pymorphy Documentation

Выпуск 0.5.6

Mikhail Korobov

03 October 2011

Оглавление

Содержание:

pymorphy

1.1 Возможности библиотеки

1. Умеет приводит слово к нормальной форме (например, в ед.ч., И.п. для существительных):

```
>>> from pymorphy import get_morph
>>> morph = get_morph('dicts/ru')
>>> print morph.normalize(u"ЛЮДЕЙ")
ЧЕЛОВЕК
```

2. Умеет ставить слово в нужную форму. Например, ставить слово во множественное число, менять падеж слова и т.д.:

```
>>> print morph.inflect_ru(u"СУСЛИК", u"мн,рд") # много кого?
СУСЛИКОВ
```

   Есть template filter, который позволяет делать это прямо в шаблоне django:

```
{# в переменной animals "тридцать восемь попугаев и Удав" #}

{% load pymorphy_tags %}
{{ animals|inflect:"дт" }} захотелось пройтись по лесу.

{# выведет "тридцати восьми попугаям и Удаву захотелось пройтись по лесу" #}
```

3. Умеет возвращать грамматическую информацию о слове (число, род, падеж, часть речи и т.д.). Делает это по словарю, для неизвестных слов работает предсказатель, если возможных форм несколько - возвращает несколько форм:

```
>>> info = morph.get_graminfo(u"БУТЯВКОВЕДАМИ")
>>> print info[0]['norm'] # нормальная форма
БУТЯВКОВЕД
>>> print info[0]['class'] # часть речи, С = существительное
С
>>> print info[0]['info'] # род, число, падеж и т.д.
мр,мн,тв
```

1.2 Установка

1. Устанавливаем pymorphy

   $ pip install pymorphy

2. Скачиваем словари. Для начала можно скачать файл с русскими словарями в sqlite.

   Все словари лежат тут: https://bitbucket.org/kmike/pymorphy/downloads/

3. Распаковываем скачанный файл.

1.3 Пример использования

```
from pymorphy import get_morph

morph = get_morph('<путь/до/папки/в/которую/были/распакованы/скачанные/словари>')

#слова должны быть в юникоде и ЗАГЛАВНЫМИ
info = morph.get_graminfo(u'ВАСЯ')
```

Больше информации можно найти в Руководстве.

Примечание: В pymorphy все, что не относится к django, можно использовать без django.

1.4 Лицензия, полезные ссылки и т.д.

Лицензия - MIT.

Должно работать на windows и *nix-системах, python 2.5, 2.6 и 2.7. Python 2.4 не поддерживается.

- Обсуждение (тут можно задавать вопросы, делиться опытом, предлагать идеи)
- Сообщить об ошибке
- Репозиторий с исходным кодом

Подключайтесь к разработке! Замечания, исправления, документация, патчи в любом виде всегда приветствуются.

Если у вас какой-то вопрос, пишите о нем, по возможности, в гугл-группу - найденным ответом смогут потом воспользоваться другие люди.

Глава 2

Руководство

Содержание:

2.1 Использование

2.1.1 Подготовка

Чтобы использовать морфологический анализатор, нужно сначала создать объект класса pymorphy.Morph:

```
from pymorphy import get_morph
morph = get_morph('dicts/ru')
```

Аргументы pymorphy.get_morph:

- path - обязательный параметр, путь до папки с файлами;
- backend - используемое key-value хранилище ('sqlite' по умолчанию);
- cached - использовать ли кэш (True по умолчанию).

Можно также передавать любые дополнительные аргументы, которые принимает конструктор класса pymorphy.Morph.

Примечание: Обратите внимание, все методы Morph ожидают, что строковые аргументы (в.т.ч. пустые или латинские, если используется pymorphy-speedups) - это unicode-строки. Кроме того, слова для обработки должны быть в верхнем регистре.

Предупреждение: Всегда старайтесь использовать единственный экземпляр анализатора. Объекты класса pymorphy.Morph требуют довольно много ресурсов для создания, не уничтожаются сборщиком мусора и не закрывают за собой файловые дескрипторы, поэтому постоянное создание анализаторов будет приводить к утечке ресурсов.

2.1.2 Получение информации о слове

```
>>> word = u'Вася'.upper()
>>> info = morph.get_graminfo(word)[0]
>>> print info['norm']
ВАСЯ
>>> print info['class']
С
>>> print info['info']
мр,имя,ед,им
>>> print info['method']
lemma(ВАС).suffix(Я)
```

Morph.get_graminfo() возвращает list всех возможных вариантов разбора слова. Каждый вариант разбора - dict, в котором есть нормальная форма, часть речи, грамматическая информация и служебные данные для отладки. См. также Обозначения для грамматической информации в pymorphy.

2.1.3 Получение нормальных форм

```
>>> morph.normalize(u'БУТЯВКАМ')
set(u'БУТЯВКА')
```

Morph.normalize() возвращает множество (set) всех возможных нормальных форм слова.

2.1.4 Склонение

```
>>> morph.inflect_ru(u'БУТЯВКА', u'дт,мн')
БУТЯВКАМ
```

Morph.inflect_ru() возвращает слово в форме, которая соответствует переданной и меньше всего отличается от исходной. В случае, если такую форму найти не удается, возвращается исходное слово.

Примечание: Этот метод на данный момент не работает с фамилиями. См. Склонение имен и фамилий.

См. также: Обозначения для грамматической информации в pymorphy

2.1.5 Постановка во множественное число

Простое:

```
>>> morph.pluralize_ru(u'БУТЯВКЕ')
БУТЯВКАМ
```

Согласованное с цифрой:

```
>>> morph.pluralize_inflected_ru(u'ПОПУГАЙ', 1)
ПОПУГАЙ
>>> morph.pluralize_inflected_ru(u'ПОПУГАЙ', 2)
ПОПУГАЯ
>>> morph.pluralize_inflected_ru(u'ПОПУГАЙ', 38)
ПОПУГАЕВ
```

См. Morph.pluralize_ru(), Morph.pluralize_inflected_ru().

2.2 Интеграция с django

2.2.1 Настройка

1. Добавляем 'pymorphy' в INSTALLED_APPS.
2. Описываем в settings.py установленные словари:

   ```
   PYMORPHY_DICTS = {
       'ru': { 'dir': '/usr/share/pymorphy/ru' },
   }
   ```

   более сложный пример:

   ```
   PYMORPHY_DICTS = {
       'ru': {
           'dir': os.path.join([PROJECT_DIR, 'files', 'dicts']),
           'backend': 'cdb',
           'use_cache': False,
       },
   }
   ```

   Параметры:

   - dir - обязательный параметр, путь до папки с файлами;
   - backend - используемое key-value хранилище ('sqlite' по умолчанию);
   - use_cache - использовать ли кэш (True по умолчанию).

2.2.2 Получение экземпляра анализатора для ручного использования

В случае, когда настроена интеграция pymorphy с django, экземпляр анализатора следует получать следующим образом:

```
from pymorphy.django_conf import default_morph as morph
```

Не стоит получать экземпляр анализатора через pymorphy.get_morph во вьюхах (или еще где-то на каждый запрос) - это приведет к утечке ресурсов.

2.2.3 Шаблонные фильтры

Фильтры подключаются следующей командой:

```
{% load pymorphy_tags %}
```

inflect

Меняет грамматическую форму каждого слова на указанную в параметрах. Про доступные параметры можно почитать тут (см. "Полный формат").

