side too.

Any line not containing a rule following one of the above formats (i.e. any line not including a `=>') is treated as a comment. Similarly, every character after the `#' mark is treated as a comment. Furthermore, all occasions of multiple spaces are trimmed to just one space when the algorithm reads the format.
This way, the rule file format allows for using spaces for formatting purposes, blank lines, lines with documentation, and comments after specific rules. As a result, the format feels user-friendly and lax.

As is noticeable in the example rules, some rules will overlap. For example, the word `kiss' matches rules for `-ss', `-s' and `-'. In such cases, the first rule has priority.

Lastly, all rules containing `-' that do not signify a suffix rule (e.g. `break-away') will also match when there is a space in place of the dash.
This is implemented by the algorithm, which parses the format of the rules file, and generates an executable program with the desired morphological generation functions described in *Section 3.1: Wealth of functionality*.

The format for verbs as used in `verbs.lei' is very similar, but rather than using this noun format:

- `singular => plural | classical plural'
  e.g.`*brother => brothers | brethren'

`verbs.lei' has the following format:

- `singular plural preterite present participle past participle'
  e.g.`flies fly flew flying flown'

Similarly, `adjectives.lei' is implemented with the following line format

- `singular => plural'
  e.g.`my => our'

Note that these rulesets are also used for morphological analysis, i.e. for detecting wordforms. With other words, the rule `*child => children' states the following facts:

- A noun ending in `child' that is converted to *plural* will have `child' replaced with `children'. (Morphological Generation)
- A noun ending in `children' that is converted to *singular* will have `children' replaced with `child'. (Morphological Generation)
- A noun ending in `child' is deemed *singular*. (Morphological Analysis)
- A noun ending in `children' is deemed *plural*. (Morphological Analysis)
- Lastly, because there is no specific rule for the classical plural, it copies the rule for regular plural:
  - A noun ending in `child' that is converted to *classical plural* will have `child' replaced with `children'. (Morphological Generation)

Because this ruleset largely revolves around general rules rather than very specific outliers, `Lingua::EN::Inflexion` very easily extends for unknown words. Even words that do not exist will often be converted like one would expect.
This is where such a rule-based system is guaranteed to outperform the competing method of a lookup table. With a lookup table, words that match one of the entries in the table *exactly* will be converted according to the table. This system works well for common words, but requires immensely many rows if accuracy of *any* word is important. As a result, solely using a lookup table is no longer state of the art.

Note that although `Lingua::EN::Inflexion` is designed with British English inflection rules in mind, American English conversions are

generally also possible. For example, if a user asks it to generate the plural of the American English noun `color', it will still correctly generate `colors'. This is a consequence of how the inflectional rules between American and British English are very similar. If you input American English, the output is likely going to be American English too. The same holds for British English.

In conclusion, any user can easily modify an inflection rule through these files, rather than having to delve deep into code to do so, which is a refreshing change of pace.

The technical implementation details of `Lingua::EN::Inflexion` will not be discussed within this thesis. Interested readers are advised to look at the Perl Module page at `https://metacpan.org/pod/Lingua::EN::Inflexion` for more information.

3.3 Reputation

Damian Conway, the author of `Lingua::EN::Inflexion`, also wrote the predecessor `Lingua::EN::Inflect` [17]. According to Conway himself, `Lingua::EN::Inflexion` has a cleaner and more convenient interface, with many more features, and is much better tested [17].

However, the predecessor `Lingua::EN::Inflect` is what many of the often recommended Python modules such as `inflect` [3], `Pattern` [8] and `TextBlob` [10] are based on. This inspires hope that our `Inflexion` module based on the seemingly superior module `Lingua::EN::Inflexion` from Damian Conway can outperform the often recommended Python modules.

3.4 Bugs and limitations

Though Conway mentions there are no outstanding bugs, he recognises that the complexity of the English language goes beyond what can reasonably be coded. During our process of porting `Lingua::EN::Inflexion` from Perl to Python, we discovered several undiscovered bugs and fixed them in the conversion from this direct port to the modified `Inflexion`:

- Classical plural of nouns ending in `'s' (like in `Tom's') where the phrase before has capitalisation will cause the following two warnings, and then return only `'s'.
  - `Use of uninitialized value $plural in pattern match (m//) at Nouns.pm line 7969.`

  - `Use of uninitialized value $plural in concatenation (.) or string at Nouns.pm line 7969.`
  - e.g. `` `One's' ``, `` ` one's' `` or `` `ONE'S' ``.
- The several forms of the verb `` `to be' `` are not converted to the correct person if they contain capitalisation, when converting to singular.
  - e.g. `` `AM' ``, `` `Are' `` or `` `Is' ``.
- The plurals of several noun collocations will be given a hyphen when they should not be.
  - e.g. the plural of `` `court martial' `` is considered `` `court-martials' ``.
- The order of inflection rules are applied strictly in order, causing some shorter rules to be applied before longer ones.
  - e.g. The verb `` `shear' `` matches `` `*hear' `` before matching `` `*shear' ``. As a result, the past of `` `shear' `` is considered `` `sheard' `` instead of `` `shorn' ``.
  - e.g. The noun collocation `` `walk of life' `` matches `` `*life' `` before matching `` `(SING)-(PREP)-* => (PL)-(PREP)-*' ``. So, the plural of `` `walk of life' `` is considered `` `walk of lives' `` instead of `` `walks of life' ``.

We have fixed all of these bugs and several less significant ones in the creation of `Inflexion`, after which many changes with the intent of improvements were implemented. Both the direct port of `Lingua::EN::Inflexion` and `Inflexion` will be evaluated against other existing Python modules using the evaluation procedure described in *Section 5.1: Qualitative evaluation procedure*. See *Appendix A: Qualitative evaluation results* for the results of the evaluation.
The changes that converted the Python port of `Lingua::EN::Inflexion` into `Inflexion` will be described next.

Chapter 4

Inflexion

The aim of our efforts was to port the existing Perl module `Lingua::EN::Inflexion` to Python and expand upon it in an attempt to improve it. Porting it to Python should allow a wider audience to enjoy the work. For context, approximately 70% of all GitHub repositories tagged as Natural Language Processing (`` `NLP' ``) are written in Python, as compared to less than 1% for Perl [18].

Beyond merely creating a port to Python, we made changes to this `Lingua::EN::Inflexion` Python port to create our own open-sourced module `Inflexion`. These changes can be classified by the following categories:

- General changes.
- Changes for nouns.
- Changes for verbs.

The technical implementation details of our module[1] `Inflexion` are not discussed, as they mainly overlap with `Lingua::EN::Inflexion`. Instead, the changes relative to `Lingua::EN::Inflexion` are described.

The changes described in this chapter were devised in the following ways:

1. Some changes were inspired by existing research. In particular, morphological generators (and analysers) were used, such as `morphg` and `morpha` [15, 19] from *Section 9.2: Minnen et al. (2000), Minnen et al. (2001)*.

   Furthermore, the paper of Bloch [20] entitled "English Verb Inflection" from *Section 9.5: Bloch (1947)* was useful in providing

[1] Readers interested in the precise implementation of `Inflexion` may look at `https://github.com/tomaarsen/Inflexion` for source code.

information on inflectional rules and a list of irregular verbs.

Lastly, several other existing modules mentioned in *Chapter 6: Existing Python modules* were analysed to see if they contained rules that could be adapted into `Inflexion`.

We had high hopes that existing rulesets from previous research or modules would prove useful, but we ended up not adapting much from them.

2. Other changes were based on careful and thorough inspection of the morphological analysis and generation rules from `Lingua::EN::Inflexion`. This includes investigating whether certain rules have unintended consequences, e.g. change 7 for *Section 4.3: Changes for nouns*:
   - The `` `*louse => lice' `` rule designed for `` `louse' `` and `` `headlouse' `` also unintentionally states that the singular of `` `blouse' `` is `` `blice' ``.

   Many small issues were found and removed by studying the consequences of rules starting with `` `- `` and `` `* ``.

3. We also passed testing data from *Section 5.3: Testing data* (without the known correct results) through both our `Inflexion` and another one of the eight modules tested in this thesis. Whenever the predicted outputs between `Inflexion` and the other module differed, we manually determined whether this was an error on `Inflexion`'s side. If so, we marked it down as a potential issue to overcome.

   Note that our goal was not to discover edge cases of single words, but find inflectional patterns such as the rule we discovered in change 8 for *Section 4.3: Changes for nouns*:
   - `` `-[aeo]use => -[aeo]uses' ``

   Naturally, this method does not find issues that exist in both tested modules. To combat this downside, each of the eight other modules was used as the secondary tested module.
   The majority of all changes were based on errors found through this method.

4. Lastly, the qualitative evaluation introduced in this thesis in *Section 5.1: Qualitative evaluation procedure* was applied on `Inflexion` with all data from *Section 5.3: Testing data*. This resulted in precise information for which kinds of words `Inflexion` performed poorly, e.g.:

- The direct port of `Lingua::EN::Inflexion` had 269 cases out of 830 reported errors where the output of the conversion from plural to singular noun *should* have ended in `` `e' ``, but the predicted output ended in `` `is' `` instead.

  This helped identify that the rule `` `-[^ns]sis => | -[^ns]ses' `` has an undesirable downside. This rule states that the classical plural of a word matching `` `-[^ns]sis' `` has the form `` `-[^ns]ses' ``. However, it consequently also states that the singular of a noun matching `` `-[^ns]ses' `` matches `` `-[^ns]sis' ``.
  As a result, the singular of elementary plural nouns such as `` `houses' `` was deemed `` `housis' ``.

  The fix for this was proposed as change 8 in *Section 4.3: Changes for nouns*.

Note that the last two methods were only used to introduce large and general changes, and never to add rules specifically for single words that fail. There is an argument to be made that all discovered failing words should be added to the ruleset, so the final product is more often correct (at the cost of execution time). However, it would invalidate the test results from *Chapter 7: Results* and *Appendix A: Qualitative evaluation results*, as we would be overfitting on the `CELEX` lexicon.
As a result, this was *not* done.

The source code of our Python port of `Lingua::EN::Inflexion`, as well as the new `Inflexion` described in this thesis can be found on `https://github.com/tomaarsen/Inflexion`. As of the time of writing, the `master` branch holds the `Inflexion` source code, while the `original` branch stores the direct Python port of `Lingua::EN::Inflexion`.

4.1 General changes

1. Ensure that rules cannot be empty.
   The files generated by `Lingua::EN::Inflexion` have the following rules, among others:
   - `` ` => _' ``
   - `` `_ => could' ``

   In our `Inflexion`, changes were made so that these empty or otherwise nonsensical rules do not occur.

2. Many performance changes were implemented, such as:
   - Reducing the number of checks required for finding out whether a word is known to be of a certain form.

- Preventing calling `is_plural` twice with the same parameters when calling `to_plural`.
- Merging several regular expressions into one.
- If an intensive function has already been called with the regular form of a word, only call it again with the lowercase of that word if it wasn't lowercase to begin with.

3. Added custom casing in addition to Title case, UPPER case and lower case.
   - Any output that is only the letter `` `I' `` is always capitalised.
   - Any output starting with `` `mc' `` is converted to `` `Mc' `` plus the title case of the following word, e.g. `` `McInnes' ``.

4. Preservation of unusual whitespace before, after and between words.
   - For example, the plural of the noun collocation `` `  walk  of  life' `` is `` `  walks  of  life' ``.

4.2 Changes for verbs

1. Ensure that rules containing `` `*' `` need to match at least two characters in place of the `` `*' `` as opposed to just one. This helps with the following cases, among others:
   - `` `shear' `` no longer matches `` `*hear' ``.
   - `` `blend' `` no longer matches `` `*lend' ``.
   - `` `blight' `` no longer matches `` `*light' ``.
   - `` `stake' `` no longer matches `` `*take' ``.

2. Verb collocation support is implemented. Every time a verb is converted to a particular form in `Inflexion`, the following steps occur:
   (a) Get the subterm of the input:
      - Split the input into words by splitting on whitespace.
      - Split the first word into subwords by splitting on hyphens.
      - The last subword of the first word is the subterm.
   (b) Split off a prefix from the subterm, if one exists. Sample prefixes are `` `trans' ``, `` `inter' ``, `` `under' ``, `` `over' ``, `` `out' ``, `` `mis' ``, `` `un' ``, `` `in' ``, `` `re' `` and more.
   (c) Convert the remaining term to the desired form instead of the full input.
   (d) Then re-add the subwords split off in the previous two steps.

   Getting the first word of the input will ensure that this word receives the inflection, which is the desired behaviour:

- `work out' → `working out'
- `go through with' → `going through with'
- `look forward to' → `looking forward to'

Getting the last subword of the first word will ensure this section of a hyphenated input receives the inflection, which is the desired behaviour:

- `blow-dry' → `blow-drying'
- `over-eat' → `over-eating'
- `baby-sit him' → `baby-sitting him'

Splitting off the prefix and converting the remainder allows uncommon irregular verbs to be converted according to the irregular rules of a more common subverb:

- past of `uphold' → `up' + past of `hold' → `upheld'
- past of `foresee' → `fore' + past of `see' → `foresaw'
- past of `rewrite' → `re' + past of `write' → `rewrote'

3. Improved consonant duplication as used in converting to past, past participle and present participle. These rules are used as preprocessing before adding `ed' for past and past participle and `ing' for present participle:
   - Words ending in `fer' are always converted to `ferr'.
     - `prefer' → `preferred'
   - Words ending in `c' always converted to `ck'.
     - `mimic' → `mimicking'
   - Words ending in `en' always kept the same.
     - `listen' → `listened'
   - Words ending in a consonant plus `on' always kept the same.
     - `abandon' → `abandoned'
   - Words ending in a consonant plus a vowel plus an `l' always have the `l' duplicated according to the British English rules.
     - `barrel' → `barrelled'
     - `label' → `labelled'

   Furthermore, the consonant is duplicated whenever:

   - One of the following:
     - The word is just one syllable according to the `prosodic` [21] Python module.
     - The word is just one syllable according to the new `is_syllable` function, which counts the number of vowel clusters.

    - The last syllable of the word is stressed according to the `prosodic` [21] Python module.
  - And the word ends with roughly a consonant plus vowel plus consonant.
    - In particular, it must end with a consonant followed by a vowel or `` `y' ``, followed by a single character in `` `bcdlgkmnprstvz' ``.

4. Added rules for `` `to be' ``.
   - For example, previously the past tense of `` `be' `` was deemed `` `bed' ``, rather than `` `was' ``.

5. Fixed rule for `` `to bid' ``.
   - The past is now `` `*bid' `` instead of `` `*bade' ``, and the past participle is now `` `*bid' `` instead of `` `*bidden' ``.

6. Added `` `*' `` to `` `to bind' ``, `` `to breed' ``, `` `to ride' `` and `` `to run' ``, as there exist many words that end with these verbs that follow their irregular inflection rules.

7. Removed `` `*' `` from `` `to do' `` and `` `to let' ``, as there exist words that end with these verbs that follow different inflection rules.
   - e.g. `` `to torpedo' `` or `` `to fillet' ``

8. Added the rules for several verbs ending in `` `ie' ``. These verbs are otherwise converted to end in `` `iing' `` incorrectly:
   - `` `to underlie' ``, `` `to caddie' ``, `` `to hie' ``, `` `to outvie' `` and `` `to vie' ``

9. Added rule for `` `to arc' ``. This is an exception to the consonant doubling rule introduced stating that the past tense of a verb ending in `` `c' `` ends with `` `cked' ``

10. Added rules for these verbs ending in `` `s' ``, as these often had the final `` `s' `` removed when converting to plural, past, past participle or present participle:
    - `` `to bias' ``, `` `to bus' ``, `` `to caucus' ``, `` `to canvas' ``, `` `to chorus' ``, `` `to dis' ``, `` `to focus' ``, `` `to gas' ``, `` `to hocus' `` and `` `to wis' ``

11. Added rule for `` `to rent' ``. The word `` `rent' `` is the past and past participle of the verb `` `to rend' ``. As a result, the singular of `` `rent' `` would return `` `rends' ``, which is not the desired nor the expected behaviour.

