
Canadam 2009 - Sage-words Library

The Sage-words library

by Sébastien Labbé and Arnaud Bergeron

inspired from a Sage worksheet written by Franco Saliola

Canadam 2009

Montréal, May 25th, 2009

Outline

1. This talk's objective
2. A word on Sage
3. An historic of the sage-words library at LaCIM
4. New Design to come
5. Many examples
6. An example on the study of palindrome complexity
7. For more informations

This talk's objectives

- Existence of an open-source library for research in combinatorics on words.
- The community is encourage to use it for their research and improve it.
- Give many examples.
- Explain how to get more informations.

Attention : In 25 minutes, we do **not** have time enough to :

- Explain Python and Sage syntax
- Show many cool possibilities of Sage
- Cython
- sagetex

A word on sage

- Sage is a free open-source mathematics software system licensed under the GPL. It combines the power of many existing open-source packages into a common Python-based interface. Its mission is to create a viable free open source alternative to Magma, Maple, Mathematica and Matlab.
- Started at Harvard in January 2005 by William Stein.
- There are currently 143 contributors in 86 different places from all around the world.
- An introductory talk on sage is usely one hour.

```
m = matrix(2, [1,2,3,4])
```

```
m
```

```
[1 2]
[3 4]
```

```
m.determinant()
```

```
-2
```

```
m.determinant?
```

```
<built-in method determinant of
sage.matrix.matrix_integer_dense.Matrix_integer_dense object at
0xbd6fa4c>
```

```
latex(m)
```

```
\left(\begin{array}{rr}
1 & 2 \\
3 & 4
\end{array}\right)
```

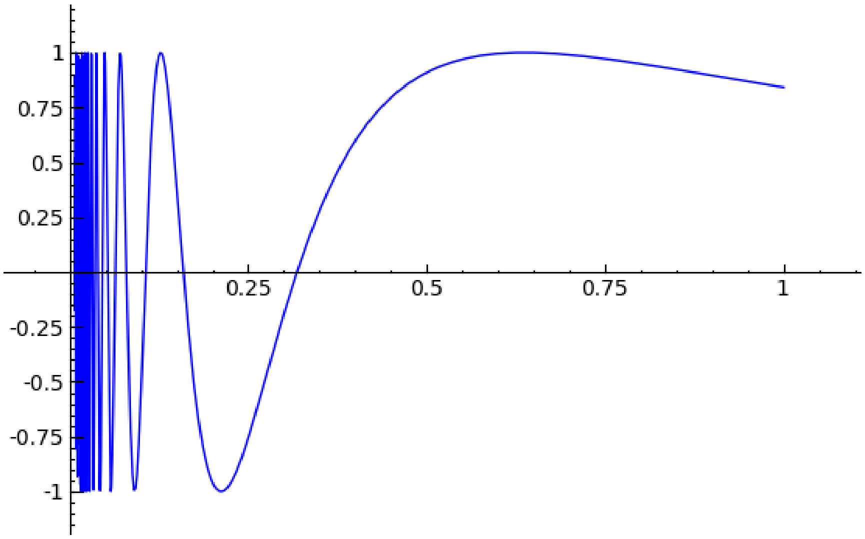
```
m.parent()
```

```
Full MatrixSpace of 2 by 2 dense matrices over Integer Ring
```

Historic of the combinatorics on words library at LaCIM, UQAM

- 1990 : Srecko Brlek developped the idea of a tool to study Combinatorics on words.
- 1995 : Patricia Lamas, a student of Brlek, implemented a set of functions for words in the language Scheme.
- 1996 : Workshop organized in Montreal by Brlek (Valérie Berthé, Julien Cassaigne, Sebastien Ferenczi, Michel Koskas, Dominique Bernardi, Jean Paul Allouche, etc) and discussion about a project named CABAC.
- 1997 : Annie Ladouceur, student supervised by Brlek and supported by Dominique Bernardi, Michel Koskas and Jean-Paul Allouche, rewrote in C a combinatorics on words library.
- 2003 : Xavier Provençal continued the precedent work of A. Ladouceur.
- 2006 : Arnaud Bergeron translated into Ruby language all the work of Annie Ladouceur while correcting many bugs. Thierry Montheil (LIRMM) showed an interest. *This Ruby package didn't have any interface and the lack of friendliness was also blocking its use.*
- Summer 2008: Arnaud Bergeron (Ruby package), Franco Saliola (suffix tree python code), Amy Glen (episturmian code) and Sébastien Labbé (morphisms and palindrome complexity python code) merged their work in what is now called the sage-words project.
- December 2008 : sage-words got merged into sage-3.2.2.The code showed quickly to be slow.
- Spring 2009: Franco Saliola and Sébastien Labbé rewrote the design of sage-words in order to increase its speed. Vincent Delecroix wrote a C datatype that will improve even more the speed of the library. We hope all this to get merged into Sage in the summer.

```
plot(sin(1/x), (x,0,1))
```



New Design (summer 2009)

Goal : Separate the data structures from the mathematical objects.