Пример:

```
{# в переменной city "Нижний Новгород" #}

{% load pymorphy_tags %}
Мы начали работу в {{ city|inflect:"пр" }}!

{# выведет "Мы начали работу в Нижнем Новгороде!" #}
```

Пример с несклоняемой частью

```
{% load pymorphy_tags %}

Не осталось у нас {{ "лошадь [[Пржевальского]]"|inflect:"рд,мн" }}.

{# выведет "Не осталось у нас лошадей Пржевальского" #}
```

inflect_marked

Идентичен фильтру inflect за исключением того, что противоположным образом трактует [[ ]]

```
{% load pymorphy_tags %}
Не осталось у нас {{ "[[лошадь]] Пржевальского"|inflect_marked:"рд,мн" }}.

{# выведет "Не осталось у нас лошадей Пржевальского" #}
```

plural

Ставит слово в форму, которая согласуется с заданным числом (1 попугай, 2 попугая, 5 попугаев).

```
{% load pymorphy_tags %}

{# в переменной num число попугаев (пусть = 38) #}
На дереве {{ num }} {{ "попугай"|plural:num }}.
{# выведет "На дереве 38 попугаев." #}

{# в переменной animal - "лошадь" #}
А еще есть {{ num }} {{ animal|plural:num }}.
{# выведет "А еще есть 38 лошадей." #}
```

Фильтры inflect и plural не склоняют все, что заключено в двойные квадратные скобки. Фильтр inflect_marked - наоборот, работает только с тем, что в двойных квадратных скобках.

Можно указать другие разделители (обязательно 2х-символьные), определив в settings.py переменные PYMORPHY_MARKER_OPEN и PYMORPHY_MARKER_CLOSE.

Примечание: Фильтры из pymorphy_tags стараются сохранить написание больших-маленьких букв (обрабатываются варианты “ВСЕ СЛОВО БОЛЬШИМИ”, “С заглавной”, “все маленькими”).

Если по какой-то причине смена формы не удалась, возвращают исходную строку.

> Предупреждение: Фильтры в настоящий момент могут плохо работать с именами и фамилиями.

2.3 Скорость

2.3.1 Скорость разбора

С pymorphy можно ожидать разбор нескольких сотен русских слов в секунду "из коробки". После дополнительной настройки можно получить производительность в несколько тысяч слов в секунду.

Этой скорости достаточно для многих задач (например, для различных экспериментов и задач, связанных с web), но pymorphy в нынешнем виде, думаю, не подойдет, если нужно быстро обрабатывать очень большие объемы данных. В этом случае лучше использовать lemmatizer или mystem.

У pymorphy нет цели быть быстрым, приоритет отдается качеству разбора и легкости сопровождения. С учетом того, что это хобби-opensource-проект, код и алгоритмы должны быть максимально простыми и понятными, чтобы облегчить внесение изменений и доработку под конкретные задачи.

На данный момент pymorphy можно заставить работать быстрее несколькими способами:

- перейти на более быстрое хранилище (sqlite → cdb → pickle);
- отключить ненужные предсказатели;
- установить simplejson (для упрощения установки pymorphy его не требует и использует по умолчанию встроенный медленный модуль):

  ```
  $ pip install simplejson
  ```

- поставить пакет pymorphy-speedups, который содержит авто-подключаемое Cython-расширение:

  ```
  $ pip install pymorphy-speedups
  ```

Примечание: Для установки pymorphy-speedups и simplejson потребуются заголовочные файлы питона и среда с компилятором (как и для сборки любых других расширений).

2.3.2 Выбор хранилища для словарей

pymorphy поддерживает разные форматы для хранения словарей. Формат по умолчанию - sqlite. Этот формат поддерживается везде, не требует настройки, но, одновременно, является самым медленным.

Более быстрые альтернативы - cdb, bsddb, tcb, tch - имеют свои плюсы и минусы, отличаются друг от друга способом установки, скоростью и потреблением памяти.

Самый быстрый вариант - это загрузка словарей целиком в память (через pickle backend). В этом случае нет задержек на чтение данных с диска и преобразование их в нужный формат (все читается сразу), но расходуется 200-300Мб оперативной памяти. В этот формат словари можно преобразовать с помощью скрипта encode_dicts.py (лежит в репозитории с исходным кодом).

Более подробно обо всем этом можно узнать тут: Поддерживаемые типы хранилищ.

2.4 Дополнительные возможности

2.4.1 Разбиение текста на слова и токены

Примечание: В библиотеке nltk реализовано большое число классов для разбора текста на токены (см. nltk.tokenize), которыми можно пользоваться, если функции из pymorphy чем-то не подходят.

Установка nltk не всегда тривиальная, а в написанном по-быстрому регэкспе можно легко не учесть какие-то детали, поэтому в pymorphy все-таки появился pymorphy.contrib.tokenizers.

---

pymorphy.contrib.tokenizers.extract_tokens(text)
: Разбивает текст на токены - слова, пробелы, знаки препинания и возвращает полученный массив строк.

pymorphy.contrib.tokenizers.extract_words(text)
: Разбивает текст на слова. Пунктуация игнорируется. Слова, пишущиеся через дефис, считаются 1 словом. Пример использования:

```
from pymorphy.contrib import tokenizers

for word in tokenizers.extract_words(text):
    print word
```

Возвращает генератор, выдающий слова из текста (не list).

2.4.2 Склонение имен и фамилий

Склонение фамилий

При разборе текста фамилии часто неотличимы (без дополнительной информации) от других слов: “много козлов” - “василий козлов”. При этом фамилии склоняются по другим правилам, не как обычные слова, поэтому морфологическому анализатору для правильной работы необходимо:

- знать, что слово - фамилия;
- уметь склонять фамилии по специальным правилам.

В pymorphy реализована вторая часть: склонять фамилии можно с помощью функций из модуля pymorphy.contrib.lastnames_ru, который реализует правила склонения фамилий, описанные на gramota.ru.

> Предупреждение: Работа с фамилиями в pymorphy сейчас экспериментальная, и не все методы обычного анализатора доступны (в частности, не завершена работа над постановкой фамилий во множественную форму), API может поменяться в последующих релизах.

Склонение фамилий использует алгоритм, похожий на алгоритм предсказания склонения слов, отсутствующих в словаре.

Угадывание леммы и суффикса

Если слово содержит суффикс, характерный для фамилии (например, суффикс -ов- характерен для русских фамилий “Усов”, “Хвостов”, “Котов”), то часть до суффикса принимается за лемму, часть после суффикса отбрасывается, и лемма плюс суффикс склоняются.

Например, слово “Петровичу” будет разделено на составляющие “петров”, “-ич-” и “у”. Где “петров” - лемма, “-ич-” - суффикс фамилии, а “у” будет проигнорировано. Согласно правилам склонения фамилий с суффиксом -ич-, слово “петров[-ич-]” будет склонено в шести падежах как “петрович”, “петровича”, “петровичу”... для мужского рода, и “петрович”, “петрович”, “петрович”... для женского.

Нормализация фамилий

На базе этого построена нормализация фамилий: нормальная форма получается только после выделения леммы и суффикса во время попытки склонения. Получив возможные варианты во всех падежах, мы можем проверить, что а) входное слово “петровичу” может склоняться как фамилия; б) исходное слово - это дательный падеж фамилии; и принять именительный падеж (“петрович”) за нормальную форму, если оба условия выполняются.