12. Added the following general inflection rules:

Singular	Plural	Past	Present Participle	Past Participle
-ics	-ic	-icked	-icking	-icked
-Vzzes	-Vz	-Vzzed	-Vzzing	-Vzzed
-zes	-z	-zed	-zing	-zed
-ies	-ie	-ied	-ying	-ied

    Where V stands for any vowel. These rules match the following verbs that were previously improperly converted, among others:
    - `to panic', `to mimic', `to picnic' and `to traffic'.
    - `to quiz' and `to whiz'.
    - `to waltz'.
    - `to underlie', `to caddie', `to hie', `to outvie' and `to vie'.

    The latter three of these rules work in conjunction with similar rules that were already present in Lingua::EN::Inflexion.

13. Lastly, the present participle rule for
    - `-ues -ue -ued -ueing -ued'

    was changed from `-ueing' to `-uing' such that:
    - `argue' now converts to `arguing' instead of `argueing'.
    - `rescue' now converts to `rescuing' instead of `rescueing'.
    - `sue' now converts to `suing' instead of `sueing'.

4.3 Changes for nouns

1. Many duplicate rules were removed. For example, there were several equal rules for:
    - French imports like `bandeau', `beau' and `tableau'.
    - Nationalities such as `Congoese' or `Vermontese'.

2. Fixed rules for words that had separate rules for classical and modern plural forms. This caused `nucleus' and `mafioso' to always inflect according to the classical rules - even if the modern plural was requested.

3. Converted the rule `(SING)-(PREP) => (PL)-(PREP)' to `(SING)-(PREPR) => (PL)-(PREPR)'. The abbreviation `(PREPR)' stands for `prepositions reduced', and matches all prepositions *except* `out', `about', `off', `in', `on' and `over'. Noun collocations ending in these prepositions often apply the inflection on the preposition rather than on the word before:
    - `show-off' → `show-offs'
    - `voice-over' → `voice-overs'
    - `drive-in' → `drive-ins'

4. Increased precedence of the rules that require sub-nouns to be inflected, so that these rules are tried before other rules. Previously, `walk of life' would match `*life' before matching `(SING)-(PREP)-* => (PL)-(PREP)-*'. As such, the plural of `walk of life' was considered to be `walk of lives', while now it becomes the correct `walks of life'.

5. Added `tax => taxes' to avoid the singular of `taxes' returning the correct but unexpected Latin import `taxis'.

6. Fixed several typos in `nouns.lei', such as:
   - Replacing `hamadrayads' with `hamadryads'.
   - Removing `--ox => --oxen'.

7. Removed `*' from `*lux => lux' and `*louse => lice', as there exist words that end with these nouns that follow different inflection rules:
   - The plural of `flux' should be `fluxes', instead of `flux'.
   - The plural of `blouse' should be `blouses', instead of `blice'.

8. Added the following general inflection rules:

Rule #	Singular	Plural
1	-oux	-oux
2	-nge	-nges
3	-[^aeoui][aeouiy]che	-[^aeoui][aeouiy]ches
4	-[rnlpwaeio]se	-[rnlpwaeio]ses
5	-[aeo]use	-[aeo]uses

   These rules match the following types of nouns that were previously improperly converted, among others:
   1. French imports like `roux' that do not inflect.
   2. Nouns like `binge', `orange' and `hinge' used to convert to singular by following the classical inflection rule of `-[aiy]nges => -[aiy]nx' which works for e.g. `meninges' → `meninx'.
   3. Nouns like `aches', `pastiches', `cliches', `creches', `fiches' and `psyches' used to follow the rule `-[cs]hes => -[cs]h', incorrectly removing the final `e'.
   4. Nouns like `courses', `horses', `curses', `pulses', `collapses', `browses', `bases' and `closes' used to convert to singular by following the classical inflection rule of `-[^ns]sis => -[^ns]ses' which works for e.g. `analysis' → `analyses'.
   5. Nouns like `causes', `pause', `masseuses', `houses' and `blouses' used to convert to singular by following the classical

inflection rule of `-[^ns]sis => -[^ns]ses' which works for e.g. `analysis' → `analyses'.

Chapter 5

Evaluation

5.1 Qualitative evaluation procedure

With the creation of `Inflexion` in *Chapter 4: Inflexion* described, the focus shifts to the evaluation procedure developed to test its performance. We have developed the following qualitative testing framework that will be applied to several other Python modules that implement morphological analysis and generation, allowing weaknesses and strengths for these modules to be identified.

The results of this qualitative evaluation procedure applied on 10 different modules are placed in *Appendix A: Qualitative evaluation results*, while information on the other modules can be found in *Chapter 6: Existing Python modules*.

The qualitative evaluation procedure restricts itself to the correctness of the prediction. This is deemed more important than execution time. As a result, execution time is neither mentioned nor measured. An evaluation procedure that takes execution time into consideration is described in *Chapter 11: Future work*.

Each word or collocation converted via a morphological analysis and generation algorithm is classified according to one of the following categories:

1. `empty`
   The predicted output is *empty*.
2. `correct`
   The predicted output is *correct*.
3. `whitespace_error`
   The predicted output can be converted to a correct output by only changing the amount of *whitespace*.

4. `case_error`
   The predicted output can be converted to a correct output by only changing the *capitalisation*.
5. `dash_space_mismatch`
   The predicted output can be converted to a correct output by only changing *spaces into hyphens or vice versa*.
6. `collocations_mismatch`
   The predicted output has a *different number of words* than any correct output.
7. `quote_mismatch`
   The predicted output can be converted to a correct output by only changing *spaces into hyphens or vice versa*.
8. `collocations_wrong_term_changed`
   The *wrong word* in the collocation was modified in the conversion from the input to the prediction.
9. `suffix`
   The predicted output can be converted to a correct output by only *changing the suffix* of the prediction to a different suffix.
   These categories are listed as `` `$suffix1$_vs_$suffix2$' ``, where $suffix1$ is the suffix in the prediction, and $suffix2$ is the suffix in the correct output.
   - For example, given a prediction of `` `loafs' `` and a correct output of `` `loaves' ``, the evaluation procedure will produce `` `fs_vs_ves' ``.

Each of these categories are further elaborated upon in the algorithm procedure coming up next.
Using these categories, very specific weaknesses of a morphological analysis and generation algorithm can be discovered. This is invaluable information for developing, maintaining and improving such algorithms.

The algorithm classifying the errors of a chosen conversion takes following several parameters as inputs:

- `input`, the word or collocation passed to the morphological analysis and generation algorithm. This parameter is not strictly required as only the correctness of the prediction is looked at, but is sensible to have for logging purposes.
- `predicted`, the predicted output when passing `input` through the morphological analysis and generation algorithm that is being tested, for the chosen conversion.
- `outputs`, the set of known correct outputs when converting an `input` using the chosen conversion. Note that this set can be non-singular:

  - e.g. an `output` containing `` `placebos' `` and `` `placeboes' ``, the two accepted plural nouns for given the `input` of `` `placebo' ``.

  Optionally, this `outputs` may be ordered to give preference to the first output.

These parameters are used in the following algorithm:

1. *Classifying empty cases.*
   Some algorithms will return `None` or an empty string whenever they fail to perform the conversion. This results in `predicted` being an empty string or `None`, and we classify the conversion as `empty`.

2. *Classifying correct cases.*
   If `predicted` is in `outputs`, then we classify this case as `correct`.

3. *Classifying cases where only the whitespace di ers.*
   - e.g. `` `  word  of mouth  ' `` versus `` `word of mouth' ``.

   `predicted` is stripped of all whitespace before and after the word or collocation. Furthermore, every occurrence of two adjacent whitespace characters is replaced by a single character of whitespace. This operation is also applied on each output in `outputs`.
   If the modified `predicted` is in the modified `outputs` set, then we classify this case as `whitespace_error`. Otherwise, we undo the modifications made and proceed.

4. *Classifying cases where only the capitalisation di ers.*
   - e.g. `` `Anglo-american' `` versus `` `Anglo-American' ``.

   `predicted` is converted to lowercase. Each output in `outputs` is also converted to lowercase.
   If the modified `predicted` is in the modified `outputs` set, then we classify this case as `case_error`. Otherwise, we undo the modifications made.

5. *Classifying cases where the di erence between a hyphen and a space is the only di erence.*
   - e.g. `` `show off' `` versus `` `show-off' ``.

   If the number of dashes in `predicted` does not match with the number of dashes in any output in `outputs`, then we classify this case as `dash_space_mismatch`.

6. *Classifying cases with a di ering number of words.*
   - e.g. `` `sons-of-a-gun' `` versus `` `sons-of-guns' ``.
   - e.g. `` `brother' `` versus `` `brother-in-law' ``.

Replace all dashes in `predicted` and all values in `outputs` with spaces. If the number of spaces in `predicted` does not match with the number of dashes in any output in `outputs`, then we classify this case as `collocations_mismatch`.

- For each `output` in `outputs`:
  - For each word `pred_word` and `out_word` in `predicted` and `output`, respectively:

    7. *Classifying cases with a di ering quotes.*
       – e.g. `` `do's' `` versus `` `does' ``.

       If `pred_word` contains a quote, while `out_word` does not, or vice versa, then we classify this case as `quote_mismatch`.

       Otherwise, the difference in suffixes is extracted between `pred_word` and `out_word` by removing the overlapping parts of the two words:
       – `` `does' `` and `` `doths' `` → `` `es' `` and `` `ths' ``
       – `` `persons' `` and `` `people' `` → `` `rsons' `` and `` `ople' ``
       – `` `loafs' `` and `` `loaves' `` → `` `fs' `` and `` `ves' ``
       – `` `addendums' `` and `` `addenda' `` → `` `ums' `` and `` `a' ``
       – `` `mottos' `` and `` `mottoes' `` → `` `s' `` and `` `es' ``
       – `` `book' `` and `` `books' `` → `` `' `` and `` `s' ``

       These two extracted suffixes are entered in a tuple. So, every `output` is converted to a list of suffix tuples, one tuple per word. For example:
       – `predicted = `walk of lives'` and
       `output = `walks of life'` →
       `[(`', `s'), (`', `'), (`ves', `fe')]`

From now onwards, classifications will be made per `output`, based on the list of suffix tuples from the previous step. Each `output` has one such list, and the classification differs on which `output` is deemed best.

If `outputs` is ordered, then the next steps proceed with only the list of suffix tuples from the first `output`. Otherwise, an arbitrary `output` from `outputs` is picked, and its list of suffix tuples will be used in the next steps.

8. *Classifying cases where the wrong word in a collocation is converted.*
   • e.g. `` `walk of lives' `` versus `` `walks of life' ``.

   If the list of tuples of suffixes contains more than one non-empty tuple (i.e. more than one tuple that indicates an error), then we classify this case as `collocations_wrong_term_changed`.

9. *Classifying cases where the only di erence is one su x.*
   - e.g. `meat loafs' versus `meat loaves'.

   If this step is reached, then there is a list with just one non-empty tuple of suffixes, such as [(`', `'), (`fs', `ves')]. We classify this case as `$suffix1$_vs_$suffix2$' where $suffix1$ and $suffix2$ are taken from the one non-empty tuple. In the case of the previous example, this becomes `fs_vs_ves'.
   This classification means that the prediction ended with `fs', when it should have ended in `ves' instead.

   This way, similar conversion errors can be grouped together to identify commonly made mistakes for a specific morphological analysis and generation algorithm, for a particular conversion.

The outputs of this evaluation procedure applied to all tested modules can be found in *Appendix A: Qualitative evaluation results*.

5.2 Quantitative evaluation procedure

We also introduce a quantitative evaluation procedure using the accuracy measure, which gives a less detailed look at the correctness of a conversion. It does not give a notion of what kind of error was made, but merely shows the number of correct results relative to the total number of conversions.

For computing this accuracy, the input data for each of the conversions is further split up into two separate categories:

- *collocations*:
  This category represents all inputs that contain either a space (` ') or a hyphen (`-').
  - e.g. `baby-sit', `court martial', `blow-dry', `son of a gun' or `all-rounder'.
- *words*:
  This category represents all other inputs, i.e. inputs containing just one word.

Many of the modules tested in this thesis do not work well with collocations. If this partitioning was not used, these modules would end up with poor accuracy scores overall, even if they perform supremely for just words. In short, partitioning allows each module to display its true strength for the kind of inputs it was designed for.

5.3 Testing data

With the morphological analysis and generation module prepared and the evaluation procedures described, the next step is gathering the inputs and correct outputs to run the evaluation with. This is where the `CELEX` Lexical Database [11] comes in. `CELEX` is a conjoint initiative of five research institutes in the Netherlands, who have developed lexical databases to aid researchers.

These databases contain the plural, singular, past, past participle and present participle forms of well over 8,000 English verbs, as well as the plural and singular of over 20,000 nouns. The databases contain both words and collocations, among much more information.
`CELEX` mainly contains British English, but it should be noted that it is inconsistent at times with its consonant doubling [19], hinting at some American English influences.

The `CELEX` data is extracted using the WebCelex [22] web interface. The `Lexicon` option is selected, after which the `English Wordforms` option is chosen. From this point, the steps required to fetch the verb and noun data differ. Each of these two POS that are tested have a section explaining the method of fetching the data.

5.3.1 Verbs

When fetching data, WebCelex asks which information is desired. For verbs, the following items are selected in order:

- `Lemma Head` - The base form of the word.
- `Word` - The word itself.
- `FlectType` - The type of inflection applied on the word.
- `Lemma Class` - The POS, i.e. `` `N' `` for nouns and `` `V' `` for verbs.

Afterward, the following query is entered, ensuring that only verbs are returned:

- `Lemma Class eq "V"`

This returns an output starting with:

```
Lemma Head\Word\FlectType\Lemma Class
abandon\abandon\i\V
abandon\abandoned\a1S\V
abandon\abandoning\pe\V
abandon\abandons\e3S\V
abase\abase\i\V
abase\abased\a1S\V
abase\abases\e3S\V
abash\abash\i\V
abash\abashed\a1S\V
```

```
abash\abashes\e3S\V
abash\abashing\pe\V
abase\abasing\pe\V
...
```

Each line will be parsed such that there is:

- One data structure which converts from the *lemma head* to any desired *wordform*.
- One data structure which converts from a given *wordform* to its *lemma head*.

With these two data structures, any two wordforms can be mapped together to provide testing data. For example, if testing data is desired from past participle to singular, then:

- For each past participle extracted from `CELEX`, convert it to the lemma head using the appropriate data structure (wordform → lemma head).
- Convert this lemma head to singular using the appropriate data structure (lemma head → wordform).

This can be expanded to produce a mapping between any two wordforms. This will produce the `input` and `outputs` values, respectively, for the evaluation procedures from this chapter.

For parsing, the following `FlectType` values will be used:

- `` `i' `` and `` `eP' ``:
  These represent the infinitive and the plural, and the `Word` values will be stored as plurals in these cases. Note that the values for `` `i' `` and `` `eP' `` are identical for all cases except for the most irregular verb in the English language: `` `to be' ``. For this verb `CELEX` gives the `Word` `` `are' `` for `` `eP' `` and `` `be' `` for `` `i' ``.
- `` `e3S' `` and `` `e3Sr' ``:
  These represent the third person singular and the classical third person singular.
  - e.g. `` `have' `` and `` `hath' ``, respectively.
- `` `aP' `` and `` `aPr' ``:
  These represent the past tense and classical past tense.
  - e.g. `` `spoke' `` and `` `spake' `` are the past tense and classical past tense of `` `to speak' ``, respectively.
- `` `pa' `` and `` `par' ``:
  These represent the past participle and classical past participle.
  - e.g. `` `blended' `` and `` `blent' `` are the past participle and classical past participle of `` `to blend' ``, respectively.
- `` `pe' `` and `` `per' ``:
  These represent the present participle and classical present participle.