Mathematical Objects :

- Classes of words
    - Combinatorial class of all words
    - Combinatorial class of all words over a given alphabet
- Words
    - Finite words
    - Infinite words

Data Structures :

- Python lists
- Python string
- Python tuple
- Python functions
- Python iterators
- C++ vector (by Vincent Delecroix, Marseille)

Many examples

Finite words from python strings, lists and tuples.

You can also use the **Word** command to construct a word.

```
Word("abbabaab")
```

```
word: abbabaab
```

```
Word([0,1,1,0,1,0,0,1])
```

```
word: 01101001
```

```
Word( ('a', 0, 5, 7, 'b', 9, 8) )
```

```
word: a057b98
```

Finite words from words.

Words can be concatenated.

```
u = Word("abcccabba")
v = Word([0, 4, 8, 8, 3])
```

```
u * v
```

```
word: abcccabba04883
```

```
v * u
```

```
word: 04883abcccabba
```

```
u + v
```

```
word: abcccabba04883
```

```
u^3 * v^(8/5)
```

```
word: abcccabbaabcccabbaabcccabba04883048
```

Infinite words from finite words.

```
vv = v^Infinity
vv
```

```
word: 0488304883048830488304883048830488304883...
```

Finite words from infinite words.

If you have an infinite word, then you can slice it to get a finite word.

```
vv[10000:10015]
```

```
word: 048830488304883
```

Constructing infinite words.

Infinite words from functions.

An **infinite word** can be described by a function f:\mathbb{N}\rightarrow A that takes values in the alphabet *A*.

```
def f(n):
   return n%3
```

```
u = Word(f)
u
```

```
    word: 0120120120120120120120120120120120120120...
```

```
u[:13]
```

```
    word: 0120120120120
```

```
def t(n):
   return add(Integer(n).digits(base=2)) % 2
```

```
tm = Word(t, alphabet = [0, 1])
tm
```

```
    word: 0110100110010110100101100110100110010110...
```

```
tm[:37]
```

```
    word: 0110100110010110100101100110100110010
```

```
Word(lambda n : add(Integer(n).digits(base=2)) % 2, alphabet = [0, 1])
```

```
    word: 0110100110010110100101100110100110010110...
```

As matrix and many other sage objects, words have a parent.

```
u.parent()
```

```
    Words
```

```
tm.parent()
```

```
    Words over Ordered Alphabet [0, 1]
```

Collection of all words over an alphabet.

To create the collection of all words over an alphabet, use the **Words** command.

```
Words([0,1,2])
```

```
Words over Ordered Alphabet [0, 1, 2]
```

```
A = Words("ab")
A
```

```
Words over Ordered Alphabet ['a', 'b']
```

To create a word in this set, pass data that describes the word.

```
A("abbabaab")
```

```
word: abbabaab
```

```
A(["a","b","b","a","b","a","a","b"])
```

```
word: abbabaab
```

```
W = Words([0,1,2], length=3)
W
```

```
Finite Words of length 3 over Ordered Alphabet [0, 1, 2]
```

```
W.list()
```

```
[word: 000, word: 001, word: 002, word: 010, word: 011, word: 012,
word: 020, word: 021, word: 022, word: 100, word: 101, word: 102,
word: 110, word: 111, word: 112, word: 120, word: 121, word: 122,
word: 200, word: 201, word: 202, word: 210, word: 211, word: 212,
word: 220, word: 221, word: 222]
```

Words from iterators.

Words (finite or infinite) can be constructed using an iterative process.

```
it=iter('abc')
```

```
it.next()
```

```
'a'
```

```
it.next()
```

```
'b'
```

```
it.next()
```

```
'c'
```

```
it.next()
```

```
Traceback (click to the left for traceback)
...
StopIteration
```

```
Word( iter('abbccdef') )
```

```
word: abbccdef
```

```
from itertools import count,repeat
Word( repeat(4) )
```

```
word: 4444444444444444444444444444444444444444...
```

```
Word( count() )
```

```
word:
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,\
26,27,28,29,30,31,32,33,34,35,36,37,38,39,...
```

Infinite words from morphisms.