Если из входного слова могут быть выделены лемма и суффикс, но в то же время оно не является вариантом склонения этой фамилии, то это расценивается как ложное срабатывание. Пример: слово “кроссовый” хоть и подходит под шаблон фамилии “кросс[-ов-]”, но, по правилам склонения фамилий, не может быть её производным.

Если decline() не нашла характерных признаков фамилии, входное слово будет склонено как обычное (не фамилия) соответствующего рода в именительном падеже. Например, в случае нормализации слова “кроссового”, вызов lastnames_ru.normalize(morph, u’КРОССОВОГО’, u’мр’) будет аналогичен вызову morph.inflect_ru(u’КРОССОВОГО’, u’им,ед,мр’).

Известные ограничения

В некоторых случаях невозможно точно определить нормальную форму из-за особенностей написания некоторых фамилий: Крамского, Достоевского. В первом случае нормальной формой должно быть “Крамской”, во втором случае “Достоевский”, но т.к. обе фамилии имеют суффикс -ск-, то будут склонены одинаково, и в первом случае именительным падежом будет “Крамский”.

pymorphy.contrib.lastnames_ru.decline(lastname)
Склоняет фамилию и возвращает все возможные формы

pymorphy.contrib.lastnames_ru.normalize(morph, lastname, gender_tag)
Возвращает нормальную форму (именительный падеж) фамилии для заданного рода

```
>>> from pymorphy.contrib import lastnames_ru
>>> print lastnames_ru.normalize(morph, u'СУВОРОВУ', u'мр')
СУВОРОВ
>>> print lastnames_ru.normalize(morph, u'СУВОРОВУ', u'жр')
СУВОРОВА
```

pymorphy.contrib.lastnames_ru.inflect(morph, lastname, gram_form)
Вернуть вариант фамилии который соотвествует данной грамматической форме

Параметры:

- morph - объект Morph
- lastname - фамилия которую хотим склонять
- gram_form - желаемые характеристики грам. формы (если ‘жр’ отсутствует в этом параметре, то по-умолчанию принимается ‘мр’, или ‘мр-жр’, если указано ‘мн’)

pymorphy.contrib.lastnames_ru.get_graminfo(lastname)
Вернуть грамматическую информацию о фамилии и её нормальную форму

pymorphy.contrib.lastnames_ru.pluralize(morph, lastname, gram_form=u‘‘)
Вернуть фамилию во множественном числе.

Параметры:

- morph - объект Morph
- lastname - фамилия которую хотим склонять

•gram_form - желаемые характеристики грам. формы

Если в gram_form явно указать род - это будет использовано как подсказка.

```
>>> print lastnames_ru.pluralize(morph, u'СУВОРОВУ', u'')
СУВОРОВЫМ
>>> print lastnames_ru.pluralize(morph, u'СУВОРОВУ', u'жр')
СУВОРОВЫХ
>>> print lastnames_ru.pluralize(morph, u'СУВОРОВУ', u'жр,дт')
СУВОРОВЫМ
```

В первом случае pluralize() воспринял входную фамилию в мужском роде, дательном падеже (муской род принимается по-умолчанию если не указан женский). Во втором - как женскую фамилию в винительном падеже. В последнем случае подсказка была проигнорирована т.к. требуемый падеж указан явно.

Указания на число ('ед' и 'мн') игнорируются.

pymorphy.contrib.lastnames_ru.pluralize_inflected(morph, lastname, gender_tag, num)

Вернуть фамилию в форме, которая будет сочетаться с переданным числом. Например: 1 Попугаев, 2 Попугаевых, 5 Попугаевых.

Параметры:

•morph - объект Morph

•lastname - фамилия которую хотим склонять

•gender_tag - род фамилии ('мр' или 'жр')

•num - число

Склонение имен людей

Имена, в отличие от фамилий, должны быть в словарях и склоняться стандартным морфологическим анализатором. Тонкость тут в том, что имена часто могут распознаваться еще и как другие части речи, и, чтобы склонение работало правильно, нужно явно указать желаемую часть речи (существительное):

```
>>> print morph.inflect_ru(u'КАТЯ', u'дт')  # катя бочку
КАТЯ
>>> print morph.inflect_ru(u'КАТЯ', u'дт', u'С') # теперь правильно
КАТЕ
```

Как работает pymorphy

3.1 Общая информация

pymorphy - библиотека для морфологического анализа на Python, распространяется по лицензии MIT.

За основу были взяты наработки с сайта aot.ru. Словари (LGPL) для русского и английского, а также идеи - оттуда.

На aot.ru описаны и конкретные алгоритмы реализации, но в терминах теории автоматов. Реализация в pymorphy независимая, не использует конечные автоматы, данные хранятся в key-value хранилище (поддерживаются разные варианты), все алгоритмы переписаны с учетом этого факта.

В pymorphy также есть некоторые возможности, отсутствующие в оригинальной реализации, например:

- поддерживается разбор слов с дефисами, разбор слов со сложными префиксами;
- реализовано склонение слов, постановка слов во множественное число;

3.2 Словари aot.ru

Словари с сайта aot.ru содержат следующую информацию:

1. парадигмы слов и конкретные правила образования;
2. ударения;
3. пользовательские сессии;
4. набор префиксов (продуктивных приставок);
5. леммы (незменяемые части слова, основы);
6. грамматическая информация - в отдельном файле.

Примечание: См. также описание формата mrd-файла

Из этого всего нам интересны правила образования слов, префиксы, леммы и грамматическая информация.

Все слова образуются по одному принципу:

префикс + приставка + основа + окончание

Префиксы Префиксы - это всякие “мега”, “супер” и т.д. Набор префиксов хранится просто списком.

Приставки Имеются в виду приставки, присущие грамматической форме, но не присущие неизменяемой части слова (“по”, “наи”). Например, “наи” в слове “наикрасивейший”, т.к. без превосходной степени будет “красивый”.

Правила образования слов Это то, что надо приписать спереди и сзади основы, чтобы получить какую-то форму. В словаре хранятся пары “приставка - окончание”, + “номер” записи о грамматической информации (которая хранится отдельно).

Парадигмы Правила образования слов объединены в парадигмы. Например, для какого-нибудь класса существительных может быть описано, как слово выглядит во всех падежах и родах. Зная, что существительное принадлежит к этому классу, мы сможем правильно получить любую его форму. Такой класс - это и есть парадигма.

Леммы Леммы - это неизменяемые части слов. В словаре хранится информация о том, какой лемме соответствуют какие парадигмы (какой набор правил для образования грамматических форм слова). Одной лемме может соответствовать несколько парадигм.

Грамматическая информация Грамматическая информация - просто пары (“номер” записи, грам. информация). “Номер” в кавычках, т.к. это 2 буквы, просто от балды, но все разные.

Примечание: Неправильно считать, что лемма - это корень слова, или приставки означают то же самое, что и на уроке русского языка. Привычные со школы категории (корень, суффикс, приставка, окончание) в pymorphy не используются или имеют другой смысл, т.к. эти категории слишком слабо формализованы и поэтому не очень подходят для машинного морфологического анализа.