– e.g. `running' is the present participle of `to run'. There is no example of a classical present participle in CELEX.

This concludes the data fetching for verbs. The steps for nouns are similar, but have some extra steps for uncountable words. These steps will be described next.

5.3.2 Nouns

For nouns, the following items are selected in the web interface, in order:

- Lemma Head - The base form of the word.
- Word - The word itself.
- FlectType - The type of inflection applied on the word.
- Lemma Class - The POS, i.e. `N' for nouns and `V' for verbs.
- Lemma C_N - States whether the word is countable.

Afterward, the following query is entered, ensuring that only nouns are returned:

- Lemma Class eq "N"

This returns an output starting as:

```
1  Lemma Head\Word\FlectType\Lemma Class\Lemma C_N
2  a\a\S\N\Y
3  AA\AA\S\N\Y
4  AA\AAs\P\N\Y
5  abacus\abaci\P\N\Y
6  abacus\abacus\S\N\Y
7  abacus\abacuses\P\N\Y
8  abandon\abandon\S\N\N
9  abandonment\abandonment\S\N\N
10 abasement\abasement\S\N\N
11 abatement\abatement\S\N\N
12 abattoir\abattoir\S\N\Y
13 abattoir\abattoirs\P\N\Y
14 ...
```

Each line is parsed similarly to nouns, except with just two wordforms instead of five. Furthermore, if a line is uncountable (indicated by an `N' value for Lemma C_N) and the corresponding FlectType is `S', then the word and head will be identical. The head is then added to a set of uncountable nouns and noun collocations.

Note that some words are deemed both uncountable and countable, such as `hives', which exists as a word twice in CELEX:

```
1 Lemma Head\Word\FlectType\Lemma Class\Lemma C_N
2 hive\hives\P\N\Y
3 hives\hives\S\N\N
```

Line 2 states that `hives' is a plural form of the lemma `hive', and hence the lemma `hive' is a singular form of `hives'. It also says that

the lemma `hive' is countable.
Line 3 states that `hives' is a singular form of the lemma `hives', and shows that the lemma `hives' is uncountable.
As such, CELEX accepts `hives' and `hive' as the singular of `hives', through lines 3 and 2, respectively. This means that `hives' isn't exclusively uncountable. These entries that aren't exclusively uncountable will not be put in the set of uncountable nouns and noun collocations, as they are already present in the countable conversions.

For parsing, the following FlectType values will be used:

- `P' and `Pr':
  These represent the plural and the classical plural nouns, and the Word values will be stored as plurals in these cases.
  - e.g. `brothers' and `brethren', respectively.
- `S':
  This represents the singular nouns.

5.3.3 Preprocessing

The Word values from CELEX that represent words imported from foreign language often contain some speech indicators or accents, which will be removed as a preprocessing step. Some examples are:

```
1 Lemma Head\Word\FlectType\Lemma Class\Lemma C_N
2 applique\appliqu#e\S\N\0.00\Y
3 confrere\confr`ere\S\N\0.00\Y
4 debacle\d#eb^acle\S\N\0.30\Y
5 kummel\k"ummel\S\N\0.00\N
6 senor\se~nor\S\N\0.60\Y
7 curacao\cura,cao\S\N\0.00\N
8 ...
```

If value for Word contains `#', ` ', `^', `"', `∼' or `,', while the Lemma Head value does not contain this character, then this character will be stripped from Word.
As a result, the only difference between the Lemma Head and the Word is the conversion corresponding to the FlectType. After all, this conversion is the only feature that is being tested.

5.4 Testing

Testing is split between nouns and verbs. All nouns will be converted to singular and to plural. The CELEX testing data for nouns was partitioned between singular, plural and uncountable, and as a result the following conversions will be tested for nouns:

- Singular to singular.

- Singular to plural.
- Plural to singular.
- Plural to plural.
- Uncountable to singular.
- Uncountable to plural.

Note that for the last two conversions, a correct morphological analysis and generation algorithm should not modify these inputs.

Similarly, all verbs will be converted to the following verb forms: singular, plural, past, past participle and present participle. As with nouns, these conversions are made from each of the wordforms in the `CELEX` testing data for verbs. This results in 25 conversions: from each of the 5 wordforms to each of the 5 wordforms.

Adjective conversions are not tested, as there are only a fixed number of adjectives that change in the conversion from plural to singular or vice versa. As a result, these can easily be converted using a simple lookup dictionary, leaving the tests for adjective conversions uninteresting for this thesis.

Our exact Python port of Damian Conway's `Lingua::EN::Inflexion` as well as our `Inflexion` will be tested against 8 other competitive modules that have implemented functions for a subsection of all conversions. The modules are only given the input word and the desired wordform, meaning that the modules don't know which wordform the input word is.
For each module, the outputs of all of these conversions will be applied to both the procedure from *Section 5.1: Qualitative evaluation procedure* and the procedure from *Section 5.2: Quantitative evaluation procedure*.
The results of the qualitative evaluation procedure are displayed in *Appendix A: Qualitative evaluation results*, and the results of the quantitative evaluation are displayed in *Chapter 7: Results*.
Both of these results are elaborated on in *Chapter 8: Discussion*.

Chapter 6

Existing Python modules

With a morphological analysis and generation algorithm from *Chapter 4: Inflexion*, an evaluation procedure from *Section 5.1: Qualitative evaluation procedure* and testing data all ready, all that remains is preparing competing modules that provide morphological analysis and generation.

This chapter contains 8 commonly recommended modules, each of which implement some of the conversions that can be tested using the data fetched in *Section 5.3: Testing data*. Each conversion for each of the modules is implemented solely using functionality provided by that module, in a way that would likely give the best results. As a consequence, if a module has a preprocessing tool for an input that would likely improve the overall performance of a conversion, it will be used. This gives each module the best possible chance.

Figure 6.1 shows the relation between each of these modules. Every arrow from module x to module y states that module y is inspired by, a port of, or otherwise uses module x.
We have introduced the `Inflexion` module in this thesis.

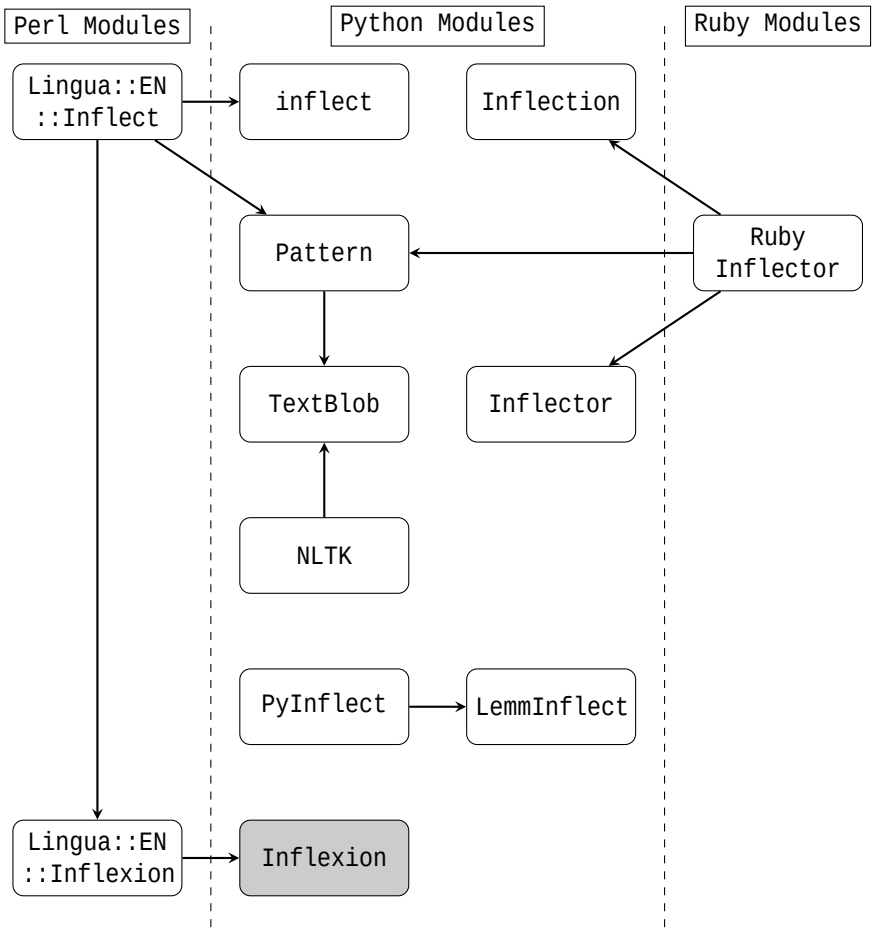


Figure 6.1: Relations between modules implementing morphological analysis and generation.

6.1 Functionality

Before diving into specifics of each of the modules, a broad overview is given of the functionality supported by each of these modules. Table 6.1 includes all conversions that are tested in this thesis, and marks for each module whether that conversion is supported.
Note that our module `Inflexion` as introduced in this thesis is underlined in all tables from now onwards.

Nouns, converting to ...	sing	plur
inflect	×	×
Inflection	×	×
Inflector	×	×
Lingua::EN::Inflexion	×	×
Inflexion	×	×
LemmInflect	×	×
NLTK	×	
Pattern	×	×
PyInflect	×	×
TextBlob	×	×

Verbs, converting to ...	sing	plur	past	past part	pres part
inflect		×			×
Inflection					
Inflector					
Lingua::EN::Inflexion	×	×	×	×	×
Inflexion	×	×	×	×	×
LemmInflect	×	×	×	×	×
NLTK		×			
Pattern	×	×	×	×	×
PyInflect	×	×	×	×	×
TextBlob		×			

Table 6.1: Python modules and their supported functionality, marked with a ×.

6.2 inflect

The `inflect` [3] module authored by Paul Dyson and maintained by Jason R. Coombs, is based on the Perl module `Lingua::EN::Inflect` described in *Section 3.3: Reputation*. It has been consistently improved upon since the creation of this module back in 2010. It has cemented its position as a solid choice for forming plural and singular nouns.

As `inflect` is based on the predecessor of `Lingua::EN::Inflexion`, it is not surprising that it boasts many functions that are also present in our port of `Inflexion`.

- `plural_noun(word, count=None)`
  This function takes a *singular* English noun or pronoun and returns its plural. Pronouns in the nominative (`` `I' `` → `` `we' ``) and accusative (`` `me' `` → `` `us' ``) cases are handled, as are possessive pronouns (`` `mine' `` → `` `ours' ``).

  That this function does not allow for the input to be a plural

English noun means that it is not recommended for converting any noun to plural. Nonetheless, this function will be used to convert any given noun to plural form, as the alternative of not testing these conversions is worse than performing poorly on them.

- `singular_noun(word, count=None)`
  This function takes a *plural* English noun or pronoun and returns its singular. Pronouns in the nominative (`"we" -> "I"`) and accusative (`"us" -> "me"`) cases are handled, as are possessive pronouns (`"ours" -> "mine"`). When third person singular pronouns are returned they take the neuter gender by default (`"they" -> "it"`), not `"they"-> "she"` nor `"they" -> "he"`.

  Note that this function can return `False` whenever the input word is deemed singular. In this case we simply return the input word.

- `plural_verb(word, count=None)`
  This function takes the *singular* form of a conjugated verb (that is, one which is already in the correct person) and returns the corresponding plural conjugation.

  As with `plural_noun`, this function likely is not a good option for converting any given verb to plural, as the documentation states it requires a singular verb. Still, this function will be used to convert any given verb to plural form.

- `present_participle(word)`
  This function takes the *singular* form of a conjugated verb and returns the corresponding present participle form.

  Again, `word` must already be singular, with the aforementioned consequences as a result.

6.3 Inflection

`Inflection` [4] is a direct port of the Ruby on Rails module `Inflector` [23]. The port is authored by Janne Vanhala, while the original module is by Bermi Ferrer Martinez. This original module, like `Lingua::EN::Inflect`, is the basis for several modules on this list. It is more simplistic, and does not work with verbs.

The exact functionality that overlaps with `Inflexion`'s is as follows, according to `Inflection`'s documentation:

- `pluralize(word)`
  Returns the plural form of a word.
- `singularize(word)`
  Returns the singular form of a word, the reverse of `pluralize()`.

The documentation of both of these functions is very broad with their definition of word, but judging by both the performance and the implementation used, it is clear that this is only designed for nouns. Testing these functions with verbs would do the otherwise respectable performance a disservice.

6.4 Inflector

The `Inflector` [5] Python module is unsurprisingly another port of the Ruby on Rails module of the same name by Bermi Ferrer Martinez. The author of this port is unknown, as the GitHub user who hosts the project states he took it from a forum post, and published it on GitHub and PyPI so developers may conveniently use it. It offers roughly the same functionality as `Inflection`, but has a slightly different implementation.

6.5 LemmInflect

Brad Jascob is the author of `LemmInflect` [6], the first Python module on this list that is not directly inspired by another developer their module. Instead, it is an improvement upon his own `PyInflect` [9], another respected module implementing morphological analysis and generation, of which the functionality will be described in *Section 6.8: PyInflect*.

`LemmInflect` boasts one relevant function:

- `getInflection(lemma, tag, inflect_oov=True)`
  The method returns the inflection for the given lemma based on the Penn Treebank [24] POS tag. It first does a dictionary lookup and use a rule-based system if the lookup fails, much like `Inflexion` and many other modules.
  The function returns a tuple with different spellings of the inflection. For testing, the first item from the tuple is taken.
  The function parameters are as follows:
  - lemma: the word to inflect
  - tag: the Penn Treebank POS tag
  - inflect_oov: if `False`, then the rule based inflections will not be used.

For testing, this function will be called with the following POS tags:

Used for	Tag	Tag meaning
noun, singular	`NN'	Noun, singular or mass
noun, plural	`NNS'	Noun, plural
verb, singular	`VBZ'	Verb, 3rd person singular present
verb, plural	`VB'	Verb, base form
verb, past	`VBD'	Verb, past tense
verb, past participle	`VBN'	Verb, past participle
verb, present participle	`VBG'	Verb, gerund or present participle

Table 6.2: Penn Treebank POS tags, their uses in our tests, and their meanings.

However, testing shows that `LemmInflect` performs considerably better when the input to `getInflection` is a lemma, i.e. a base form of a particular word. Luckily, `LemmInflect` also provides lemmatization. This lemmatization function will be used as a preprocessing step before calling `getInflection`, to give `LemmInflect` the best possible performance.

Note that a lemma for nouns is by definition singular, and a lemma for verbs is plural. So, for these two conversions we are only using the lemmatization functionality, rather than also calling `getInflection`.

6.6 NLTK

The Natural Language Toolkit, or `NLTK` [7], is one of the most commonly used Python modules for natural language processing. It implements a function described by Princeton University their WordNet [25] team. This WordNet is a large lexical database of the English language, which is the core reason why this function is useful.
The function itself, called `morphy` [26], uses WordNet to guarantee that whatever output it gives is a word that exists in the English language. However, as WordNet only contains lemmas, `morphy` can only convert to singular nouns and to plural verbs.

The documentation is as follows:

- `morphy(form, pos=None, check_exceptions=True)`
  Find a possible base form for the given form, with the given part of speech, by checking the list of exceptions in WordNet, and by recursively stripping affixes for this part of speech until a form in WordNet is found.