Let A=\{a,b\} and \mu : A^* \rightarrow A^* be the morphism defined by a\mapsto ab, b\mapsto ba.

```
mu = WordMorphism('a->ab,b->ba'); mu
```

```
Morphism from Words over Ordered Alphabet ['a', 'b'] to Words over
Ordered Alphabet ['a', 'b']
```

```
print mu
```

```
WordMorphism: a->ab, b->ba
```

```
mu('a')
```

```
word: ab
```

```
mu('a', order=2)
```

```
word: abba
```

```
mu('a', order=3)
```

```
word: abbabaab
```

```
mu('a', order=4)
```

```
word: abbabaabbaababba
```

```
mu('a', order=5)
```

```
word: abbabaabbaababbabaababbaabbabaab
```

```
tm = mu('a', order=Infinity)
tm
```

```
Fixed point beginning with 'a' of the morphism WordMorphism:
a->ab, b->ba
```

```
tm[:37]
```

```
word: abbabaabbaababbabaababbaabbabaabbaaba
```

Pre-defined words.

```
words.FibonacciWord()
```

```
Fibonacci word over Ordered Alphabet [0, 1], defined recursively
```

```
words.FibonacciWord("ab")
```

```
Fibonacci word over Ordered Alphabet ['a', 'b'], defined recursively
```

```
words.ThueMorseWord("ab")
```

```
Thue-Morse word over Ordered Alphabet ['a', 'b']
```

```
words.FixedPointOfMorphism(mu,'a')
```

```
Fixed point beginning with 'a' of the morphism WordMorphism:
a->ab, b->ba
```

```
words.ChristoffelWord(7,3,"xy")
```

```
Lower Christoffel word of slope 7/3 over Ordered Alphabet ['x', 'y']
```

```
words.RandomWord(18,5)
```

```
word: 311004213304004223
```

```
Tribonacci = words.StandardEpisturmianWord(Word('abc'))
Tribonacci
```

```
Standard episturmian word over Python objects
```

```
Tribonacci[:40]
```

```
word: abacabaabacababacabaabacabacabaabacababa
```

Interrogating words

```
w = Word('abaabbba'); w
```

```
word: abaabbba
```

```
w.is_palindrome()
```

```
False
```

```
w.is_lyndon()
```

```
False
```

```
print w.lyndon_factorization()
```

```
(ab.aabbb.a)
```

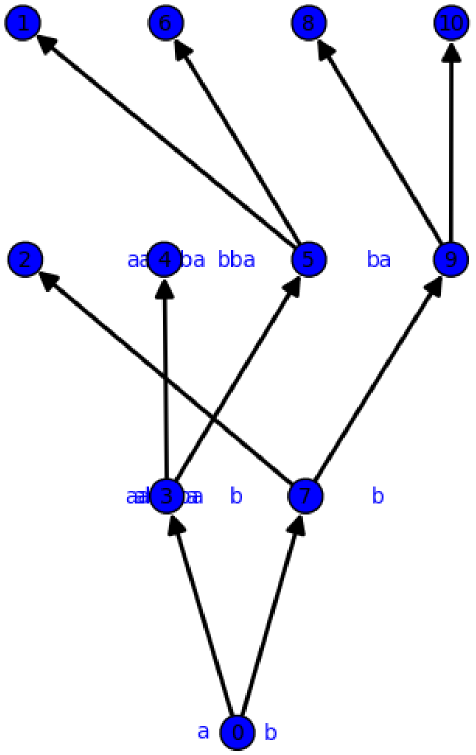
```
print w.crochemore_factorization()
```

```
(a.b.a.ab.bb.a)
```

```
st = w.suffix_tree()
st
```

```
Implicit Suffix Tree of the word: abaabbba
```

```
st.show(word_labels=True)
```



```
w.number_of_factors()
```

```
28
```

```
w.factor_set()
```

```
{word: , word: aba, word: baa, word: b, word: ab, word: bba, word:
ba, word: abbba, word: aabbba, word: baabbba, word: bb, word:
abaabbba, word: a, word: aab, word: baabbb, word: aabb, word: abbb,
word: bbba, word: aa, word: abb, word: baab, word: bbb, word: abaa,
word: baabb, word: aabbb, word: abaabb, word: abaab, word: abaabbb}
```

```
T = words.FibonacciWord('ab')
T.longest_common_prefix(Word('abaababababbbbbb'))
```

```
word: abaababa
```