Файл со словарем - обычный текстовый, для каждого раздела сначала указано число строк в нем, а потом идут строки, формат их описан тут.

Поняв структуру словаря, можно написать первую версию морфологического анализатора.

3.3 Морфологический анализ

По сути, нам дано слово, и его надо найти среди всех разумных комбинаций вида:

префикс + приставка + лемма + окончание

и:

приставка + лемма + окончание

Дело упрощает то, что оказалось (как показала пара строчек на питоне), что “приставок” у нас в языке (да и в английском вроде тоже) всего 2. А префиксов в словаре - порядка 20 для русского языка. Поэтому искать можно среди комбинаций:

префикс + лемма + окончание

объединив в уме список приставок и префиксов, а затем выполнив небольшую проверочку.

Если слово начинается с одного из возможных префиксов, то мы его (префикс) отбрасываем и пытаемся морфологически анализировать остаток (рекурсивно), а потом просто припишем отброшенный префикс к полученным формам.

В итоге получается, что задача сводится к поиску среди комбинаций:

лемма + окончание

Ищем подходящие леммы, потом смотрим, есть ли для них подходящие окончания. [1]

Для поиска задействован стандартный питоновский ассоциативный массив (dict, или любой объект, поддерживающий __getitem__, __setitem__ и __contains__), в который поместил все леммы. Получился словарь вида:

lemmas: {base -> [paradigm_id]}

т.е. ключ - это лемма, а значение - список номеров допустимых парадигм. А дальше поехали - сначала считаем, что лемма - это первая буква слова, потом, что это 2 первых буквы и т.д. По лемме пытаемся получить список парадигм. Если получили, то в каждой допустимой парадигме пробегаем по всем правилам и смотрим, получится ли наше слово, если правило применить. Получается - добавляем его в список найденных форм.

Примечания

3.3.1 Дополнительные детали работы морфологического анализатора

Слова без неизменяемой части

Если вспомнить пример, который был в начале, про “ЛЮДЕЙ” - “ЧЕЛОВЕК”, то станет понятно, что есть слова, у которых неизменяемая часть отсутствует. Выяснилось, что есть в словаре такая хитрая магическая лемма “#”, которая и соответствует всем пустым леммам. Для всех слов нужно искать еще и там.

Склонение слов

Для “склонения” слова (постановке его в определенную грамматическую форму) анализатор сначала составляет список всех форм, в которых может находиться данное слово, потом убирает из них те, которые не соответствуют переданной форме, а потом выбирает из оставшихся вариант, по форме наиболее близкий к исходному.

Постановка слов во множественное число после этого тривиальным образом реализуется через “склонение”.

3.3.2 Предсказатель

Реализован “предсказатель”, который может работать со словами, которых нет в словаре. Это не только неизвестные науке редкие слова, но и просто описки, например.

Для предсказателя реализованы 2 подхода, которые работают совместно.

Первый подход: угадывание префикса

Если слова отличаются только тем, что к одному из них приписано что-то спереди, то, скорее всего, склоняться они будут однаково.

[1] Еще был вариант - составить сразу словарь всех возможных слов вида лемма + окончание, получалось в итоге где-то миллионов 5 слов, не так и много, но вариант, вообщем, мне не очень понравился.

Реализуется очень просто: пробуем считать сначала одну первую букву слова префиксом, потом 2 первых буквы и т.д. А то, что осталось, передаем морфологическому анализатору. Ну и делаем это только для не очень длинных префиксов и не очень коротких остатков.

Второй подход: предсказание по концу слова

Если 2 слова оканчиваются одинаково, то и склоняться они, скорее всего, будут одинаково.

Второй подход чуть сложнее в реализации (так-то сильно сложнее, если нужна хорошая реализация)) и "поумнее" в плане предсказаний.

Первая сложность связана с тем, что конец слова может состоять не только из окончания, но и из части леммы. Для простоты тут задействован опять ассоциативный массив (или duck typing-заменитель) с предварительно подготовленными всеми возмоными окончаниями слов (до 5 букв). Их получилось несколько сот тысяч. Ключ массива - конец слова, значение - список возможных правил. Дальше - все как при поиске подходящей леммы, только у слова берем не начало, а 1, 2, 3, 4, 5-буквенные концы, а вместо лемм у нас теперь новый монстромассив.

Вторая сложность - получается много заведомого мусора. Мусор этот отсекается, если учесть, что полученные слова могут быть только существительными, прилагательными, наречиями или глаголами.

Даже после этого у нас остается слишком много не-мусорных правил. Для определенности, для каждой части речи оставляем только самое распространенное правило.

Примечание: Если слово не было предсказано как существительное, хорошо бы добавить вариант с неизменяемым существительным в ед.ч. и.п., но этот участок кода сейчас закомментирован, т.к. на практике он не давал почти никакого улучшения качества разбора при большом числе ложных срабатываний.

3.3.3 Сложные слова

В версии 0.5 появилась поддержка разбора сложных слов, записанных через дефис (например, "ПО-БРАТСКИ" или "ЧЕЛОВЕК-ПАУК").

Поддерживаются слова, образованные 2 способами:

- левая часть - неизменяемая приставка/основа (например, "ИНТЕРНЕТ-МАГАЗИН", "ВОЗДУШНО-КАПЕЛЬНЫЙ". В этом случае форма слова определяется второй частью. Этот случай добавляется в возможные варианты разбора всегда.
- 2 равноправные части, склоняемые вместе (например, "ЧЕЛОВЕК-ПАУК"). Этот случай добавляется в возможные варианты разбора только тогда, когда обе части имеют одинаковую форму (есть варианты разбора первой части, которые совпадают с вариантами разбора второй).

Справка по API

4.1 Поддерживаемые типы хранилищ

4.1.1 Общая информация

Алгоритмы в pymorphy устроены так, что требуют для работы данные в виде словарей и массивов. Изначально pymorphy использовал структуры list и dict для хранения данных о словах и словообразовании.

Это работало быстро, но требовало большого количества оперативной памяти. Поэтому сейчас для того, чтобы не загружать все словари сразу в память, данные берутся из одной из key-value базы данных.

Примечание: Интерфейс доступа к key-value хранилищам при этом остался тем же. Т.е. требование к хранилищу (обертке над хранилищем) - притворяться массивом или словарем, а именно - поддерживать [] и in (__getitem__, __setitem__ и __contains__).

Словари лежат тут: https://bitbucket.org/kmike/pymorphy/downloads/ и называются по формуле <язык>.<тип базы>-<формат данных>.zip.

Предупреждение: Не скачивайте словари вида <язык>.<тип базы>.zip (например, ru.sqlite.zip - без 'json'), эти словари устаревшие: работают медленнее, в sqlite-словарях серьезная ошибка. В разделе для скачивания они пока оставлены в целях совместимости.

4.1.2 Типы хранилищ

SQLite

Файлы со словарями имеют расширение ".sqlite". Набор словарей для русского языка: ru.sqlite-json.zip. Пример подключения:

```
m = get_morph('dicts/ru') # или так: get_morph('dicts/ru', 'sqlite')
```

Преимущество - в совместимости. Не требует установки, кроссплатформенный формат хранения данных. Если какие-то проблемы с использованием других вариантов, можно использовать SQLite. Вариант по умолчанию.

Этот самый медленный вариант.