6.7 Pattern

This module is hosted by CLiPS [27], the Computational Linguistics and Psycholinguistics Research Center at the University of Antwerp, and authored by Tom de Smedt and Walter Daelemans. `Pattern` [8] is packaged with many useful features, including noun pluralisation, singularisation and verb conjugation:

- `pluralize(word, pos=NOUN, custom={}, classical=True)`
  Returns the plural form of the plural form of a singular noun. The `pos` parameter (part-of-speech) can be set to `NOUN` or `ADJECTIVE`, but only a small number of possessive adjectives inflect (e.g. my → our). The `` `custom' `` dictionary is for user-defined replacements.
  This function is adapted from Damian Conway's `Lingua::EN::Inflect` and its corresponding paper [28].

- `singularize(word, pos=NOUN, custom={})`
  Returns the singular form of a plural noun. The parameters work the same way as for `pluralize`.
  This function is adapted from Bermi Ferrer Martinez's `Inflector` module.

- ```
  conjugate(verb,
     tense = PRESENT,       # INFINITIVE, PRESENT,
                            # PAST or FUTURE
    person = 3,             # 1, 2, 3 or None
    number = SINGULAR,      # SG, PL
      mood = INDICATIVE,    # INDICATIVE, IMPERATIVE,
                            # CONDITIONAL or SUBJUNCTIVE
    aspect = IMPERFECTIVE,  # IMPERFECTIVE, PERFECTIVE or
                            # PROGRESSIVE
   negated = False,         # True or False
     parse = True)          # True or False
  ```
  `Pattern` has a lexicon of 8,500 common English verbs and their conjugated forms, which can be requested using this function. This functionality is based on XTAG [29] by the University of Pennsylvania. If a word is not in the lexicon from the aforementioned University, then a number of rules will be applied to generate the conjugated form.