Currently available commands

For words:

```
w = Word('asodhfa')
```

```
w.[TAB]
```

```
Syntax Error:
    w.[TAB]
```

```
for s in dir(w):
  if not s.startswith("_"):
```

```
    print s
```

```
WARNING: Output truncated!
full_output.txt



BWT
alphabet
apply_morphism
apply_permutation_to_letters
apply_permutation_to_positions
border
category
charge
coerce
colored_vector
commutes_with
complete_return_words
concatenate
conjugate
conjugate_position
conjugates
count
critical_exponent
crochemore_factorization
db
defect
deg_inv_lex_less
deg_lex_less
deg_rev_lex_less
degree
delta
delta_derivate
delta_derivate_left
delta_derivate_right
delta_inv
dump
dumps
evaluation
evaluation_dict
evaluation_partition
evaluation_sparse
exponent
factor_iterator
factor_occurrences_in
factor_set
find
first_pos_in
freq
good_suffix_table
implicit_suffix_tree
inv_lex_less
inversions
is_balanced
is_cadence
is_conjugate_with
```

```
is_cube
is_cube_free
is_empty
is_factor
is_factor_of
is_full
is_lyndon
is_overlap
is_palindrome

...

is_suffix
is_suffix_of
is_symmetric
iterated_left_palindromic_closure
iterated_palindromic_closure
iterated_right_palindromic_closure
lacunas
last_position_dict
last_position_table
length
length_border
lengths_lps
lengths_unioccurrent_lps
letters
lex_greater
lex_less
longest_common_prefix
longest_common_suffix
lps
lyndon_factorization
minimal_period
nb_factor_occurrences_in
nb_subword_occurrences_in
number_of_factors
order
overlap_partition
palindromes
palindromic_closure
palindromic_lacunas_study
parent
parikh_vector
phi
phi_inv
prefix_function_table
primitive
primitive_length
quasiperiods
rename
reset_name
return_words
return_words_derivate
rev_lex_less
reversal
rfind
```

```
save
shifted_shuffle
shuffle
size_of_alphabet
standard_factorization
standard_factorization_of_lyndon_factorization
standard_permutation
string_rep
suffix_tree
suffix_trie
swap
swap_decrease
swap_increase
to_integer_list
to_integer_word
version
```

[full_output.txt]

For classes of words :

```
W = Words('ab')
```

```
W.[TAB]
```

```
Syntax Error:
    W.[TAB]
```

```
W.size_of_alphabet()
```

```
2
```

For morphisms:

```
m = WordMorphism({'a':'ab','b':'ababb'})
```

```
m.[TAB]
```

```
Syntax Error:
    m.[TAB]
```

```
for n in m.list_of_conjugates(): print m
```

```
WordMorphism: a->ab, b->ababb
WordMorphism: a->ab, b->ababb
WordMorphism: a->ab, b->ababb
WordMorphism: a->ab, b->ababb
WordMorphism: a->ab, b->ababb
WordMorphism: a->ab, b->ababb
```

Get help.

```
w = Word('abbbbababab')
```

```
w.[TAB]
```

```
Syntax Error:
```

```
w.[TAB]
```

```
w.number_of_factors?
```

```
44
```

A study of palindrome complexity.

Let w=w_0w_1w_2\cdots be a (finite or infinite) word. Let $Pal(w)$ be the set of all palindromes factors of w. We define the palindrome complexity function of the prefixes of w by

\begin{array}{lcll}g_w : & \mathbb{N}&\rightarrow&\mathbb{N}\\ & i & \mapsto & |Pal(w_0w_1\cdots w_{i-1})| \\ \end{array}.

, i.e. the number of distincts palindromes in the prefix of length i of w. It was shown by Droubay, Justin and Pirillo (2001) that

|Pal(w)| \leq |w|+1.

Then, Brlek, Hamel, Nivat, Reutenauer (200?) defined the defect $D(w)$ of a word w by

D(w)=|w|+1 - |Pal(w)|.