CDB

Файлы со словарями имеют расширение ".cdb". Набор словарей для русского языка: ru.cdb-json.zip. Пример подключения:

```
m = get_morph('dicts/ru', 'cdb')
```

Это самый быстрый вариант (не считая варианта с полной загрузкой словарей в память).

Установка:

```
$ pip install python-cdb
```

Для установки потребуются установленные средства сборки (gcc, заголовочные файлы питона).

Минус - лицензия GPL. А pymorphy - под лицензией MIT. И я вот не знаю, можно ли вообще его использовать.

Официальный сайт: http://cr.yp.to/cdb.html

Официальный сайт библиотеки для python: http://pilcrow.madison.wi.us/

Автор - D. J. Bernstein.

Shelve

Файлы со словарями имеют расширение ".shelve". Набор словарей для русского языка: ru.shelve-json.zip. Пример подключения:

```
m = get_morph('dicts/ru', 'shelve')
```

Этот вариант потребляет меньше всего оперативной памяти при работе.

Shelve - это включенная в стандартную поставку библиотека, которая предоставляет dict-like доступ к базам данных BSDDB, GDB, BDB и DumbDB.

Минусы - меньшая скорость работы, чем у альтернативных вариантов, и не всегда переносимый формат словарей. Из-за этого нельзя быть уверенным, что скачанный словарь заработает на конкретной машине.

Чтобы вариант гарантированно заработал, может потребоваться переконвертиртация словаря для поддерживаемого формата данных на конкретной машине. Переконвертация осуществляется с помощью скрипта encode_dicts.py, который есть в репозитории (он не устанавливается через easy_install и pip).

Стоит учесть, что файл, размещенный для скачивания, создан, используя BSDDB 4.6. Поэтому для работы нужен python с поддержкой bsddb 4.6. Если поддерживается версия 4.5 (как в debian lenny), то словари можно перевести в более старый формат:

```
$ for f in *.shelve; do db4.6_dump $f | db4.5_load new_$f; done
```

Tokyo Cabinet

Файлы со словарями имеют расширение ".tcb" и ".tch" для Btree+ и Hash-вариантов базы. Наборы словарей для русского языка: ru.tcb.zip и ru.tch.zip.

Примечание: Btree-вариант занимает меньше места, но работает чуть медленнее.

Пример подключения:

m = get_morph('dicts/ru', 'tch')

Наследник BSDDB, BDB, GDBM, QDBM. Обеспечивает хорошую скорость работы и небольшой размер словарей. Лицензия LGPL.

Требует установки tokyocabinet средствами ОС.

debian

```
$ sudo aptitude install tokyocabinet-bin
$ pip install pytc
```

macports

```
$ sudo port install tokyocabinet
$ pip install pytc
```

Официальный сайт: http://1978th.net/tokyocabinet/

4.1.3 Выбор хранилища

- Хочется быстро все попробовать, не заморачиваясь за установку: SQLite.
- Нужна большая скорость: CDB или Tokyo Cabinet.
- Нужна максимальная скорость: используем pickle (осторожно, потребуется 200-300Мб оперативной памяти).
- Очень мало оперативной памяти: Shelve (BSDDB), отключаем кеширование.

Кеширование сильно ускоряет работу и включено по умолчанию, но оно увеличивает потребление памяти в соответствии с тем, сколько разных парадигм и правил было запрошено.

4.2 Обозначения для грамматической информации в pymorphy

Реализовано 2 формата выдачи результатов: формат по умолчанию и упрощенный стандартизованный формат, согласованный на конференции ДИАЛОГ-2010.

4.2.1 Полный формат

Это формат по умолчанию.

Обозначения соответствуют тем, что описаны тут: http://www.aot.ru/docs/rusmorph.html

При указании в качстве параметров к методам их следует указывать через запятую без пробелов, порядок - произвольный, регистр учитывается:

"им,мр"

Часть речи обычно идет отдельным параметром и не передается в строках с грам. информацией.

Краткая справка по падежам

им Именительный (Кто? Что?)

рд Родительный (Кого? Чего?)

дт Дательный (Кому? Чему?)

вн Винительный (Кого? Что?)

тв Творительный (Кем? Чем?)

пр Предложный (О ком? О чём? и т.п.)

зв Звательный (Его формы используются при обращении к человеку - им. падеж: Аня — звательный: Ань!)

Все используемые граммемы

мр, жр, ср мужской, женский, средний род

од, но одушевленность, неодушевленность

ед, мн единственное, множественное число

им, рд, дт, вн, тв, пр, зв падежи (см. информацию по падежам)

2 обозначает второй родительный или второй предложный падежи

св, нс совершенный, несовершенный вид

пе, нп переходный, непереходный глагол

дст, стр действительный, страдательный залог

нст, прш, буд настоящее, прошедшее, будущее время

пвл повелительная форма глагола

1л, 2л, 3л первое, второе, третье лицо

0 неизменяемое

кр краткость (для прилагательных и причастий)

сравн сравнительная форма (для прилагательных)

имя, фам, отч имя, фамилия, отчество

лок, орг локативность, организация

кач качественное прилагательное

вопр,относ вопросительность и относительность (для наречий)

дфст слово обычно не имеет множественного числа

опч частая опечатка или ошибка

жарг, арх, проф жаргонизм, архаизм, профессионализм

аббр аббревиатура

безл безличный глагол

Части речи

Части речи	Пример	Расшифровка
С	мама	существительное
П	красный	прилагательное
МС	он	местоимение-существительное
Г	идет	глагол в личной форме
ПРИЧАСТИЕ	идущий	причастие
ДЕЕПРИЧАСТИЕ	идя	деепричастие
ИНФИНИТИВ	идти	инфинитив
МС-ПРЕДК	нечего	местоимение-предикатив
МС-П	всякий	местоименное прилагательное
ЧИСЛ	восемь	числительное (количественное)
ЧИСЛ-П	восьмой	порядковое числительное
Н	круто	наречие
ПРЕДК	интересно	предикатив
ПРЕДЛ	под	предлог
СОЮЗ	и	союз
МЕЖД	ой	междометие
ЧАСТ	же, бы	частица
ВВОДН	конечно	вводное слово
КР_ПРИЛ	красива	краткое прилагательное
КР_ПРИЧАСТИЕ	построена	краткое причастие
ПОСЛ		пословица
ФРАЗ		

4.2.2 Упрощенный формат

Данные в этом формате возвращает функция get_graminfo, вызванная с параметром standard=True. Формат был согласован на конференции Диалог-2010.

Примечание: Получение результатов в этом формате НЕ быстрее, чем в полном. Разбор “внутри” все равно идет в “полном” формате, и лишь перед выводом данные преобразуются в упрощенный.

Части речи

Для разметки используется упрощенная система частеи речи:

S существительное (яблоня, лошадь, корпус, вечность)

A прилагательное (коричневыи, таинственныи, морскои)

V глагол (пользоваться, обрабатывать)

PR предлог (под, напротив)

CONJ союз (и, чтобы)

ADV — прочие не няемые слова (частицы, междометия, вводные слова)

Могут быть размечены любым образом:

Местоимения (включая наречные и предикативные)

Числительные

Морфология (грамматические_признаки)

В категориях ADV,PR,CONJ поле остается пустым. Морфология указывается только для S,A,V.