6.8 PyInflect

As mentioned, `PyInflect` [9] is the predecessor of Brad Jascob's `LemmInflect` [6]. `PyInflect` is designed as an extension for `spaCy` [30], one of the largest open-source NLP modules. Like `LemmInflect`, it is based on the Automatically Generated Inflection Database (AGID) [31]. The AGID
  ```

data provides a list of inflections for various word lemmas.
This module can also be used standalone using functions with signatures identical to `LemmInflect`'s. However, as opposed to `LemmInflect`, these functions can return `None`.

Also note that no lemmatization is applied before calling these functions for `PyInflect`, as it does not implement lemmatization itself.

6.9 TextBlob

The final module on this list is `TextBlob` [10], a text processing module authored by Steven Loria. It uses both `NLTK` and `Pattern` for some of its functionality, and in particular adapted its pluralize and singularize methods from `Pattern`.
Though these functions claim to work on any word, it is clear they are only designed for nouns and adjectives, just like the original functions by `Pattern`.

It also implements lemmatization using the `morphy` from in `NLTK`, though implemented slightly differently than merely calling `morphy` from `NLTK` directly. This lemmatization is used for converting a verb to its plural form.

Chapter 7

Results

The results of this chapter are from the quantitative evaluation procedure described in *Section 5.2: Quantitative evaluation procedure*. The results from the qualitative evaluation procedure from *Section 5.1: Qualitative evaluation procedure* are placed in *Appendix A: Qualitative evaluation results*.

The tables in this chapter contain accuracy scores of the conversion from an input described by the table column to an output described by the section header. Each wordform in the header, e.g. singular, is further split up in:

- *words*:
  The conversion is applied on single words only.
- *colloc*: (short for collocation)
  The conversion is applied on collocations only, i.e. only on multiple words separated by a space or hyphen, such as:
  - `` `son of a gun' ``, `` `baby-sit' ``, `` `passer-by' `` or `` `mother in law' ``.

Many modules described in *Chapter 6: Existing Python modules* were not built for conversions on collocations. If the split was not used, these modules would end up with poor accuracy scores, even if they perform excellently for just words. As an additional consequence, users can base their module choice based on whether their data has collocations or not. See *Section 5.2: Quantitative evaluation procedure* for more information.

The accuracy scores in the results are based on the number of correct outputs divided by the number of inputs. In other words, when a module is unsure about a conversion and returns nothing, it is counted as incorrect.

The three best accuracy scores of every column are colored differing

greys, such that the darkest color is the best accuracy. Accuracy scores below 25% are not colored.

7.1 Nouns

Nouns are converted from singular, plural and uncountable to both singular and plural, for each module. This results in two types of conversions, to singular and to plural. See *Section 5.4: Testing* for more information.

7.1.1 Accuracy of converting nouns to singular

From / Modules	Singular		Plural		Uncountable	
	words	colloc	words	colloc	words	colloc
`inflect`	97.40%	97.94%	97.53%	96.90%	73.28%	82.70%
`Inflection`	98.13%	98.64%	95.78%	93.77%	90.18%	86.38%
`Inflector`	96.77%	97.71%	95.38%	93.64%	70.69%	81.30%
`Lingua::EN::Inflexion`	99.44%	99.32%	96.23%	94.33%	93.73%	87.62%
`Inflexion`	99.47%	99.34%	97.65%	96.29%	93.77%	87.84%
`LemmInflect`	99.39%	97.23%	97.43%	91.25%	95.35%	76.27%
`NLTK`[1]	93.60%	8.78%	90.52%	8.42%	79.43%	7.68%
`Pattern`	96.62%	97.35%	96.54%	94.55%	70.94%	80.22%
`PyInflect`[1]	96.55%	0.00%	1.78%	0.00%	86.71%	0.00%
`TextBlob`	96.61%	97.35%	96.69%	94.55%	70.96%	80.22%

7.1.2 Accuracy of converting nouns to plural

From / Modules	Singular		Plural		Uncountable	
	words	colloc	words	colloc	words	colloc
`inflect`	97.76%	96.91%	1.78%	0.94%	0.44%	0.54%
`Inflection`	97.08%	94.78%	98.53%	96.21%	9.51%	13.51%
`Inflector`	97.48%	94.69%	90.29%	91.18%	7.07%	13.57%
`Lingua::EN::Inflexion`	98.09%	95.76%	99.07%	98.53%	6.64%	12.59%
`Inflexion`	98.56%	97.41%	99.23%	98.39%	6.60%	12.43%
`LemmInflect`	95.33%	92.01%	95.56%	92.72%	46.07%	12.16%
`Pattern`	96.63%	94.15%	1.71%	0.78%	1.11%	3.08%
`PyInflect`[1]	95.46%	0.00%	1.61%	0.00%	0.78%	0.00%
`TextBlob`	96.75%	94.30%	1.73%	0.80%	1.07%	2.76%

[1]Returned no output for some inputs.

7.2 Verbs

For verbs the following five wordforms are considered: singular, plural, past, past participle and present participle. Conversions are made between each of these wordforms. This results in five types of conversions: to singular, to plural, to past, to past participle and to present participle.

7.2.1 Accuracy of converting verbs to singular

From / Modules	Singular		Plural		Past		Past Part.		Present Part.	
	words	colloc	words	colloc	words	colloc	words	colloc	words	colloc
`Lingua::EN::Inflexion`	99.95%	5.44%	99.50%	4.58%	0.87%	0.08%	0.93%	0.08%	0.00%	0.00%
`Inflexion`	99.95%	99.84%	99.73%	99.64%	0.87%	4.10%	0.93%	6.51%	0.00%	0.00%
`LemmInflect`	99.57%	5.36%	99.16%	4.46%	98.16%	3.91%	98.17%	3.87%	99.00%	4.38%
`Pattern`	98.97%	5.56%	98.83%	4.34%	96.89%	3.63%	97.19%	3.60%	97.94%	3.99%
`PyInflect`[1]	0.03%	0.00%	99.11%	0.00%	0.90%	0.00%	0.98%	0.00%	0.00%	0.00%

[1] Returned no output for some inputs.

7.2.2 Accuracy of converting verbs to plural

From / Modules	Singular		Plural		Past		Past Part.		Present Part.	
	words	colloc	words	colloc	words	colloc	words	colloc	words	colloc
`inflect`	99.36%	8.44%	99.98%	100.0%	0.87%	4.10%	0.93%	6.51%	0.00%	0.00%
`Lingua::EN::Inflexion`	99.47%	4.50%	99.81%	99.21%	0.87%	4.02%	0.93%	6.39%	0.00%	0.00%
`Inflexion`	99.59%	99.01%	99.91%	99.92%	0.87%	4.10%	0.93%	6.51%	0.00%	0.00%
`LemmInflect`	99.67%	4.38%	99.45%	97.40%	98.30%	7.66%	98.25%	9.88%	99.09%	4.30%
`NLTK`[12]	97.80%	2.21%	97.83%	2.25%	96.62%	2.01%	96.83%	1.99%	97.33%	2.13%
`Pattern`	99.19%	4.42%	99.74%	98.74%	97.23%	7.46%	97.53%	9.72%	98.21%	3.95%
`PyInflect`[1]	0.09%	0.00%	99.66%	0.00%	0.87%	0.00%	0.93%	0.00%	0.00%	0.00%
`TextBlob`	96.91%	2.21%	99.88%	99.96%	95.89%	6.03%	96.10%	8.42%	96.49%	2.13%

7.2.3 Accuracy of converting verbs to past

From / Modules	Singular		Plural		Past		Past Part.		Present Part.	
	words	colloc	words	colloc	words	colloc	words	colloc	words	colloc
`Lingua::EN::Inflexion`	94.45%	4.22%	94.22%	4.14%	98.61%	4.37%	98.02%	4.25%	3.10%	0.00%
`Inflexion`	98.01%	98.38%	97.87%	98.38%	99.17%	98.22%	98.64%	95.98%	4.06%	23.41%
`LemmInflect`	97.87%	4.02%	97.59%	4.03%	97.91%	3.98%	97.85%	3.94%	97.92%	4.07%
`Pattern`	98.01%	3.71%	97.92%	3.79%	98.00%	3.75%	98.19%	3.71%	98.23%	3.71%
`PyInflect`[1]	0.03%	0.00%	97.95%	0.00%	0.88%	0.00%	0.95%	0.00%	0.00%	0.00%

[1] Returned no output for some inputs.

[2] 99.96%, 100.00%, 99.37%, 99.50% and 99.74% if empty outputs are ignored when converting words to plural from singular, plural, past, past participle and present participle, respectively.

7.2.4 Accuracy of converting verbs to past participle

From / Modules	Singular		Plural		Past		Past Part.		Present Part.	
	words	colloc	words	colloc	words	colloc	words	colloc	words	colloc
`Lingua::EN::Inflexion`	94.40%	4.14%	94.24%	4.14%	98.15%	4.29%	98.64%	4.33%	3.10%	0.00%
`Inflexion`	97.95%	98.42%	97.90%	98.42%	98.60%	94.35%	99.19%	99.27%	4.03%	23.45%
`LemmInflect`	97.89%	3.98%	97.66%	3.99%	97.89%	3.94%	97.95%	3.91%	97.97%	4.03%
`Pattern`	98.01%	3.94%	97.92%	3.95%	97.57%	4.02%	97.92%	3.98%	98.14%	3.99%
`PyInflect`[1]	0.03%	0.00%	98.16%	0.00%	0.87%	0.00%	0.98%	0.00%	0.00%	0.00%

7.2.5 Accuracy of converting verbs to present participle

From / Modules	Singular		Plural		Past		Past Part.		Present Part.	
	words	colloc	words	colloc	words	colloc	words	colloc	words	colloc
`inflect`	93.92%	4.11%	93.86%	4.34%	0.87%	0.08%	0.93%	0.08%	0.00%	0.00%
`Lingua::EN::Inflexion`	95.26%	4.38%	95.01%	4.42%	3.75%	0.08%	3.88%	0.08%	100.0%	4.66%
`Inflexion`	98.66%	99.01%	98.62%	98.97%	4.60%	24.57%	4.81%	26.90%	99.95%	99.68%
`LemmInflect`	98.23%	4.22%	97.90%	4.23%	97.79%	3.83%	97.75%	3.79%	98.30%	4.34%
`Pattern`	98.54%	4.42%	98.45%	4.34%	97.74%	3.99%	98.00%	3.95%	98.76%	4.38%
`PyInflect`[1]	0.03%	0.00%	98.16%	0.00%	0.90%	0.00%	1.00%	0.00%	0.00%	0.00%

[1] Returned no output for some inputs.

Chapter 8

Discussion

This chapter will explain why the results are as they are, and give recommendations based on our personal judgment of the results as shown in *Chapter 7: Results* and *Appendix A: Qualitative evaluation results*.

This chapter has the same structure as *Chapter 7: Results*. Each table from that chapter is discussed using the following sections:

- First impressions.
  - What is our judgment of the results from both *Chapter 7: Results* and the data in *Appendix A: Qualitative evaluation results*?
- `Inflexion` versus `Lingua::EN::Inflexion`.
  - Have our changes actually improved upon Damian Conway's `Lingua::EN::Inflexion`?
- Recommendation.
  - What module should be used for each conversion, depending on the characteristics of the inputs?

Data from both *Chapter 7: Results* and *Appendix A: Qualitative evaluation results* will be used to back up the recommendations and judgments made. After discussing each table individually, a general judgment on the evaluation procedure and the modules implementing morphological analysis and generation is given in *Section 8.3: Our overall judgment*, which helps answering the two research sub-questions.

8.1 Nouns

8.1.1 Judgment of converting nouns to singular

- *First impressions:*
  At first glance it is clear that `Inflexion` dominates the conversion to singular.

`inflect` outperforms in particular for plural to noun, but falls far behind in the other categories. On average, the conversion from singular to singular is performed very well. This is not particularly surprising, as many of these modules will default to not changing anything when converting to singular, and no changes need to be made for singular to singular.

This default behaviour explains the strikingly good performance for uncountable inputs to singular, as compared to the performance for uncountable inputs to plural. For this conversion, a common default behaviour is adding a `` `s' ``, e.g. in `` `book' `` to `` `books' ``.

It can be noted that many modules support noun collocations. This is a different story for verbs.

Because `Inflexion` performs well for any input noun, a user can use it to convert any unknown noun to singular and be fairly confident that the result is a correct singular noun.

- `Inflexion` *versus* `Lingua::EN::Inflexion`:
  Our program `Inflexion` outperforms the module it was based on, `Lingua::EN::Inflexion`, in every single conversion, both for words and collocations. As can be seen in *Section A.1.1: Singular to singular*, the cases where the results differ are classified by the evaluation procedure to be of the class `` `incorrect suffix used' ``.

  The corresponding suffix table shows that the only difference is that `Lingua::EN::Inflexion` has 6 extra errors (out of over 20 thousand test cases) for the category of `` `ouse_vs_ice' ``.

  This means that `Lingua::EN::Inflexion` converted a word into a form that ended with `` `ouse' ``, while the real singular ends with `` `ice' ``. This is a direct consequence of change 7 from *Section 4.3: Changes for nouns*, which replaces the rule `` `*louse => lice' ``.

- *Recommendation*:
  Regardless of whether your data contains just words or also collocations, our `Inflexion` is likely to outperform all other modules for this conversion.

  We strongly recommend the use of `Inflexion` for all conversions of nouns to singular.

8.1.2 Judgment of converting nouns to plural

- *First impressions:*
  The accuracy when converting from uncountable nouns to plural is strikingly low for almost all modules, with the exception of `LemmInflect`, which stands shoulders above the rest. For the remaining columns, `Inflexion` and even `Lingua::EN::Inflexion` dominate. The `inflect` and `Inflection` modules get some third places and a second, but on average do not compete. Most modules make at least 1.5 times as many errors as `Inflexion`.

  Like with converting to singular, `PyInflect` only performs well when the input is already of the base form of the noun POS: singular. This is in stark contrast to its successor `LemmInflect`, which outperforms it greatly.
  However, despite the good performance for uncountable to plural, `LemmInflect` still performs poorly on average as compared to the top modules for this conversion.

  The difference in accuracy between `Inflexion` and the other modules can be marked up to how `Inflexion` makes considerably less errors in the `suffix` category, as can be seen in *Section A.1.4: Singular to plural*. The suffix table shows that failing to append an `` `s' `` is an issue for `Inflection` and `LemmInflect`, who claim 107 and 364 of these errors, respectively. These values are massive as compared to `Inflexion`'s 28. Other struggles are using the `` `es' `` prefix as opposed to just `` `s' ``, and adding an `` `s' `` where there should not be one. `Inflexion` performs well in all of these categories.

- `Inflexion` *versus* `Lingua::EN::Inflexion`:
  Unlike with converting to singular, our `Inflexion` does not exclusively improve upon `Lingua::EN::Inflexion`. Though beating the latter by 0.49, 1.65 and 0.16 percentage points, it loses to `Lingua::EN::Inflexion` by 0.14, 0.04 and 0.16 percentage points on the rightmost three columns. However, the conversion from singular to plural is the most important, as it will be most commonly used in practice. This is where `Inflexion` beats its predecessor handily. Furthermore, the performance gains are much larger than the losses. So, `Inflexion` is still a significant improvement upon `Lingua::EN::Inflexion`.

  For the conversion from singular to plural, `Inflexion` has reduced the number of errors classified as `wrong_term_changed` (which only occur for collocations) by half. It has also reduced the number of suffix errors

by roughly 25% relative to `Lingua::EN::Inflexion`. The reduction in performance for plural to plural collocations is due to one new error classified as `case`. Similarly marginal changes for some suffix are the cause of the slight lack in performance for uncountable to plural nouns.

In the end, `Inflexion` improves upon `Lingua::EN::Inflexion` for this conversion.

- *Recommendation*:
  Due to `LemmInflect`'s disappointing performance on singular to plural and plural to plural, it should not be used despite the good accuracy for uncountable to plural.
  Once again we strongly recommend the use of `Inflexion` for all conversions of nouns to plural.

8.2 Verbs

8.2.1 Judgment of converting verbs to singular

- *First impressions:*
  The five modules implementing this conversion are all impressive, with `PyInflect` falling behind. It is immediately noticeable that both `Inflexion` and `Lingua::EN::Inflexion` only perform when the input is either a singular or a plural. Furthermore, it is striking that only `Inflexion` has the ability to generate outputs for verb collocations such as `` `box in' `` accurately.

  For past tense, past participle and present participle only `LemmInflect` and `Pattern` compete. `LemmInflect` has the great advantage of being able to use its formidable lemmatization function as a preprocessing step for the input. This allows it to first convert these verbs from past tense, past participle and present participle back to plural. After doing so, it can apply its morphological generation algorithm.

  This is where `LemmInflect` and `Pattern` shine, despite performing worse than `Inflexion` and `Lingua::EN::Inflexion` for converting from singular and plural.

  According to Appendix sections A.2.2 to A.2.5, the collocation errors for all modules except `Inflexion` are primarily a result of the wrong term being changed, e.g. `` `box ins' `` instead of `` `boxes in' ``.

- `Inflexion` *versus* `Lingua::EN::Inflexion`:
  There is a noticeable difference in collocation accuracy between our `Inflexion` and `Lingua::EN::Inflexion`, directly as a result of

change 2 described in *Section 4.2: Changes for verbs*. This jump from 5.44% to 99.84% and 4.58% to 99.64% are very noteworthy improvements.

For words, there is only one conversion where a change is noticeable, and that is the most useful one: converting plural to singular.
For this conversion, roughly half of the errors were removed, moving the accuracy from an already immaculate 99.50% to the even better 99.73%. *Section A.2.2: Plural to singular* shows us that this difference is likely in the reduction of the 49 suffix errors to only 19 suffix errors, out of some 8,300 test cases. This is mostly a result of the 27 times `Lingua::EN::Inflexion` fails to append an `s', while `Inflexion` has only failed to do so twice.

- *Recommendation*:
  If the inputs to the morphological analysis and generation algorithm contains words that are in past tense, past participle or present participle, use `LemmInflect` for the conversion to singular.

  Otherwise, if the inputs are all words or collocations are plural, or if the inputs contain exclusively singulars and plurals, use `Inflexion`.

  Lastly, there is no module that supports collocations in past tense, past participle or present participle. Our recommendation is to split up these words, identify which word should be changed in morphological generation, and use `LemmInflect` to convert that word.

8.2.2 Judgment of converting verbs to plural

- *First impressions:*
  `TextBlob` and `inflect`, two modules that have not been very present thus far, are suddenly top performers for plural to plural. However, this conversion is not particularly noteworthy, as no changes need to be made to the input when converting from plural to plural.
  As these two modules fail to perform well relative to the other modules for the different conversions, neither is a good general pick for converting to plural.

  `Inflexion` continues to perform very well from singular and plural, and like `Lingua::EN::Inflexion` fails on conversions from past tense, past participle and present participle.

  `LemmInflect` is very formidable when converting from singular

to plural. This is a result of its respectable lemmatization function, which helps it to take first place for a conversion from singular for the first time and last time.
In addition to taking the first place here, it also outperforms all modules by a large margin for converting from past, past participle and present participle.
Despite placing near last for plural to plural, its accuracy of 99.45% is still good, and it makes this module a solid choice for this conversion.

As can be seen when looking at Appendix A.2.6 to A.2.10, the `NLTK` module often returns nothing, but makes very little mistakes when it does return an output. As the footnote under *Section 7.2.2: Accuracy of converting verbs to plural* states, the accuracies are 99.96%, 100.00%, 99.37%, 99.50% and 99.74% if empty outputs are ignored when converting words to plural from singular, plural, past, past participle and present participle, respectively. These accuracy scores are considerably higher than the otherwise best scores. Because of this, `NLTK` can be used for converting to plural with a high level of trust that if a word is returned, it will be the correct plural. If nothing is returned, the second best option of `LemmInflect` can be used.

- `Inflexion` *versus* `Lingua::EN::Inflexion`:
  Once again, there is a striking difference in the accuracy of converting singular verb collocations to plural. Beyond that, our `Inflexion` either performs equally or better than `Lingua::EN::Inflexion`.

  Converting words from singular to plural has seen a small improvement, and so has the detection of whether a word is already plural. According to *Section A.2.7: Plural to plural*, the number of suffix errors has been reduced from 31 to 7 for this conversion, out of over 8,300 test cases.
  This difference is caused by how `Lingua::EN::Inflexion` incorrectly removes an `` `s' `` 31 times, while `Inflexion` only makes this mistake 7 times.

- *Recommendation*:
  If the inputs contain collocations, and are only singular or plural, use `Inflexion`.

  If the inputs are only or mostly words, first use `NLTK`'s conversion. In the event that nothing is returned, use `LemmInflect` instead.

  If the inputs contain collocations, and are not only singular or

plural, split up the collocation into words and pass the verb through `NLTK` and `LemmInflect` like described previously.

8.2.3 Judgment of converting verbs to past

- *First impressions:*
  The pattern that is visible when converting to singular and to plural is not as present here. That is, `Inflexion` and `Lingua::EN::Inflexion` don't exclusively perform well when converting from plural or singular, but also when converting from past tense and past participle. `PyInflect` is placed in the top three for a conversion for the first time, and immediately takes first place for plural to past.
  Furthermore, `LemmInflect` is not as dominating as it is when converting to singular or plural.

  The highest accuracy from both past and past participle can be explained by how both `Lingua::EN::Inflexion` and `Inflexion` have a system that attempts to check whether a word is already in the right form. As the past and past participle form of a verb only differ if the verb is irregular, a word that is in past participle is often identified to also be in past tense.

  However, like when converting from e.g. past tense to singular, `Inflexion` and `Lingua::EN::Inflexion` do not remove suffixes before converting. As a result, for present participle to past, it will incorrectly convert e.g. `` `working' `` to `` `workinged' ``.

  The `LemmInflect` and `Pattern` modules do not have this limitation, and as a result have more consistent results. Especially the latter module performs quite well.

- `Inflexion` *versus* `Lingua::EN::Inflexion`:
  Our `Inflexion` is an immense improvement upon `Lingua::EN::Inflexion` for converting to past tense. The accuracy of collocation conversions of all forms except present participle to past tense has been increased from roughly 4% to 95-98%.

  Furthermore the accuracy of conversion from singular and plural to past were improved from roughly 94% to roughly 98%. According to *Section A.2.11: Singular to past*, this is a consequence of how the number of suffix errors made were more than halved. This is a result of the improvement of the consonant duplication algorithm described as change 3 in *Section 4.2: Changes for verbs*.

Even converting from past and past participle has improved by roughly 0.6 percentage points each.

- *Recommendation*:
  If the inputs are exclusively words of the base form of plural, use `PyInflect`. Otherwise, if the inputs contain collocations, or if the inputs are known to have few or no present participle forms, then `Inflexion` should be used.
  Alternatively, if there are present participle forms, `` `Pattern' `` should be used for more consistent results.

8.2.4 Judgment of converting verbs to past participle

- *First impressions:*
  The results look very similar to the conversion of to past. This is not particularly surprising as the past tense and past participle only differ if the verb is irregular, e.g. `` `flew' `` and `` `flown' ``.

  One interesting difference is how the balance between `LemmInflect` and `Pattern` has shifted slightly. Rather than `Pattern` beating `LemmInflect` outright for all words, the latter is outperforming the former for two out of five word forms.

- `Inflexion` *versus* `Lingua::EN::Inflexion`:
  Almost identically to the previous conversion, our `Inflexion` knocks `Lingua::EN::Inflexion` out of the water.

- *Recommendation*:
  When the inputs are only plural words, use `PyInflect` for the best results. Otherwise, if the inputs contain collocations, or if there are few or no present participle forms among the inputs, then `Inflexion` ought to be used.
  If there are present participle forms, or if more consistent results are desired, `Pattern` or alternatively `LemmInflect` should be used.

8.2.5 Judgment of converting verbs to present participle

- *First impressions:*
  The accuracy for present participle to present participle at 100% for `Lingua::EN::Inflexion` is quite surprising. Furthermore, `Inflexion` once again takes the first place performance for the conversions from singular and plural. `Pattern` and `LemmInflect` are once again very similar in performance, but the former definitely performs better on average.

Unlike the previous two conversions, `PyInflect` is no longer the best choice. It is also clear that `inflect` is by far the worst performing module for this conversion.

Once again, `Pattern` is the best option when consistency is desired, while `Inflexion` outperforms `Pattern` for the cases that it supports, but does horribly for the two that it does not.

- `Inflexion` *versus* `Lingua::EN::Inflexion`:
  For the conversions from singular and plural, our `Inflexion` greatly outperforms `Lingua::EN::Inflexion`, reducing the percentage of incorrect conversions from roughly 5% to roughly 1.4% on average.
  The difference in results for collocations are even more striking, improving the percentage of correct conversions from roughly 4.4% to roughly 90%.

  Converting words from present participle to present participle has not seen a noticeable change in accuracy, while the collocation accuracy is improved heavily.

  Neither `Lingua::EN::Inflexion` nor `Inflexion` work well when converting from past or from past participle to present participle. However, `Inflexion` has improved an impressive 25 percentage points on `Lingua::EN::Inflexion` for collocations of these conversions.

- *Recommendation*:
  If the inputs contain collocations or do not contain any past tense or past participle forms, then `Inflexion` will have the best results. However, `Pattern` should be used whenever the inputs do contain past tense or past participle words, for better consistency.

8.3 Our overall judgment

Our `Inflexion` consistently performs as or among the best, and is the uncontested best option for converting nouns. For verbs it has great performance, but gives up some consistency by only performing wonderfully for several wordforms.
The `Pattern` module with its `conjugate` function and the `LemmInflect` module with its `getLemma` and `getInflections` functions both give more consistent results for verbs, but never outperform our `Inflexion` on the conversions that both modules are good at.

Our `Inflexion` module is a top contender for a morphological analysis and generation algorithm, especially for nouns, and also for verbs. It

consistently outperforms its predecessor `Lingua::EN::Inflexion`, and shines due to its low number of suffix errors, as can be determined from the results of the tests using our evaluation procedure in *Appendix A: Qualitative evaluation results*.

Though the evaluation procedure from *Section 5.1: Qualitative evaluation procedure* is enormously helpful in identifying exactly for which types of inputs certain modules perform better or worse, most users will only care whether a conversion is executed correctly or not. After all, the noun `` `book' `` being converted to plural incorrectly and resulting in `` `bookes' `` is not in any way better or worse than being incorrectly converted to `` `bookis' ``.
As a result, this evaluation procedure is wonderful for comparing and contrasting the intricacies of morphological analysis and generation algorithms, but is not more useful than merely determining whether an output is correct or not when it comes to determining which module is *better*.
For determining which module is *better*, the secondary evaluation procedure described in *Section 5.2: Quantitative evaluation procedure* is more useful.

In short, the qualitative evaluation procedure is invaluable for developing and testing a morphological analysis and generation algorithm, while the quantitative evaluation procedure works wonders for comparing the raw performance of such algorithms.

Chapter 9

Related work

This chapter contains some of the key papers that helped guide us in creating this thesis.
Beyond that, the 8 Python modules listed in *Chapter 6: Existing Python modules* may be considered as related work as well. We opted to explain them earlier on instead, as they help explain the results in *Chapter 7: Results*.

9.1 Conway (1998)

Damian Conway his paper [28] is the basis of several Python modules: *Section 6.2: in ect*, *Section 6.7: Pattern* and *Section 6.9: TextBlob*. It introduces a pluralizing algorithm that is used in `Lingua::EN::Inflect` [17]. Conway has adapted and improved upon this work to create its successor `Lingua::EN::Inflexion` [2], but the core idea remains.

There is not a large difference in supported functionality between the two modules by Conway. As such, it would be interesting to apply the evaluation procedures introduced in *Section 5.1: Qualitative evaluation procedure* on `Lingua::EN::Inflect` as well, so there is more robust evidence that `Lingua::EN::Inflexion` indeed outperforms its predecessor `Lingua::EN::Inflect`.
Currently, the only evidence of that is Damian Conway's word [17].

This paper [28] introduces several rule-based algorithms. As opposed to models that require learning, these rule-based algorithms do not overfit, as long as the rules are kept general. These general rules based on grammatical rules attempt to create a pluralisation scheme that even works on words that do not exist (yet). As language continues to expand, this rule-based system might work better on new words than models that are

taught on existing language.

9.2 Minnen et al. (2000), Minnen et al. (2001)

Minnen, Carroll and Pearce wrote two papers [15, 19] on their tools `morpha` and `morphg`, a morphological analyser and generator, respectively. As opposed to any other module in this paper, these tools allow the user to switch between British English and American English. It does so by controlling the behaviour with respect to consonant doubling. This could definitely be adapted into `Inflexion`, but testing the performance of this requires splitting the testing data up into American English and British English.

The morphological generator is derived automatically from the morphological analyser. On the other hand, the morphological analyser is derived using morphological knowledge acquired semi-automatically from large corpora and dictionaries.
Neither system uses a lexicon for conversion, but rather roughly 1,400 rules ranging from very general to very specific, including rules for irregular words.

As with Conway's work [2, 17, 28] and our adaptation `Inflexion`, this system is likely to contain a rule to correctly handle words that have not previously been encountered. Further similarities include that all of these systems allow multiple accepted forms for a conversion, e.g. both `` `mangoes' `` and `` `mangos' `` are accepted plurals of the noun `` `mango' ``.

`morpha` and `morphg` are also tested with `CELEX` data, but these models were finetuned specifically to this testing set, making the boasted accuracy of roughly 99.95% nothing more than a fabrication. What kind of accuracy can be expected on other data sets is the question.

Though both `morhpa` and `morphg` are still technically freely available, any user wishing to access the source code is advised to look via an archiving website, as all links to the source code in the papers are no longer accessible.

9.3 Van den Bosch et al. (1999)

The paper [12] by Van den Bosch and Daelemans constricts itself solely to morphological analysis using memory-based learning. This is supervised machine learning that learns by storing examples of a task in memory. When new examples are presented, it searches for the best-matching example in memory to help give an appropriate result. With enough examples in

memory, a memory-based learning algorithm can learn classifications.

The model in this paper [12] works on chunks of words of at most 11 characters, and has reformulated the problem into a classification problem. It does perform noticeably worse on unseen words, hinting at overfitting or weaknesses. This paper veered us away from using machine learning on letter chunks to classify words or subwords. Though it would be useful to be able to segment words correctly in order to apply the morphological generation step on only the relevant subword, the performance here does not seem sufficient to be worth looking further into.

However, this paper and several others that were glanced over before starting with this research used `CELEX` as a source for testing and training data. This caused us to look in that direction for a source of our testing data.

It should be noted that Daelemans is also the author of the *Section 6.7: Pattern* Python module.

9.4 Heemskerk (1993)

Heemskerk's paper [32] also introduces a method for morphological analysis with a MORphological PArser (MORPA), which uses a lexicon to segment words into components. Many words can be segmented into multiple seemingly valid chunks, e.g.:

- `` `repaired' `` → `` `re' `` + `` `pair' `` + `` `ed' `` versus `` `rep' `` + `` `air' `` + `` `ed' ``

In such cases, the former segmentation is more logical. What MORPA does is add a probability through a probabilistic context-free grammar to filter out unlikely segmentations. Though MORPA was developed for Dutch, which requires more segmentation than the English language [32], a similar method could prove useful for segmentation and classification of morphemes. Then, the morphological generation step could be applied only on the relevant morphemes.
This acted as our backup in case porting and adapting the existing `Lingua::EN::Inflexion` Perl module proved unfruitful.