Fixed point of morphisms are divided into four groups according to their palindrome complexity and defect.

```
def palindrome_complexity_function(word):
   r"""
   Returns the palindrome complexity function that given an integer n returns
   the number of palindrome of the prefix of length n.
   """
   liste_zero_un = [1]*word.length()
   for lacuna in word.lacunas():
       liste_zero_un[lacuna] = 0
   liste_sum_partielle = [0]
   sum = 0
   for i in liste_zero_un:
       sum += i
       liste_sum_partielle.append(sum)

   return lambda n: liste_sum_partielle[n]

def discrete_plot(f, domain, **kwds):
   r"""
   Returns a discrete plot of the function f.
   """
   return points([(a,f(a)) for a in domain],**kwds)
```

```
thue = palindrome_complexity_function(words.ThueMorseWord()[:1000])
fibo = palindrome_complexity_function(words.FibonacciWord()[:1000])
fix = palindrome_complexity_function(words.FixedPointOfMorphism('a->abb,b->ba','a')
[:1000])
periodic = palindrome_complexity_function((Word('aababbaabbabaa')^Infinity)[:1000])
```

```
%hide
@interact
```

```
def _(length=(10..1000), bound_c=checkbox(default=False, label='Upper bound complexity
(red)'), thue_c=checkbox(default=True, label='Thue-Morse word (blue)'),
fibo_c=checkbox(default=False, label='Fibonacci word (green)'),
fix_c=checkbox(default=False, label='FixPt of a->abb, b->ba (orange)'),
per_c=checkbox(default=False, label='Periodic word (black)')):
   rep = None
   if bound_c:
       upper_bound = line([(0,0),(length,length)], rgbcolor='red')
       rep = upper_bound if rep is None else rep + upper_bound
   if fibo_c:
       p = discrete_plot(fibo, range(length), rgbcolor='green' )
       rep = p if rep is None else rep + p
   if thue_c:
       p = discrete_plot(thue, range(length),rgbcolor='blue' )
       rep = p if rep is None else rep + p
   if fix_c:
       p = discrete_plot(fix, range(length),rgbcolor='orange' )
       rep = p if rep is None else rep + p
   if per_c:
       p = discrete_plot(periodic, range(length),rgbcolor='black' )
       rep = p if rep is None else rep + p
   show(rep)
```

length

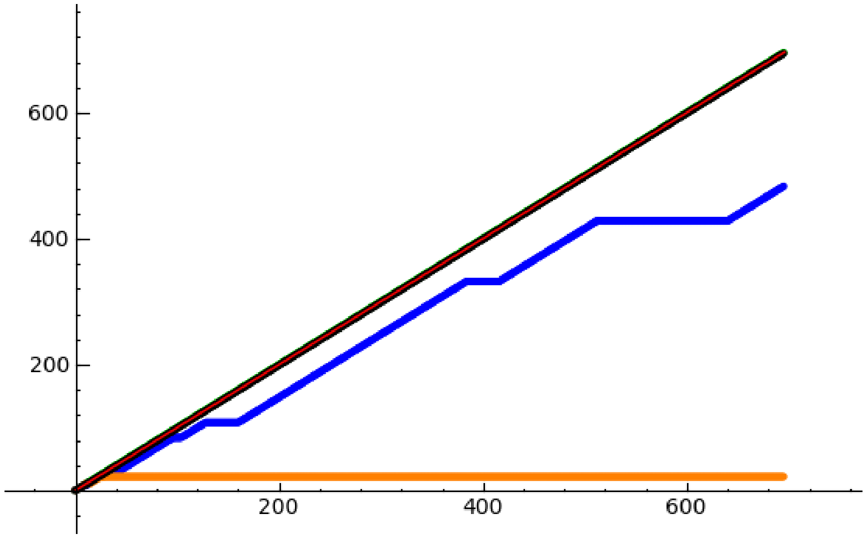
Upper bound complexity (red) ☐

Thue-Morse word (blue) ☑

Fibonacci word (green) ☐

FixPt of a->abb, b->ba (orange) ☐

Periodic word (black) ☐



For more informations.

On Sage:

- http://www.sagemath.org/
- http://wiki.sagemath.org/Talks

On Sage-Combinat:

- http://wiki.sagemath.org/combinat/

Next conferences:

- July 25-29, 2009: *-Combinat 2009. An International Sage Workshop on Free and Practical Software for Algebraic Combinatorics at RISC, Linz, Austria, right after FPSAC'09

- February 22-26, 2010: Sage days. The thematic month MathInfo 2010 at CIRM, Marseille will include a Sage days session. FlorentHivert, NicolasThiéry, and FrancoSaliola will be among the organizers, there will be a serious combinatorics slant.