Здесь также используется сокращенныи набор признаков:

род m, f, n

падеж nom, gen, dat, acc, ins, loc

число sg, pl

время/наклонение/причастие/деепричастие pres, past, imper, inf, partcp, ger

залог act, pass

лицо 1p, 2p, 3p

4.3 Морфологический анализатор

Примечание: Этот раздел справки сгенерирован автоматически.

---

pymorphy._morph.get_morph(path, backend='sqlite', cached=True, **kwargs)

Вернуть объект с морфологическим анализатором (Morph).

Параметры:

- path - путь к папке с файлами словарей (или полное имя файла со словарем, в случае pickle)
- backend - тип словарей. Может быть 'shelve', 'tch', 'tcb', 'cdb', 'pickle', 'sqlite'.
- cached - кешировать ли данные в оперативной памяти

Также можно указывать все параметры, которые принимает конструктор класса Morph.

class pymorphy._morph.Morph(data_source, check_prefixes=True, predict_by_prefix=True, predict_by_suffix=True, handle_EE=False)

Класс, реализующий морфологический анализ на основе словарей из data_source

__init__(data_source, check_prefixes=True, predict_by_prefix=True, predict_by_suffix=True, handle_EE=False)

Параметры конструктора:

- data_source: источник данных. Может быть ShelveDataSource, MrdDataSource, PickleDataSource или любой другой наследник DictDataSource, у которого есть атрибуты rules, lemmas, prefixes, gramtab, rule_freq, endings, possible_rule_prefixes, поддерживающие доступ по ключу.
- check_prefixes: проверять ли вообще префиксы
- predict_by_prefix: предсказывать ли по префиксу
- predict_by_suffix: предсказывать ли по суффиксу

- handle_EE: как обрабатывать букву ё. Если True, то все буквы ё считаются равными е, если False - разными буквами. По умолчанию в словарях все буквы ё заменены на е, и ожидается, что в пользовательском вводе букв ё не будет (handle_EE=False). Если словарь сконвертирован с handle_EE=True (в нем сохранены буквы ё), то буквы е и ё будут считаться разными буквами, это может привести к неожиданностям, т.к. часто вместо ё пишут е, а в словаре будет слово только с ё. При handle_EE=True запросы выполняются по 2 раза, со словом "как есть" и со словом, у которого все е заменены на ё.

decline(word, gram_form=u'', gram_class=None)
Вернуть все варианты слова, соответствующие заданной грамматической форме и части речи.

Параметры:

- word - слово, которое хотим склонять
- gram_form - желаемые характеристики грам. формы. Выступает в роли фильтра. Может быть пустым, тогда будут возвращены все формы слова.
- gram_class - часть речи. Если передан этот параметр, то слово будет считаться словом этой части речи (если возможно), и склоняться будет соответственно. Если параметр не передан, ограничения на часть речи накладываться не будут.

get_graminfo(word, standard=False, predict=True, **kwargs)
Вернуть грамматическую информацию о слове и его нормальную форму. Если параметр standard=True, то для каждого варианта разбора результаты возвращаются в стандартном виде (словарь вида {'class': <class>, 'info': <info>, 'norm': <norm>}, обозначения согласованы с теми, что приняты на конференции Диалог-2010. Если standard = False (по умолчанию), то возвращается больше информации (детальное разбиение на части речи, больше морфологических признаков, информация об использованном алгоритме), обозначения определяются структурой словарей.

inflect_ru(word, gram_form, gram_class=None, smart_guess=True)
Вернуть вариант слова, который соотвествует данной грамматической форме и части речи, а также менее всего отличается от исходного.

Параметры:

- word - слово, которое хотим склонять
- gram_form - желаемые характеристики грам. формы
- gram_class - часть речи. Если передан этот параметр, то слово будет считаться словом этой части речи (если возможно), и склоняться будет соответственно. Требуется для устранения неоднозначностей.
- smart_guess (True по умолчанию). Если smart_guess is True, то исходная форма слова будет угадываться как наиболее близкая к нормальной (для существительных будет отдано предпочтение варианту разбора с именительным падежом).

normalize(word)
Вернуть список нормальных форм слова

pluralize_inflected_ru(word, num, gram_class=None)
Вернуть слово в форме, которая будет сочетаться с переданным числом. Например: 1 попугай, 2 попугая, 5 попугаев.

Аналог choose_plural из pytils, для которого требуется только 1 начальная форма слова.

pluralize_ru(word, gram_form=u'', gram_class=None)
Вернуть слово во множественном числе.

4.4 Key-value - бэкенды

Примечание: Этот раздел справки сгенерирован автоматически.

Ниже описаны скорее детали реализации. Чтоб использовать pymorphy, их знать необязательно.

Если вы не планируете участвовать в разработке pymorphy, полезнее ознакомиться со следующим документом: Поддерживаемые типы хранилищ

4.4.1 Базовый класс

class pymorphy.backends.base.DictDataSource
Базовые классы: object

Absctract base class for dictionary data source. Subclasses should make class variables (rules, lemmas, prefixes, gramtab, endings, possible_rule_prefixes) accessible through dict or list syntax ("duck typing")

Абстрактный базовый класс для источников данных pymorphy. У подклассов должны быть свойства rules, lemmas, prefixes, gramtab, endings, possible_rule_prefixes, к которым можно было бы обращаться как к словарям, спискам или множествам.

rules для каждой парадигмы - список правил (приставка, грам. информация, префикс):

{paradigm_id->[(suffix, ancode, prefix)]}

lemmas для каждой леммы - список номеров парадигм (способов образования слов), доступных для данной леммы (основы слова):

{base -> [paradigm_id]}

prefixes фиксированые префиксы:

set([prefix])

gramtab грамматическая информация: словарь, ключи которого - индексы грам. информации (анкоды), значения - кортежи (часть речи, информация о грам. форме, какая-то непонятная буква):

{ancode->(type,info,letter)}

rule_freq частоты для правил, используется при подготовке словарей:

{paradigm_id->freq}

endings для каждого возможного 5 буквенного окончания - словарь, в котором ключи - номера возможных парадигм, а значения - номера возможных правил:

{word_end->{paradigm_id->(possible_paradigm_ids)}}

possible_rule_prefixes набор всех возможных приставок к леммам:

[prefix]

calculate_rule_freq()
Подсчитать частоту, с которой встречаются различные правила. Требуется для предсказателя, чтобы выбирать наиболее распространенные варианты.

convert_and_save(data_obj)
Взять данные из data_obj (наследник DataDictSource) и сохранить из в специфичном для класса формате.

load()
Загрузить данные

4.4.2 DB-источники данных

class pymorphy.backends.shelve_source.ShelveDataSource(path='', db_type=None, cached=True)
Базовые классы: pymorphy.backends.base.DictDataSource

Источник данных для морфологического анализатора pymorphy, берущий информацию из key-value базы данных, используя модифицированный интерфейс shelve из стандартной библиотеки. Позволяет не держать все данные в памяти и в то же время обеспечивает достаточно быструю скорость работы.