Like the previous paper by Van den Bosch et al. (1999) [12], MORPA was tested on data from `CELEX`, further guiding us in that direction.

9.5 Bloch (1947)

The paper of Bloch [20] focuses on linguistics as opposed to morphological analysis or generation. The paper explains inflection rules and mentions

potential suffixes and inflectional categories. Bloch gives information on the verb `be', and most noticeably provides a thought to be complete list of 200 irregular verbs in standard colloquial English, together with several classical verbs. This list was of great importance in determining whether section of rules for irregular verbs in `Lingua::EN::Inflexion` is lacking or not.

Chapter 10

Conclusions

RQ 1 As elaborated on in *Section 8.3: Our overall judgment*, our newly developed evaluation procedure described in *Section 5.1: Qualitative evaluation procedure* works remarkably well for comparing and contrasting morphological analysis and generation algorithms. However, our procedure from *Section 5.2: Quantitative evaluation procedure* is more useful for determining which algorithm performs the *best*.

RQ 2 Our module `Inflexion` which implements a morphological analysis and generation algorithm as described in *Chapter 4: In exion* consistently outperforms its predecessor `Lingua::EN::Inflexion` by Damian Conway, according to both of our evaluation procedures.

Beyond merely outperforming `Lingua::EN::Inflexion`, our module `Inflexion` is consistently at the top of modules for converting nouns and noun collocations from any form to any form. According to our judgment on the evaluation of our morphological analysis and generation algorithms of British English from *Section 8.1: Nouns*, `Inflexion` should be used over any other module for all noun conversions.

For verbs, our `Inflexion` is also dominant, but lacks consistency. As elaborated on in *Section 8.2: Verbs*, the recommended module for applying morphological analysis and generation on verbs depends on several factors. These include which wordform is being converted to, as well as the characteristics of the input data that will be used in this conversion.

Though our judgment often favors `Inflexion`, other modules like `LemmInflect` and `Pattern` are also frequently recommended. `NLTK` and `PyInflect` also perform best in some rare situations.

RQ In conclusion, our implementation of a morphological analysis and generation algorithm of British English in our `Inflexion` module often beats the competing modules, and should be considered as a very respectable and competitive option.

Chapter 11

Future work

This chapter is split up in three distinct categories:

- Future research
  This section mentions some new research that can be done to expand upon the work in this thesis.
- Evaluation
  This section enumerates possible improvements or additions to the quantitative and/or the qualitative evaluation procedures from this thesis.
- `Inflexion`
  This section enumerates possible improvements to the `Inflexion` Python module to improve its effectiveness and further broaden its applicability.

11.1 Future research

In the future, the detailed recommendations made in *Chapter 8: Discussion* can be built into a new Python module which delegates the work for a conversion to the recommended module. This way, one module can be created which has the best performance for *all* conversions.

It may include functionality to specify the type of inputs, e.g.:

- With or without collocations.
- The inclusion or exclusion of specific word forms, e.g. singular, plural, past, past participle or present participle.

The new module can use this information to better determine which modules should be used.

For any conversion the new module may optionally choose several

Python modules, perform the conversion with all picked modules, and then return the most frequently predicted output. This method might improve accuracy at the cost of execution time.

Researchers can experiment with which modules should be used for which conversions based on *Chapter 8: Discussion* and their own evaluation. Researchers may also create a flowchart for each conversion to determine which module has priority when several modules all produce different predicted outputs.

This way, even the complicated recommendations like from *Section 8.2.2: Judgment of converting verbs to plural* can effortlessly be used by developers.

11.2 Evaluation

This section enumerates possible improvements or additions to the quantitative and/or the qualitative evaluation procedures from this thesis.

1. Perform quantitative evaluation on merely the morphological analysis, i.e. answering the question: "Is the current word in a particular form". Our module `Inflexion`, and several others from *Chapter 6: Existing Python modules* support these kinds of wordform checks, but these tests extend beyond the scope of this thesis.

2. Some words convert differently based on the grammatical person desired. Some morphological analysis and generation algorithms have support for this, but this thesis does not test them. Qualitative and quantitative evaluation on conversions given a grammatical person should be performed.
   This should allow for the following examples, among others:
   - For the verb `` `to be' ``, converted to singular:

Person	**Output**	**Example**
1st	`` `am' ``	`` `I am' ``
2nd	`` `are' ``	`` `you are' ``
3rd	`` `is' ``	`` `he is' ``

   - For the noun `` `we' ``, converted to singular:

Person	**Output**	**Example**
1st	`` `I' ``	`` `I am' ``
2nd	`` `you' ``	`` `you are' ``
3rd	`` `it' ``	`` `it is' ``

   - For the adjective `` `your' ``, converted to singular:

Person	Output	Example
1st	`my'	`my book'
2nd	`your'	`your book'
3rd	`its'	`its book'

This functionality is implemented in `Inflexion`, but is not tested in this thesis.

3. Some words convert differently based on the grammatical gender desired. This functionality is supported by some morphological analysis and generation algorithms. Qualitative and quantitative evaluation should be performed for these cases.
   This should allow for the following example, among others:
   - For the adjective `our', converted to singular:

Gender	Output	Example
Masculine	`his'	`his book'
Feminine	`her'	`her book'
Neuter	`its'	`its book'

   This functionality is not implemented in `Inflexion`. The neuter case is always used as default.

4. Some nouns have classical plurals in addition to modern plurals, such as `brethren' as the classical plural of `brother', or `pence' as the classical plural of `penny'.

   This functionality is implemented in `Inflexion`, but not tested and compared to other morphological analysis and generation algorithms within this thesis. This can be expanded upon in future research.

5. Perform qualitative and quantitative evaluation on the comparative and superlative wordforms for adjectives. For example, for the adjective `good':
   - comparative: `better'
   - superlative: `best'

   This functionality does not exist in Damian Conway's `Lingua::EN::Inflexion`, but we did build it into `Inflexion`. What lacks is qualitative and quantitative evaluation for these conversions.

6. Perform quantitative evaluation on all 31 conversions with input words that were modified accordingly:
   - Different types of capitalisation:
     - `Title Case'.

  - `UPPER CASE'.
  - `RanDoM CAsE'.
- Different types of whitespace:
  - ` leading whitespace'.
  - `trailing whitespace '.
  - `duplicated whitespace between words'.
  - `replacing-spaces-with-hyphens'.
  - replacing hyphens with spaces: `knight-errant' into `knight errant'.

Results of these tests give developers insights on what kind of normalisation is required on data used in each module.

7. Modify the quantitative evaluation procedure by weighing each input by how often it exists in the English language. This way, errors for the verb `to do' have a larger effect on a module's score than errors for the verb collocation `to bowl over'.

   CELEX has support for the following parameters that may prove helpful for this:

CobDev	COBUILD frequency deviation
CobLog	COBUILD frequency, logarithmic
CobMln	COBUILD frequency (1,000,000)
CobSLog	COBUILD spoken frequency, logarithmic
CobSMln	COBUILD spoken frequency (1,000,000)
CobS	COBUILD spoken frequency 1.3m
CobSpellDev	COBUILD spelling frequency deviation
CobSpellFreq	COBUILD spelling frequency
CobWLog	COBUILD written frequency, logarithmic
CobWMln	COBUILD written frequency (1,000,000)
CobW	COBUILD written frequency 17.4m
Cob	COBUILD frequency

   Table 11.1: CELEX parameters regarding word frequency.

   We have looked into using these parameters, but were unsuccessful in creating a scoring function that met our standards without extending beyond the scope of this thesis.

8. Apply evaluation that takes into account the execution time of each program in relation to its performance. This way developers can be sure that the additional 0.1 percentage point accuracy does not come at the cost of 100 times the execution time.

11.3 `Inflexion`

This section enumerates possible improvements to the `Inflexion` Python module to improve its effectiveness and further broaden its applicability.

1. Improve morphological analysis of past participle, present participle and past tense verbs. Currently these forms are not correctly identified, causing the morphological generation to be applied on non-root words, e.g.:
   - `` `using' `` converted to past tense becomes `` `usinged' ``.

   Instead, the morphological analysis section should recognise that the input `` `using' `` is of the present participle form, and let the morphological generation apply on the root `` `use' `` instead.

2. Improve conversions surrounding comparative and superlative adjectives.
   - Add `is_comparative` and `is_superlative` functionality to detect whether a given adjective is of these forms.
   - Conversions from singular to comparative or superlative work well, but the conversion the other way around is not well supported yet.

3. Implement support for grammatical gender, or otherwise allow for non-neuter pronouns. For example, currently the singular of the adjective `` `our' `` is `` `its' ``. However, users of `Inflexion` may desire to specify the desired grammatical gender (masculine, feminine or neuter) so that `` `her' `` or `` `his' `` is returned instead.

Bibliography

[1] Algeo, John ; Butcher, Carmen A.: *The origins and development of the English language*. Cengage Learning, 2013

[2] Conway, Damian: *Lingua::EN::Inflexion*. https://metacpan.org/pod/Lingua::EN::Inflexion. Version: Jul 2020

[3] Dyson, Paul ; Coombs, Jason R. (Hrsg.): *inflect*. https://github.com/jaraco/inflect. Version: Nov 2020

[4] Vanhala, Janna: *Inflection*. https://github.com/jpvanhal/inflection. Version: Aug 2020

[5] Ixmatus: *inflector*. https://github.com/ixmatus/inflector. Version: Oct 2019

[6] Jascob, Brad: *LemmInflect*. https://github.com/bjascob/LemmInflect. Version: Feb 2020

[7] Bird, Steven ; Loper, Edward ; Klein, Ewan: *NLTK*. https://github.com/nltk/nltk. Version: Jan 2021

[8] Smedt, Tom D. ; Daelemans, Walter: *pattern*. https://github.com/clips/pattern. Version: Apr 2020

[9] Jascob, Brad: *pyInflect*. https://github.com/bjascob/pyInflect. Version: Feb 2020

[10] Loria, Steven: *TextBlob*. https://github.com/sloria/TextBlob. Version: Jan 2021

[11] Wouden, Ton Van d.: Celex: Building a multifunctional polytheoretical lexical data base. In: *Proceedings of BudaLex* 88 (1990), 363–373. https://tonvanderwouden.nl/index_files/papers/CELEX-euralex88.pdf

[12] Bosch, Antal van d. ; Daelemans, Walter: Memory-based morphological analysis. In: *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational*

Linguistics (1999). http://dx.doi.org/10.3115/1034678.1034726. – DOI 10.3115/1034678.1034726

[13] Kemmer, Suzanne: *Words in English: Structure*. https://www.ruf.rice.edu/~kemmer/Words04/structure/index.html

[14] *MORPHOLOGY II*. https://www.ling.upenn.edu/courses/Fall_1998/ling001/morphology2.html

[15] Minnen, Guido ; Carroll, John ; Pearce, Darren: Robust, applied morphological generation. In: *Proceedings of the first international conference on Natural language generation - INLG '00* (2000). http://dx.doi.org/10.3115/1118253.1118281. – DOI 10.3115/1118253.1118281

[16] Firth, John R.: Modes of meaning. Papers in Linguistics 1934-1951. In: *London, New York, et al* 190 (1957), S. 215

[17] Conway, Damian: *Lingua::EN::Inflect*. https://metacpan.org/pod/Lingua::EN::Inflect. Version: Dec 2020

[18] *Build software better, together*. https://github.com/topics/nlp

[19] Minnen, Guido ; Carroll, John ; Pearce, Darren: Applied morphological processing of English. In: *Natural Language Engineering* 7 (2001), Nr. 3, S. 207–223. http://dx.doi.org/10.1017/s1351324901002728. – DOI 10.1017/s1351324901002728

[20] Bloch, Bernard: English Verb Inflection. In: *Language* 23 (1947), Nr. 4, S. 399. http://dx.doi.org/10.2307/410300. – DOI 10.2307/410300

[21] Heuser, Ryan ; Falk, Josh ; Anttila, Arto: *quadrismegistus/prosodic*. https://github.com/quadrismegistus/prosodic. Version: 2010

[22] *Max Planck Institute for Psycholinguistics*. http://celex.mpi.nl/

[23] Martinez, Bermi F.: *Inflector*. https://api.rubyonrails.org/classes/ActiveSupport/Inflector.html. Version: Jun 2020

[24] *Penn Treebank P.O.S. Tags*. https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

[25] *What is WordNet?* https://wordnet.princeton.edu/

[26] *Morphy*. https://wordnet.princeton.edu/documentation/morphy7wn

[27] *Centre for Computational Linguistics and Psycholinguistics*. https://www.uantwerpen.be/en/research-groups/clips/

[28] Conway, Damian: An algorithmic approach to english pluralization. In: *Proceedings of the Second Annual Perl Conference*, 1998

[29] *The XTAG Project*. https://www.cis.upenn.edu/~xtag/

[30] Explosion: *spaCy*. https://github.com/explosion/spaCy

[31] Atkinson, Kevin: *Automatically Generated In ection Database (AGID)*. http://wordlist.aspell.net/agid-readme/

[32] Heemskerk, Josée S.: A probabilistic context-free grammar for disambiguation in morphological parsing. In: *Proceedings of the sixth conference on European chapter of the Association for Computational Linguistics* (1993). http://dx.doi.org/10.3115/976744.976767. – DOI 10.3115/976744.976767

Appendix A

Qualitative evaluation results

A.1 Nouns

A.1.1 Singular to singular

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	1	19734	0	0	4	0	0	0	498
Inflection	1	19881	0	0	0	0	0	0	355
Inflector	1	19625	0	0	0	0	0	0	611
Lingua::EN::Inflexion	0	20118	0	1	0	0	0	0	118
Inflexion	0	20124	0	1	0	0	0	0	112
LemmInflect	1	20019	0	76	1	0	0	0	140
NLTK	5024	15204	0	0	0	0	0	0	9
Pattern	0	19585	0	0	0	0	0	0	652
PyInflect	4952	15284	0	0	0	0	0	1	0
TextBlob	0	19584	0	0	0	0	0	0	653

Module	_vs_s	um_vs_a	my_vs_our	us_vs_i	an_vs_en	ouse_vs_ice	y_vs_ies	us_vs_era	other
inflect	459	4	0	0	9	6	3	4	13
Inflection	263	70	0	0	9	6	4	0	3
Inflector	497	70	0	22	9	6	4	0	3
Lingua::EN::Inflexion	96	3	0	0	8	6	2	0	3
Inflexion	96	3	0	0	8	0	2	0	3
LemmInflect	84	22	0	19	0	0	3	0	12
NLTK	8	0	0	0	0	0	0	0	1
Pattern	481	70	49	22	9	6	3	5	7
PyInflect	0	0	0	0	0	0	0	0	0
TextBlob	482	70	49	22	9	6	3	5	7

A.1.2 Plural to singular

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	0	20206	0	0	4	2	13	28	493
Inflection	0	19780	0	0	0	2	30	28	906
Inflector	0	19710	0	0	0	2	30	28	976
Lingua::EN::Inflexion	0	19878	0	0	0	0	10	28	830
Inflexion	0	20197	0	0	0	0	10	28	511
LemmInflect	0	19936	0	73	1	2	33	27	674
NLTK	5177	15101	0	0	0	0	0	0	468
Pattern	0	19940	0	0	0	2	28	28	748
PyInflect	20369	290	0	0	0	0	0	0	87
TextBlob	0	19963	0	1	1	7	28	28	718

Module	s_vs_	e_vs_	_vs_s	y_vs_ie	_vs_e	fe_vs_ve	i_vs_us	a_vs_um	other
inflect	67	62	72	32	29	0	36	55	140
Inflection	143	182	85	74	60	98	45	27	192
Inflector	203	182	91	75	56	98	32	27	212
Lingua::EN::Inflexion	97	53	62	78	46	1	15	15	463
Inflexion	104	81	62	78	40	0	15	15	116
LemmInflect	210	25	56	37	37	0	8	12	289
NLTK	369	8	9	0	2	0	3	7	70
Pattern	166	146	82	54	56	19	32	26	167
PyInflect	65	0	0	0	0	0	0	8	14
TextBlob	145	123	82	53	55	19	32	26	183

A.1.3 Uncountable to singular

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	0	6483	0	0	0	0	0	0	2126
Inflection	0	7693	0	0	0	0	0	0	916
Inflector	0	6282	0	0	0	0	0	0	2327
Lingua::EN::Inflexion	0	7956	0	0	0	0	0	0	653
Inflexion	0	7963	0	0	0	0	0	0	646
LemmInflect	0	7856	0	206	0	0	2	0	545
NLTK	2997	5511	0	0	0	0	0	0	101
Pattern	0	6279	0	1	0	0	0	0	2329
PyInflect	2747	5861	0	0	0	0	0	1	0
TextBlob	0	6280	0	0	0	1	0	0	2328

Module	_vs_s	um_vs_a	y_vs_ies	us_vs_i	_vs_es	my_vs_our	an_vs_en	ouse_vs_ice	other
inflect	2075	0	20	1	11	0	5	5	9
Inflection	717	148	21	0	12	0	5	5	8
Inflector	2105	148	21	23	12	0	5	5	8
Lingua::EN::Inflexion	591	0	20	1	11	0	5	5	20
Inflexion	592	0	20	1	11	0	5	1	16
LemmInflect	452	26	11	15	10	0	0	0	31
NLTK	95	0	3	0	1	0	0	0	2
Pattern	2071	148	19	21	12	35	5	5	13
PyInflect	0	0	0	0	0	0	0	0	0
TextBlob	2070	148	19	21	12	35	5	5	13

A.1.4 Singular to plural

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	0	19746	0	5	5	0	56	27	398
Inflection	0	19545	0	0	0	1	132	26	533
