FpSemigroup Python Bindings

Luke Elliott

April 21, 2017

This project connects Python 3.5 to the already existing libsemigroups C++ library. Libsemigroups allows users to compute, manipulate and find information about semigroups in a variety of ways. The library is also used in the GAP semigroups package. Both versions can be found here on github: https://github.com/james-d-mitchell/libsemigroups.

1 Maths

Definition 1.1 A *semigroup* is a set together with a closed and associative binary operation.

Definition 1.2 A *finitely presented semigroup (FpSemigroup)* is a semigroup defined by a finite set of generators (an *alphabet*) and a finite set of equivalence relations. This gives rise to a semigroup in which the elements are representatives of the equivalence classes of all lists of generators (*words*) under the given relations and the binary operation is concatenation of lists.

Example 1.3 The finite presentation $\langle\ a,\ b \mid a = aa,\ b = bb,\ ab = ba\ \rangle$ corresponds the the semigroup with elements a, b and ab where:

$$\begin{aligned}
a * a &= aa = a \\
a * b &= ab \\
a * ab &= a(ab) = (aa)b = ab \\
b * a &= ba = ab \\
b * b &= bb = b \\
b * ab &= (ba)b = (ab)b = a(bb) = ab \\
ab * a &= a(ba) = a(ab) = (aa)b = ab \\
ab * b &= abb = ab \\
ab * ab &= a(ba)b = a(ab)b = (aa)(bb) = ab
\end{aligned}$$

Example 1.4 The finite presentation $\langle\ a \mid \emptyset\ \rangle$ is merely a list of a's of any positive length. (This is clearly isomorphic to $(\mathbb{N}, +)$)

Definition 1.5 A *monoid* is a semigroup with an identity element.

Definition 1.6 A *finitely presented monoid (FpMonoid)* is defined similarly to a finitely presented semigroup but with the addition of the empty word element. Since the operation is concatenation, this is clearly the identity.

Example 1.7 The finite presentation $\langle\ a \mid \emptyset\ \rangle$ is a list of a's of any non-negative length. This is clearly isomorphic to $(\mathbb{N}_0, +)$.

Example 1.8 The finite presentation $\langle\ \emptyset \mid \emptyset\ \rangle$ contains only the empty word so is isomorphic to the trivial monoid.

2 Programming

In my project I have written code in the Cython programming language (which uses a combination of standard C++ and Python syntax) so that the already existing functions in libsemigroups can be accessed in the Python 3.5 programming language.

In doing this I have converted between input methods, converted between Python and C++ variable types, written various error catches throughout the code and written unit tests to check that my code works on various examples and to check that my error catches execute when I expect them to.

For example, it was necessary to check that user input was in the desired format, and I had to account for any conceivable way of entering the incorrect format. This means that whenever the user makes an error, there will be a message explaining what was wrong with their input format. For example, in the initialisation of semigroups and monoids, the input format of generators required for the C++ library was a vector of pairs of vectors of uint_64s, whereas my Python class takes a list of lists of strings or a list of lists of lists of integers.

Most of my work is in the semigroups.pyx, semigroups_cpp.pxd and FpSemigroup.test.py files:

semigroups.pyx: This file contains the majority of the code including the declaration of all Python classes and member functions.

semigroups_cpp.pxd: This file is used to declare the imported C++ classes and methods called in cython from semigroups.pyx.

FpSemigroup.test.py: This file contains the unit tests which check that all classes and member functions work and respond appropriately when the user makes input mistakes.

I have written the following classes and member functions:

FpSemigroup:
__init__: initializes a semigroup with an alphabet and generators
size: computes the size of the semigroup
is_finite: attempts to check if a given FpSemigroup is finite (not always possible)
set_report: toggles whether or not to report data when running certain functions
set_max_threads: sets the maximum number of threads to be used at once.
alphabet: returns the alphabet of the finitely presented semigroup
is_confluent: checks if the relations of the finitely presented semigroup are confulent.
word_to_class_index: returns the class index of a given word.
__dealloc__: deletes C++ objects when semigroup is deleted
__repr__: displays the finitely presented semigroup via it's generators and relations

FpMonoid:
__init__: Constructs a finitely presented monoid from generators and relations.
alphabet: returns the alphabet of the finitely presented monoid (including the identity)
__repr__: displays the finitely presented monoid via it's generators and relations.

FpSemigroupElement:
__init__: Constructs a finitely presented semigroup element from a finitely presented semigroup and a word.
semigroup: returns the semigroup of which this element is a member.
word: returns the word representing the element as a string or list.
mul: multiplies 2 finitely presented semigroup elements
__repr__: displays the finitely presented semigroup element
word_to_class_index: returns the class index of the finitely presented semigroup element
__dealloc__: deletes C++ objects when a finitely presented semigroup element is deleted.

FpMonoidElement:
__init__: Constructs a finitely presented monoid element from an finitely presented monoid and a word.
monoid: returns the monoid of which this element is a member.
__repr__: displays the finitely presented monoid element.

```
Examples of each class being initiated:
    >>> FpSemigroup(["a","b"],[["aa","a"],["bbb","ab"],["ab","ba"]])
    <Fpsemigroup <a,b|aa=a,bbb=ab,ab=ba>>
    >>> FpSemigroup([1,2],[[[1,1],[1]],[[2,2,2],[1,2]],[[1,2],[2,1]]])
    <Fpsemigroup <1,2|11=1,222=12,12=21>>
    >>> FpMonoid(["a","b"],[["aa","a"],["bbb","ab"],["ab","ba"]])
    <FpMonoid <a,b|aa=a,bbb=ab,ab=ba>>
    >>> FpMonoid([1,2],[[[1,1],[1]],[[2,2,2],[1,2]],[[1,2],[2,1]]])
    <FpMonoid <1,2|11=1,222=12,12=21>>
    >>> FpS=FpSemigroup(["a","b"],[["aa","a"],["bbb","ab"],["ab","ba"]])
    >>> FpSemigroupElement(FpS,"a")
    <FpSemigroup Element 'a'>
    >>> FpSemigroupElement(FpS,"ab")
    <FpSemigroup Element 'ab'>
    >>> FpS=FpSemigroup([1,2],[[[1,1],[1]],[[2,2,2],[2]],[[1,2],[2,1]]])
    >>> FpSemigroupElement(FpS,[1])
    <FpSemigroup Element '[1]'>
    >>> FpSemigroupElement(FpS,[1,2])
    <FpSemigroup Element '[1, 2]'>
    >>> FpM=FpMonoid(["a","b"],[["aa","a"],["bbb","ab"],["ab","ba"]])
    >>> FpMonoidElement(FpM,"a")
    <FpMonoid Element 'a'>
    >>> FpMonoidElement(FpM,"ab")
    <FpMonoid Element 'ab'>
    >>> FpM=FpMonoid([1,2],[[[1,1],[1]],[[2,2,2],[2]],[[1,2],[2,1]]])
```

```
>>> FpMonoidElement(FpM,[1])
<FpMonoid Element '[1]'>
>>> FpMonoidElement(FpM,[1,2])
<FpMonoid Element '[1, 2]'>
```

For instruction on how to install see README.rst in the python-bindings folder.