Грамматическая информация и префиксы загружаются в память сразу.

convert_and_save(data_obj)

load()

Интерфейс к SQLite

class pymorphy.backends.shelve_source.sqlite_shelve.SqliteDict(filename=None, connection=None, table='shelf')
Базовые классы: object

A dictionary that stores its data in a table in sqlite3 database

clear()

close()

has_key(key)

sync()

class pymorphy.backends.shelve_source.sqlite_shelve.SqliteShelf(filename=None, flag='', key_type='str', dump_method='json', cached=True, connection=None, table='shelf')
Базовые классы: pymorphy.backends.shelve_source.shelf_with_hooks.ShelfWithHooks

close()

Интерфейс к Shelve (BSDDB, GDBM, DBM)

class pymorphy.backends.shelve_source.shelf_with_hooks.ShelfWithHooks(filename, flag, key_type='str', dump_method='json', cached=True, writeback=False)
Базовые классы: shelve.DbfilenameShelf

Shelf class with key and value transform hooks.

close()

get(key, default=None)

has_key(key)

keys()

pymorphy.backends.shelve_source.shelf_with_hooks.json_dumps(value)

Интерфейс к CDB

class pymorphy.backends.shelve_source.cdb_shelve.CdbReadDict(filename)
Базовые классы: object

close()

has_key(key)

class pymorphy.backends.shelve_source.cdb_shelve.CdbShelf(filename, flag, key_type='str', dump_method='json', cached=True, writeback=False)
Базовые классы: pymorphy.backends.shelve_source.shelf_with_hooks.ShelfWithHooks

close()

class pymorphy.backends.shelve_source.cdb_shelve.CdbWriteDict(filename)
Базовые классы: object

close()

Интерфейс к Tokyo Cabinet

4.4.3 Бэкенд для разбора исходных MRD-файлов

Алгоритм работы с ним "как есть" должен быть совсем не таким, как в pymorphy, pymorphy с исходными MRD-файлами работает крайне неэффективно.

Этот бэкенд используется только для переконвертации исходных словарей.

class pymorphy.backends.mrd_source.MrdDataSource(dict_name, gramtab_name, strip_EE=True)
Базовые классы: pymorphy.backends.base.DictDataSource

Источник данных для морфологического анализатора pymorphy, берущий информацию из оригинальных mrd-файлов (в которых кодировка была изменена с 1251 на utf-8). Используется для конвертации оригинальных данных в простые для обработки ShelveDict или PickledDict.

load()

static setup_psyco()
Оптимизировать узкие места в MrdDataSource с помощью psyco

4.4.4 Pickle-источник данных

class pymorphy.backends.pickle_source.PickleDataSource(file)
Базовые классы: pymorphy.backends.base.DictDataSource

Источник данных для морфологического анализатора pymorphy, берущий информацию из файлов, куда с помощью pickle были сохранены данные. Самый быстрый, но ест уйму памяти (> 100 MB).

convert_and_save(data_obj)

load()

4.5 Описание оригинальной библиотеки с aot.ru (может оказаться полезным)

Предупреждение: The following text is a description of the original aot.ru POS tagger implementation. Pymorphy was based on aot.ru research and it uses aot.ru dictionaries, but the implementation and further development is independent. The following text is provided only for reference. It does not describe how pymorphy works.

This is a program of morphological analysis (Russian, German, and English languages).

This program is distributed under the Library GNU Public Licence, which is in the file COPYING.

This program was written by Andrey Putrin, Alexey Sokirko. The project started in Moscow in Dialing Company (Russian and English language). The German part was created at Berlin-Brandenburg Academy of Sciences and Humanities in Berlin (the project DWDS).

The Russian lexicon is based upon Zaliznyak's Dictionary . The German lexicon is based upon Morphy system (http://www-psycho.uni-paderborn.de/lezius/). The English lexicon is based upon Wordnet.

The project uses a regular expression library "PCRE" (Perl Compatible Regular Expressions). We test compilation only with version 6.4. Other versions were not tested. One should download this version from the official site and install it to the default place. If you do not want to install it or you do not have enough rights to do it, then you should create two environment variables:

1. RML_PCRE_LIB, that points to PCRE library directory, where libpcre.a and libpcrecpp.a should be located, for example:

export RML_PCRE_LIB=~/RML/contrib/pcre-6.4/.libs

2. RML_PCRE_INCLUDE, that points to PCRE include catalog, where "pcrecpp.h" is located, for example:

export RML_PCRE_INCLUDE=~/RML/contrib/pcre-6.4

The system has been developed under Windows 2000 (MS VS 6.0), but has also been compiled and run under Linux(GCC). It should work with minor changes on other systems.

Website of DDC: www.aot.ru, https://sf.net/projects/morph-lexicon/

I compiled all sources with gcc 3.2. Lower versions are not supported.

Contents of the this source archive

1. The main morphological library (Source/LemmatizerLib).
2. Library for grammatical codes (Source/AgrgamtabLib).
3. Test morphological program (Source/TestLem)..
4. Library for working with text version of the dictionaries (Source/MorphWizardLib).
5. Generator of morphological prediction base (Source/GenPredIdx).
6. Generator of binary format of the dictionaries (Source/MorphGen).

4.5.1 Installation

Unpacking

- Create a catalog and register a system variable RML, which points to this catalog:

  mkdir /home/sokirko/RML
  export RML=/home/sokirko/RML

- Put “lemmatizer.tar.gz”, ”???-src-morph.tar.gz” to this catalog, ”???” can be “rus”, “ger” or “eng” according to what you have downloaded. Unpack it:

  tar xfz lemmatizer.tar.gz
  tar xfz ???-src-morph.tar.gz

Compiling morphology

0. Do not forget to set RML_PCRE (see above)
1. cd $RML
2. ./compile_morph.sh This step should create all libraries and a test program $RMLBinTestLem.

Building Morphological Dictionary

1. cd $RML
2. ./generate_morph_bin.sh <lang> where <lang> can be Russian, German according to the dictionary yo have downloaded.

   The script should terminate with message “Everything is OK”. You can test the morphology:

   $RML\Bin\TestLem <lang>

If something goes wrong, write me to sokirko@yandex.ru.

4.5.2 MRD-file

This section describes the format of a mrd-file. Mrd-file is a text file which contains one morphological dictionary for one natural language. MRD is an abbreviation of “morphological dictionary”.

The usual place for this file is:

$RML/Dicts/SrcMorph/xxxSrc/morphs.mrd

where xxx can be “Eng”, “Rus” or “Ger” depending on the language.

The encoding of the file depends also upon the language:

- Russian - Windows 1251
- German - Windows 1252
- English - ASCII

4.5.3 Gramtab-files

A mrd-file refers to a gramtab-file, which is language-dependent and which contains all possible full morphological patterns for the words. One line in a gramtab-file looks like as follows:

<ancode> <unused_number> <part_of_speech> <grammems>

An ancode is an ID, which consists of two letters and which uniquely identifies a morphological pattern. A morphological pattern consists of <part_of_speech> and <grammems>. For example, here is a line from the English gramtab:

te 1 VBE prsa,pl

Here "te" is an ancode, "VBE" is a part of speech, "prsa,pl" are grammems, "1" is the obsolete unused number.

In mrd-files we use ancodes to refer to a morphological pattern.

Here is the list of all gramtab-files:

- Russian - $Rml/Dicts/Morph/rgramtab.tab
- German - $Rml/Dicts/Morph/ggramtab.tab
- English - $Rml/Dicts/Morph/egramtab.tab

4.5.4 Common information

All words in a mrd-file are written in uppercase.

One mrd-file consists of the following sections:

1. Section of flexion and prefix models;
2. Section of accentual models;
3. Section of user sessions;
4. Section of prefix sets;
5. Section of lemmas.