Inflector	0	19604	0	0	0	1	133	26	473
Lingua::EN::Inflexion	0	19747	0	4	0	0	132	27	327
Inflexion	0	19895	0	5	0	0	63	27	247
LemmInflect	0	19145	0	70	3	4	142	26	847
Pattern	0	19446	1	0	0	1	143	27	619
PyInflect	4952	15112	0	1	0	0	0	9	163
TextBlob	0	19471	0	0	0	1	143	27	595

Module	_vs_s	es_vs_s	s_vs_	s_vs_es	_vs_es	ans_vs_en	s_vs_x	us_vs_i	other
inflect	14	112	78	27	3	8	16	0	140
Inflection	107	16	67	44	95	0	16	35	153
Inflector	42	16	59	46	96	0	16	35	163
Lingua::EN::Inflexion	33	111	27	11	32	0	13	0	100
Inflexion	28	19	28	37	29	0	15	0	91
LemmInflect	364	21	56	16	33	134	15	0	208
Pattern	27	165	112	85	26	0	1	0	203
PyInflect	53	17	11	8	1	3	8	1	61
TextBlob	27	165	108	84	8	0	1	0	202

A.1.5 Plural to plural

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	0	332	0	0	4	0	5	2	20398
Inflection	0	20337	0	0	0	0	0	0	404
Inflector	0	18771	0	0	0	0	0	0	1970
Lingua::EN::Inflexion	0	20523	0	0	0	0	0	0	218
Inflexion	0	20549	0	1	0	0	0	0	191
LemmInflect	0	19698	0	76	3	3	1	2	958
Pattern	0	313	1	0	0	0	10	2	20415
PyInflect	20364	262	0	0	0	0	0	0	115
TextBlob	0	318	0	0	0	0	10	2	20411

Module	s_vs_	y_vs_ies	ss_vs_	es_vs_	_vs_s	ess_vs_	umss_vs_a	usess_vs_i	other
inflect	19803	0	198	157	0	47	45	25	123
Inflection	346	0	0	28	0	0	0	0	30
Inflector	647	1252	0	28	0	0	0	0	43
Lingua::EN::Inflexion	125	0	0	36	0	0	0	0	57
Inflexion	137	0	0	30	0	0	0	0	24
LemmInflect	371	33	0	78	328	0	0	0	148
Pattern	19922	0	206	26	1	47	45	25	143
PyInflect	25	0	0	60	0	0	0	0	30
TextBlob	19918	0	206	26	1	47	45	25	143

A.1.6 Uncountable to plural

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	0	40	0	0	0	0	0	0	8569
Inflection	0	893	0	0	0	0	0	0	7716
Inflector	0	729	0	0	0	0	0	0	7880
Lingua::EN::Inflexion	0	682	0	0	0	0	1	0	7926
Inflexion	0	676	0	0	0	0	1	0	7932
LemmInflect	0	3339	0	18	0	0	23	0	5229
Pattern	0	132	5	0	0	0	9	0	8463
PyInflect	2747	53	0	0	0	0	0	0	5809
TextBlob	0	123	0	0	0	0	9	0	8477

Module	s_vs_	es_vs_	ies_vs_y	es_vs_is	a_vs_um	ves_vs_f	des_vs_s	ies_vs_ey	other
inflect	5932	1713	844	34	0	8	0	9	29
Inflection	5198	1526	907	32	30	8	0	0	15
Inflector	5337	1520	907	33	30	8	0	0	45
Lingua::EN::Inflexion	5294	1652	902	32	1	8	0	1	36
Inflexion	5346	1611	902	32	1	8	0	1	31
LemmInflect	3489	823	598	30	2	0	25	0	262
Pattern	5963	1575	842	34	0	8	0	16	25
PyInflect	3482	1473	789	30	1	2	17	0	15
TextBlob	5960	1592	842	34	0	8	0	16	25

A.2 Verbs

A.2.1 Singular to singular

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
Lingua::EN::Inflexion	0	5960	0	0	0	0	0	0	2400
Inflexion	0	8353	0	0	0	0	0	0	7
LemmInflect	0	5936	0	2	5	2	0	1	2414
Pattern	0	5906	0	2	1	0	0	5	2446
PyInflect	8341	2	0	0	0	0	0	0	17

Module	s_vs_	ies_vs_y	es_vs_	_vs_s	s_vs_r	es_vs_s	s_vs_g	s_vs_es	other
Lingua::EN::Inflexion	2372	20	6	0	0	0	0	0	2
Inflexion	4	0	1	0	0	0	0	0	2
LemmInflect	2346	20	7	1	9	2	8	0	21
Pattern	2374	20	2	9	0	6	0	7	28
PyInflect	2	0	13	0	0	0	0	0	2

A.2.2 Plural to singular

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
Lingua::EN::Inflexion	0	5908	0	0	0	0	2397	1	49
Inflexion	0	8330	0	0	0	0	5	1	19
LemmInflect	0	5885	0	1	5	5	2394	2	63
Pattern	0	5863	0	2	1	0	2391	6	92
PyInflect	2553	5769	0	0	0	0	0	0	33

Module	_vs_s	es_vs_s	s_vs_es	es_vs_ses	_vs_es	s_vs_ds	s_vs_	inds_vs_ounds	other
Lingua::EN::Inflexion	27	3	6	0	6	0	0	0	7
Inflexion	2	1	4	4	5	0	0	0	3
LemmInflect	21	2	1	4	2	4	4	4	21
Pattern	35	6	8	2	4	7	2	4	24
PyInflect	7	16	1	4	0	0	2	0	3

A.2.3 Past to singular

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
Lingua::EN::Inflexion	0	53	0	0	0	1	2445	3	5975
Inflexion	0	157	0	0	0	1	6	3	8310
LemmInflect	0	5884	0	0	4	4	2444	2	139
Pattern	0	5802	0	1	20	1	2439	8	206
PyInflect	8384	53	0	0	0	1	0	0	39

Module	ds_vs_s	eds_vs_s	peds_vs_s	leds_vs_s	teds_vs_s	geds_vs_s	s_vs_es	beds_vs_s	other
Lingua::EN::Inflexion	2832	2290	96	91	77	58	4	36	491
Inflexion	3520	3212	157	105	104	79	10	52	1071
LemmInflect	5	0	0	0	0	0	17	0	117
Pattern	1	0	1	0	0	0	35	0	169
PyInflect	2	0	0	0	0	0	1	0	36

A.2.4 Past participle to singular

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
Lingua::EN::Inflexion	0	57	0	0	0	1	2474	2	5980
Inflexion	0	225	0	0	0	1	6	2	8280
LemmInflect	0	5895	0	0	4	4	2473	1	137
Pattern	0	5830	0	1	20	1	2467	7	188
PyInflect	8432	58	0	0	0	1	0	0	23

Module	ds_vs_s	eds_vs_s	peds_vs_s	ns_vs_s	leds_vs_s	teds_vs_s	geds_vs_s	s_vs_es	other
Lingua::EN::Inflexion	2832	2290	96	25	91	77	58	4	507
Inflexion	3520	3212	157	140	105	104	79	5	958
LemmInflect	5	0	0	12	0	0	0	18	102
Pattern	1	0	1	2	0	0	0	35	149
PyInflect	2	0	0	0	0	0	0	1	20

A.2.5 Present participle to singular

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
Lingua::EN::Inflexion	0	0	0	0	0	0	2396	0	5950
Inflexion	0	0	0	0	0	0	6	0	8340
LemmInflect	0	5866	0	1	5	2	2395	1	76
Pattern	0	5794	0	2	6	0	2390	5	149
PyInflect	8344	0	0	0	0	0	0	0	2

Module	ings_vs_s	ings_vs_es	yings_vs_ies	tings_vs_s	pings_vs_s	lings_vs_s	gings_vs_s	nings_vs_s	other
Lingua::EN::Inflexion	2399	2638	197	86	96	91	58	42	343
Inflexion	3743	3456	263	243	157	105	86	86	201
LemmInflect	0	0	0	0	0	0	0	0	76
Pattern	0	0	0	0	0	0	0	0	149
PyInflect	0	2	0	0	0	0	0	0	0

A.2.6 Singular to plural

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	0	6002	0	0	0	0	1	1	2356
Lingua::EN::Inflexion	0	5908	0	0	0	0	20	1	2431
Inflexion	0	8311	0	0	0	0	0	1	48
LemmInflect	0	5917	0	1	2	0	50	1	2389
NLTK	2604	5753	0	0	1	0	0	0	2
Pattern	0	5890	0	2	2	0	26	0	2440
PyInflect	8341	5	0	0	0	0	0	0	14
TextBlob	0	5701	0	0	1	0	1	1	2656

Module	s_vs_	es_vs_	ies_vs_y	_vs_e	e_vs_	s_vs_ve	y_vs_ie	l_vs_	other
inflect	2098	184	43	5	10	0	11	0	5
Lingua::EN::Inflexion	2152	195	45	3	12	9	9	0	6
Inflexion	10	0	1	3	19	0	8	0	7
LemmInflect	2124	193	44	4	2	9	3	3	7
NLTK	1	0	0	0	0	0	0	0	1
Pattern	2146	196	45	3	4	9	2	7	28
PyInflect	14	0	0	0	0	0	0	0	0
TextBlob	2325	206	52	57	0	9	2	0	5

A.2.7 Plural to plural

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	0	8379	0	0	0	0	0	0	1
Lingua::EN::Inflexion	0	8349	0	0	0	0	0	0	31
Inflexion	0	8373	0	0	0	0	0	0	7
LemmInflect	0	8282	0	2	2	3	0	0	91
NLTK	2605	5774	0	0	1	0	0	0	0
Pattern	0	8333	0	3	1	0	0	0	43
PyInflect	2556	5824	0	0	0	0	0	0	0
TextBlob	0	8372	0	0	1	0	0	0	7

Module	_vs_s	_vs_nd	_vs_d	ind_vs_ound	_vs_ind	_vs_ong	_vs_g	l_vs_	other
inflect	1	0	0	0	0	0	0	0	0
Lingua::EN::Inflexion	31	0	0	0	0	0	0	0	0
Inflexion	7	0	0	0	0	0	0	0	0
LemmInflect	22	17	5	4	6	6	5	3	23
NLTK	0	0	0	0	0	0	0	0	0
Pattern	29	0	6	3	0	0	0	0	5
PyInflect	0	0	0	0	0	0	0	0	0
TextBlob	2	0	1	4	0	0	0	0	0

A.2.8 Past to plural

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	0	157	0	0	0	1	1	4	8314
Lingua::EN::Inflexion	0	155	0	0	0	1	20	4	8297
Inflexion	0	157	0	0	0	1	1	4	8314
LemmInflect	0	5989	0	0	2	2	44	2	2438
NLTK	2696	5744	0	0	1	1	0	0	35
Pattern	0	5921	0	1	22	1	24	3	2505
PyInflect	8384	51	0	0	0	1	0	0	41
TextBlob	0	5805	0	0	1	1	1	3	2666

Module	ed_vs_	d_vs_	ied_vs_y	ped_vs_	ted_vs_	led_vs_	ame_vs_ome	ged_vs_	other
inflect	3705	2782	231	157	104	105	48	79	1103
Lingua::EN::Inflexion	3696	2776	231	157	104	105	47	79	1102
Inflexion	3705	2782	231	157	104	105	48	79	1103
LemmInflect	1029	459	42	57	25	14	42	20	750
NLTK	0	2	0	0	0	0	0	0	33
Pattern	1032	460	43	59	25	14	43	20	809
PyInflect	1	3	0	0	0	0	0	0	37
TextBlob	1123	515	56	68	34	21	45	27	777

A.2.9 Past participle to plural

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	0	225	0	0	0	1	1	3	8284
Lingua::EN::Inflexion	0	222	0	0	0	1	19	3	8269
Inflexion	0	225	0	0	0	1	1	3	8284
LemmInflect	0	6057	0	0	2	2	42	1	2410
NLTK	2717	5767	0	0	1	1	0	0	28
Pattern	0	6010	0	1	22	1	22	2	2456
PyInflect	8432	55	0	0	0	1	0	0	26
TextBlob	0	5892	0	0	1	1	1	2	2617

Module	ed_vs_	d_vs_	ied_vs_y	n_vs_	ped_vs_	ted_vs_	led_vs_	ne_vs_	other
inflect	3705	2782	231	140	157	104	105	52	1008
Lingua::EN::Inflexion	3696	2776	231	140	157	104	105	52	1008
Inflexion	3705	2782	231	140	157	104	105	52	1008
LemmInflect	1029	459	42	105	57	25	14	42	637
NLTK	0	2	0	0	0	0	0	0	26
Pattern	1032	460	43	96	59	25	14	43	684
PyInflect	1	3	0	0	0	0	0	0	22
TextBlob	1123	515	56	96	68	34	21	43	661

A.2.10 Present participle to plural

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	0	0	0	0	0	0	1	1	8344
Lingua::EN::Inflexion	0	0	0	0	0	0	21	1	8324
Inflexion	0	0	0	0	0	0	1	1	8344
LemmInflect	0	5869	0	1	2	0	87	1	2386
NLTK	2618	5712	0	0	1	0	0	0	15
Pattern	0	5809	0	2	7	0	27	0	2501
PyInflect	8344	0	0	0	0	0	0	0	2
TextBlob	0	5663	0	0	1	0	1	1	2680

Module	ing_vs_	ing_vs_e	ting_vs_	ping_vs_	ning_vs_	ling_vs_	ging_vs_	bing_vs_	other
inflect	4529	2910	243	157	87	105	86	52	175
Lingua::EN::Inflexion	4520	2901	241	157	87	105	86	52	175
Inflexion	4529	2910	243	157	87	105	86	52	175
LemmInflect	1434	560	136	56	39	14	25	15	107
NLTK	0	0	0	0	0	0	0	0	15
Pattern	1469	574	137	58	40	14	26	16	167
PyInflect	1	1	0	0	0	0	0	0	0
TextBlob	1576	637	152	68	46	21	33	19	128

A.2.11 Singular to past

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
Lingua::EN::Inflexion	0	5605	0	0	0	1	2416	1	333
Inflexion	0	8199	0	0	0	1	6	1	149
LemmInflect	0	5799	0	1	3	3	2416	2	132
Pattern	0	5799	0	1	6	1	2416	1	132
PyInflect	8338	2	0	0	0	0	0	0	16

Module	ed_vs_led	_vs_ed	ed_vs_d	ed_vs_ted	ed_vs_ped	ted_vs_ed	ed_vs_red	ed_vs_	other
Lingua::EN::Inflexion	8	33	4	2	3	64	10	4	205
Inflexion	0	43	11	8	11	13	13	2	48
LemmInflect	49	0	3	11	8	1	2	4	54
Pattern	9	1	13	2	2	4	0	7	94
PyInflect	0	0	0	0	0	0	0	1	15

A.2.12 Plural to past

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
Lingua::EN::Inflexion	0	5586	0	0	0	1	2313	2	449
Inflexion	0	8186	0	0	0	1	5	2	157
LemmInflect	0	5779	0	1	3	6	2310	3	249
Pattern	0	5792	0	1	4	1	2313	2	238
PyInflect	2551	5698	0	0	0	0	0	1	101

Module	ed_vs_	ed_vs_led	_vs_ed	ed_vs_d	ed_vs_ted	ed_vs_ped	ned_vs_	ted_vs_ed	other
Lingua::EN::Inflexion	65	8	33	8	2	3	14	64	252
Inflexion	2	0	43	13	8	11	0	13	67
LemmInflect	82	49	7	6	11	8	4	1	81
Pattern	59	9	7	15	2	2	17	5	122
PyInflect	1	60	9	0	5	5	0	1	20

A.2.13 Past to past

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
Lingua::EN::Inflexion	0	5920	0	0	0	0	49	0	2506
Inflexion	0	8380	0	0	0	0	0	0	95
LemmInflect	0	5869	0	1	2	3	50	2	2548
Pattern	0	5868	0	1	7	0	49	1	2549
PyInflect	8382	52	0	0	0	0	0	1	40

Module	ed_vs_	ned_vs_	ped_vs_	red_vs_	d_vs_	ted_vs_	med_vs_	ed_vs_led	other
Lingua::EN::Inflexion	1526	367	318	95	34	62	32	0	72
Inflexion	23	3	0	0	13	1	0	0	55
LemmInflect	2139	51	92	37	29	10	19	45	126
Pattern	1263	449	318	221	31	64	32	6	165
PyInflect	18	2	0	0	6	3	0	0	11

A.2.14 Past participle to past

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
Lingua::EN::Inflexion	0	5894	0	0	0	0	390	0	2228
Inflexion	0	8327	0	0	0	0	1	0	184
LemmInflect	0	5876	0	1	2	3	391	2	2237
Pattern	0	5890	0	1	7	0	390	1	2223
PyInflect	8430	56	0	0	0	0	0	1	25

Module	ed_vs_	ned_vs_	ped_vs_	red_vs_	ted_vs_	ed_vs_led	med_vs_	fed_vs_	other
Lingua::EN::Inflexion	1299	318	296	83	54	0	27	0	151
Inflexion	6	9	0	0	0	0	0	0	169
LemmInflect	1853	44	85	37	10	45	19	7	137
Pattern	1078	394	296	187	55	6	27	35	145
PyInflect	13	1	0	0	2	0	0	0	9

A.2.15 Present participle to past

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
Lingua::EN::Inflexion	0	180	0	0	0	1	2415	3	5745
Inflexion	0	829	0	0	0	1	6	3	7505
LemmInflect	0	5793	0	1	3	3	2414	1	129
Pattern	0	5802	0	1	7	1	2414	1	118
PyInflect	8342	0	0	0	0	0	0	0	2

Module	inged_vs_ed	yinged_vs_ied	ed_vs_led	inged_vs_d	inged_vs_	tinged_vs_	unninged_vs_an	other
Lingua::EN::Inflexion	5354	194	0	34	20	13	3	127
Inflexion	6832	236	0	43	37	36	28	293
LemmInflect	0	0	47	0	0	0	0	82
Pattern	0	0	6	0	0	0	0	112
PyInflect	1	0	0	0	0	0	0	1

A.2.16 Singular to past participle

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
Lingua::EN::Inflexion	0	5597	0	0	0	1	2416	1	338
Inflexion	0	8194	0	0	0	1	6	1	151
LemmInflect	0	5796	0	1	4	2	2416	1	133
Pattern	0	5802	0	1	4	1	2416	0	129
PyInflect	8336	2	0	0	0	0	0	0	15

Module	ed_vs_led	_vs_ed	ed_vs_d	ed_vs_ted	ed_vs_ped	ted_vs_ed	ed_vs_red	sed_vs_ed	other
Lingua::EN::Inflexion	8	30	9	2	3	64	10	0	212
Inflexion	0	39	14	8	11	13	13	0	53
LemmInflect	45	1	2	11	8	1	2	3	60
Pattern	9	1	13	2	2	4	0	2	96
PyInflect	0	0	0	0	0	0	0	7	8

A.2.17 Plural to past participle

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
Lingua::EN::Inflexion	0	5584	0	0	0	1	2249	2	512
Inflexion	0	8186	0	0	0	1	5	2	154
LemmInflect	0	5779	0	1	4	5	2246	2	311
Pattern	0	5793	0	1	5	1	2249	1	298
PyInflect	2549	5707	0	0	0	0	0	1	91

Module	ed_vs_	ed_vs_led	_vs_ed	ed_vs_d	ned_vs_	ed_vs_ted	ed_vs_ped	ted_vs_ed	other
Lingua::EN::Inflexion	110	8	30	8	24	2	3	64	263
Inflexion	0	0	39	13	0	8	11	13	70
LemmInflect	137	45	6	5	6	11	8	1	92
Pattern	94	9	3	15	26	2	2	5	142
PyInflect	0	57	5	0	0	5	4	1	19

A.2.18 Past to past participle

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
Lingua::EN::Inflexion	0	5888	0	0	0	0	356	1	2227
Inflexion	0	8244	0	0	0	0	1	1	226
LemmInflect	0	5864	0	1	3	2	357	4	2241
Pattern	0	5847	0	1	21	0	356	2	2245
PyInflect	8380	51	0	0	0	0	0	1	40

Module	ed_vs_	ned_vs_	ped_vs_	red_vs_	ted_vs_	ed_vs_led	med_vs_	fed_vs_	other
Lingua::EN::Inflexion	1303	318	296	83	54	0	28	0	145
Inflexion	6	0	0	0	0	0	0	0	220
LemmInflect	1858	42	85	37	10	41	19	7	142
Pattern	1082	394	296	188	55	6	28	35	161
PyInflect	13	1	0	0	2	0	0	0	24

A.2.19 Past participle to past participle

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
Lingua::EN::Inflexion	0	5933	0	0	0	0	78	0	2501
Inflexion	0	8445	0	0	0	0	0	0	67
LemmInflect	0	5881	0	1	3	2	79	3	2543
Pattern	0	5881	0	1	21	0	78	1	2530
PyInflect	8430	58	0	0	0	0	0	1	23

Module	ed_vs_	ned_vs_	ped_vs_	red_vs_	ted_vs_	med_vs_	fed_vs_	ed_vs_led	other
Lingua::EN::Inflexion	1519	379	318	95	60	32	0	0	98
Inflexion	16	1	0	0	0	0	0	0	50
LemmInflect	2145	51	92	37	12	19	7	41	139
Pattern	1254	455	318	221	61	32	41	6	142
PyInflect	14	1	0	0	2	0	0	0	6

A.2.20 Present participle to past participle

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
Lingua::EN::Inflexion	0	180	0	0	0	1	2415	2	5746
Inflexion	0	828	0	0	0	1	6	2	7507
LemmInflect	0	5795	0	1	4	2	2414	1	127
Pattern	0	5804	0	1	6	1	2414	0	118
PyInflect	8342	0	0	0	0	0	0	0	2

Module	inged_vs_ed	yinged_vs_ied	ed_vs_led	inged_vs_d	inged_vs_n	tinged_vs_	inged_vs_	ninged_vs_	other
Lingua::EN::Inflexion	5355	194	0	34	10	13	18	3	119
Inflexion	6834	236	0	43	43	36	34	28	253
LemmInflect	0	0	43	0	0	0	0	0	84
Pattern	0	0	6	0	0	0	0	0	112
PyInflect	1	0	0	0	0	0	0	0	1

A.2.21 Singular to present participle

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	0	5568	0	0	0	0	2416	0	367
Lingua::EN::Inflexion	0	5653	0	0	0	0	2415	0	283
Inflexion	0	8248	0	0	0	0	6	0	97
LemmInflect	0	5822	0	1	3	0	2415	0	110
Pattern	0	5845	0	2	4	0	2415	0	85
PyInflect	8334	2	0	0	0	0	0	0	15

Module	ing_vs_ling	ning_vs_ing	ting_vs_ing	ing_vs_eing	ring_vs_ing	ing_vs_ring	ing_vs_ting	ing_vs_ping	other
inflect	91	109	69	15	20	10	3	3	47
Lingua::EN::Inflexion	8	83	69	6	21	10	2	3	81
Inflexion	0	1	15	10	1	13	8	11	38
LemmInflect	52	1	1	7	1	2	12	8	26
Pattern	9	4	4	6	8	0	2	2	50
PyInflect	0	0	0	0	0	0	0	0	15

A.2.22 Plural to present participle

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	0	5567	0	0	0	0	2415	1	363
Lingua::EN::Inflexion	0	5636	0	0	0	0	2415	1	294
Inflexion	0	8240	0	0	0	0	5	1	100
LemmInflect	0	5799	0	1	3	3	2412	1	127
Pattern	0	5834	0	2	4	0	2415	0	91
PyInflect	2547	5707	0	0	0	0	0	0	92

Module	ing_vs_ling	ning_vs_ing	ting_vs_ing	ing_vs_eing	ring_vs_ing	ing_vs_ping	ing_vs_ting	ing_vs_ring	other
inflect	91	109	69	16	21	3	3	10	41
Lingua::EN::Inflexion	8	83	69	8	21	3	2	10	90
Inflexion	0	1	15	10	1	11	8	13	41
LemmInflect	52	1	1	7	1	8	12	2	43
Pattern	9	6	5	6	8	2	2	0	53
PyInflect	65	0	1	6	1	5	3	1	10

A.2.23 Past to present participle

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	0	53	0	0	0	1	2464	3	5949
Lingua::EN::Inflexion	0	223	0	0	0	1	2464	3	5779
Inflexion	0	906	0	0	0	1	6	3	7554
LemmInflect	0	5855	0	1	3	2	2463	1	145
Pattern	0	5856	0	1	18	1	2463	2	129
PyInflect	8378	53	0	0	0	1	0	0	38

Module	edding_vs_ing	eding_vs_ing	ieding_vs_ying	ding_vs_ing	ing_vs_ling	ewing_vs_owing	iding_vs_ying	other
inflect	5376	66	196	32	0	9	15	255
Lingua::EN::Inflexion	5319	65	194	30	0	7	13	151
Inflexion	2746	4146	238	41	0	24	17	344
LemmInflect	0	0	0	5	48	2	0	90
Pattern	0	1	0	3	6	1	2	116
PyInflect	0	2	0	2	0	1	0	33

A.2.24 Past participle to present participle

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	0	57	0	0	0	1	2493	2	5957
Lingua::EN::Inflexion	0	231	0	0	0	1	2493	2	5783
Inflexion	0	986	0	0	0	1	6	2	7515
LemmInflect	0	5866	0	1	3	2	2492	0	146
Pattern	0	5885	0	1	18	1	2492	1	112
PyInflect	8428	59	0	0	0	1	0	0	22

Module	edding_vs_ing	eding_vs_ing	ieding_vs_ying	ning_vs_ing	ding_vs_ing	ing_vs_ling	ening_vs_ing	other
inflect	5376	66	196	33	33	0	1	252
Lingua::EN::Inflexion	5319	65	194	21	31	0	2	151
Inflexion	2746	4146	238	49	39	0	35	262
LemmInflect	0	0	0	5	6	48	6	81
Pattern	0	1	0	3	4	6	0	98
PyInflect	0	2	0	0	2	0	0	18

A.2.25 Present participle to present participle

Module	empty	correct	whitespace	case	dash_space	collocations	wrong_term_changed	quote	suffix
inflect	0	0	0	0	0	0	1	0	8343
Lingua::EN::Inflexion	0	5929	0	0	0	0	0	0	2415
Inflexion	0	8333	0	0	0	0	0	0	11
LemmInflect	0	5822	0	2	3	0	0	0	2517
Pattern	0	5850	0	2	4	0	0	0	2488
PyInflect	8342	0	0	0	0	0	0	0	2

Module	ing_vs_	ning_vs_	ping_vs_	ring_vs_	ting_vs_	ming_vs_	ing_vs_ling	ing_vs_e	other
inflect	7359	449	318	95	57	32	0	18	15
Lingua::EN::Inflexion	1511	364	318	95	57	32	0	18	20
Inflexion	11	0	0	0	0	0	0	0	0
LemmInflect	2205	30	75	0	0	0	51	18	138
Pattern	1260	449	318	220	57	32	6	18	128
PyInflect	2	0	0	0	0	0	0	0	0