Each section is a set of records, one per line. The number of all records of the section is written in the very beginning of the section at a separate line. For example, here is a possible variant of the section of user sessions:

1
alex;17:10, 13 October 2003;17:12, 13 October 2003

"1" means that this section contains only one record, which is written on the next line, thus this section contains only two lines.

Section of possible flexion and prefix models

Each record of this section is a list of items. Each item describes how one word form in a paradigm should be built. The whole list describes the whole paradigm (a set of word forms with morphological patterns).

The format of one item is the following:

%<flexion>*<ancode>

or:

%<flexion>*<ancode>*<prefix>

where <flexion> is a flexion (a string, which should be added to right of the word base) <prefix> is a prefix (a string, which should be added to left of the word base) <ancode> is an ancode.

Let us consider an example of an English flexion and prefix model:

%F*na%VES*nb

Here we have two items:

1. <flexion> = F; <ancode> = na
2. <flexion> = VES; <ancode> = nb

In order to decipher ancodes we should go the English gramtab-file. There we can find the following lines:

na NOUN narr,sg
nb NOUN narr,pl

If base "lea" would be ascribed to this model, then its paradigm would be the following:

leaf NOUN narr,sg
leaves NOUN narr,pl

It is important, that each word of a morphological dictionary should contain a reference to a line in this section.

Section of possible accentual models

Each record of this section is a comma-delimited list of numbers, where each number is an index of a stressed vowel of a word form(counting from the end). The whole list contains a position for each word form in the paradigm.

If an item of an accentual model of word is equal to 255, then it is undefined, and it means that this word form is unstressed.

Each word in the dictionary should have a reference to an accentual model, even though this model can consist only of empty items.

For one word, the number and the order of items in the accentual model should be equal to the number and the order of items in the flexion and prefix model. For example we can ascribe to word "leaf" with the paradigm:

leaf NOUN narr,sg
leaves NOUN narr,pl

the following accentual model:

2,3

It produces the following accented paradigm:

le'af NOUN narr,sg
le'aves NOUN narr,pl

Section of user section

This is a system section, which contains information about user edit sessions.

Section of prefix sets

Each record of this section is a comma-delimited list of strings, where each string is a prefix, which can be prefixed to the whole word. If a prefix set is ascribed to a word, it means, that the words with these prefixes can also exist in the language. For example, if “leaf” has the prefix set “anti,contra”, it follows the existence of words “antileaf”, “contraleaf”.

A flexion and prefix model can contain also a reference to a prefix, but this prefix is for one separate word form, while a prefix set is ascribed to the whole word paradigm.

Section of lemmas

A record of this section is a space-separated tuple of the following format:

<base> <flex_model_no> <accent_model_no> <session_no> <type_ancode> <prefix_set_no>

where

<base> is a base (a constant part of a word in its paradigm)

<flex_model_no> is an index of a flexion and prefix model

<accent_model_no> is an index of an accentual model

<session_no> is an index of the session, by which the last user edited this word

<type_ancode> is ancode, which is ascribed to the whole word (intended: the common part of grammems in the paradigm) “-” if it is undefined

<prefix_set_no> is an index of a prefix set, or “-” if it is undefined

История изменений

5.1 0.5.6 (2011-09-11)

- в релиз 0.5.5 по ошибке не вошли файлы разборщика фамилий и вошли некоторые ненужные файлы - это исправлено.
- улучшения в разборе фамилий;
- решена проблема со склонением слова “КИЕВ” и других, которые склонятор раньше пытался склонять как существительные в косвенных падежах;

5.2 0.5.5 (2011-08-13)

- улучшена и упрощена документация;
- pymorphy-speedups пересобран с Cython 0.15 (для cdb и sqlite с cache=True это почему-то дает ускорение до 1.5..2x);
- исправлена ошибка в склонении слов (иногда при склонении бралась неосновная форма слова);
- исправлена совместная работа pymorphy-speedups и шаблонных фильтров django (фильтры не работали для lazy-строк, например, SafeString или ugettext_lazy-строк);
- очень экспериментальная поддержка склонения фамилий (отдельно от основного анализатора; API будет изменен в последующих релизах).

5.3 0.5.4 (2011-07-15)

- Убрано предупреждение при одновременном обновлении pymorphy и pymorphy-speedups (например, через файл с зависимостями pip);
- файлы setup.py и скрипт запуска тестов теперь всегда используют “родной” pymorphy, а не установленный в систему.

5.4 0.5.3 (2011-07-15)

- Исправлена ошибка с определением версии pymorphy-speedups;
- вместо pymorphy.split теперь pymorphy.contrib.tokenizers с функциями extract_tokens и extract_words;
- поправлена установка из hg-репозитория для windows.

5.5 0.5.2 (2011-04-09)

- Исправлены ошибки в sqlite-словарях (внимание: скачайте новые словари для обновления);
- представление данных в json теперь компактнее, поэтому при использовании новых словарей должна повыситься скорость работы (особенно при отключенном кешировании);
- ускорение отключается с предупрежденем, если версия pymorphy-speedups не соответствует версии pymorphy;
- исправлены опечатки в документации;
- в тесты включен скрипт разбора "Золотого стандарта" с ДИАЛОГ-2010;
- в скрипт для конвертации словарей добавлена перекрестная проверка их корректности.

5.6 0.5.1 (2011-02-10)

- Sqlite-бэкенд теперь должен работать в многопоточном окружении;
- исправлена ошибка с последовательным применением шаблонных фильтров для django.

5.7 0.5.0 (2010-11-15)

- исправления и дополнения в документации
- для тестов используется unittest2
- поддержка опционального модуля pymorphy-speedups с расширением на Cython (туда также перенесен метод setup_psyco). При использовании pickle-словарей скорость при установке расширения должна увеличиться в 2+ раза. Для других (более медленных) вариантов словарей относительный прирост будет не таким значительным. Осторожно: при установленном модуле все строки должны передаваться как юникодные (в.т.ч. латинские и пустые).
- убрана зависимость от simplejson (но его лучше все равно поставить, т.к. с simplejson работа со всеми словарями, кроме pickle, ускоряется в несколько раз)
- правильное склонение слов во втором предложном, родительном или винительном падежах
- метод pluralize_inflected_ru теперь поддерживает не только существительные
- более правильное разбиение на слова в фильтрах
- Работа со словами, записанными через дефис.
- Поддержка парсинга распознанных текстов (характерные замены букв). Довольно бесполезная штука.

- Убран метод get_normal_forms, т.к. метод get_gram_info и так возвращает для каждого слова нормальную форму.

5.8 0.4.3 (2010-02-06)

Устранены небольшие ошибки.

5.9 0.4.0 (2010-01-07)

Упрощена установка: добавилась поддержка кроссплатформенных словарей в sqlite

5.10 0.3.5 (2009-12-15)

Интеграция с django: добавлены шаблонные фильтры для склонения и согласования слов. Переделаны правила получения нормальных форм слова (переделка ошибочная).

5.11 0.1.0 (2009-12-07)

pymorphy почти полностью переписан, документирован, оформлен как python-пакет и загружен на pypi.python.org

5.12 0.0.1 (2009-01-18)

первая версия, которая после написания была заброшена на год

Глава 6

Указатели и поиск

- modindex
- search

Python Module Index

p