*FPSAC 2013, Paris, France* DMTCS proc. **(subm.)**, by the authors, 1–12

# *Experimentation with Combinatorial Hopf Algebras in Sage*

Rémi MAURICE[1] and Jean-Baptiste PRIEZ[2]

[1] `maurice@univ-mlv.fr`
*Univ. Paris Est Marne-La-Vallée, Laboratoire d'Informatique Gaspard Monge, Cité Descartes, Bât Copernic – 5, bd Descartes Champs sur Marne 77454 Marne-la-Vallée Cedex 2, France*
[2] `jean-baptiste.priez@lri.fr`
*Univ. Paris-Sud XI, Laboratoire de Recherche en Informatique, PCRI Bât 650, Rue Noetzlin, 91190 Gif-sur-Yvette*

**Abstract.** We present a `Sage` patch for computing with Combinatorial Hopf algebras. It includes a new *Category* (pattern), several concrete Hopf algebras: **FQSym**, **WQSym**, **PQSym**, **PBT**, **FSym**, and easily allows implementation of other ones.

**Résumé.** Nous présentons un patch `Sage` permettant le calcul sur des algèbres de Hopf combinatoires. Il inclut une nouvelle *Category* (ou pattern) ainsi que plusieurs algèbres de Hopf concrètes: **FQSym**, **WQSym**, **PQSym**, **PBT**, **FSym**. De plus, ce patch permet d'en implémenter facilement d'autres.

**Keywords:** Combinatorial Hopf algebras, Computer algebra experimentation

## 1 Introduction

There have been some attempts to implement combinatorial Hopf algebras in various computer algebra systems [13, 1, 16, 15]. However, implementation of such combinatorial and algebraic structures requires many specific features from the targeted software or language. If elements of an algebra are in general linear combinations (in computer memory) of some basis elements, a Hopf algebra is a unique object that one want to implement with different flavors: a combinatorial Hopf algebra is a single object (abstract, *a priori*) with different explicit basis.

Developing a complete and general framework for combinatorial Hopf algebra has been a long-time goal for the *-combinat project [13, 6]. The new design of `Sage-combinat` has been modeled with this aim. Most of the basic combinatorial families of objects are already implemented (permutations, partitions, combinations, integer vectors, ...). `Sage`, whose language is based on `Python`, provides a hierarchy of abstract classes (aka the category framework dues to N. M. THIÉRY) to model advanced algebraic structures, together with the morphisms between them (aka the coercion model). The choice of the platform `Sage` was particularly motivated by the availability of the following features: tensor products, cartesian products, linear maps, set of linear combinations indexed by anything, matrices indexed by anything, abstract algebras with multiple practical realizations which all can be used in a wild generality.

Several different projects are currently trying to enhance `Sage` with some Hopf algebras. Symmetric functions provide a nice example in a relatively final form (J. BANDLOW, M. HANSEN, A. SCHILLING, N. M. THIÉRY, M. ZABROCKI). These abstract algebras have, in `Sage` version 5.6, 5 different bases implemented (*monomials, Schur, powersum, homogeneous, elementary*) with transformations morphisms for each pair of them. On the other side, J. BANDLOW, C. BERG and F. SALIOLA, N. THIÉRY are currently developing an implementation for the pair **NCSF/** $QSym$ (*Non Commutative Symmetric Functions / Quasi-Symmetric Functions*). All these different teams are currently gathering their implementations on the sage-combinat project to try to set up a proper design of code which will allow to factorize the generic code and be also compatible with each of these classes of algebras.

We present our contribution to the design of the implementation of combinatorial Hopf algebras. We improved the existing graded Hopf algebra category; and we implemented two news categories for the definition and computation of combinatorial Hopf algebras and bidendriform combinatorial Hopf algebras [3]. Several combinatorial Hopf algebras are implemented namely **FQSym** [9, 2], **WQSym** [10], **PQSym** [11], **PBT** [8, 4, 5] and **FSym** [7, 12]. In this paper, we propose to explain how to implement a combinatorial Hopf algebras and we illustrate the various tools provided by it by focusing on the example of **FQSym**.

This paper is illustrated with some examples of computations using the `Sage` console and has been automatically generated with the `Sagetex` package.

# 2 Combinatorial Hopf algebras

A combinatorial Hopf algebra is above all a module or a vector space whose bases are labelled by combinatorial objects. `Sage` contains severals classical combinatorial objects like

- several kind of trees due to F. HIVERT, F. CHAPOTON, *al.*;
- Alternating sign matrices due to M. HANSEN and *al.*;
- Dyck words due to M. HANSEN and *al.*;
- Integer Compositions due to N. M. THIÉRY and M. HANSEN;
- $k$-tableaux, Lyndon words, Non decreasing parking functions, Parking functions ...

We will assume that the reader is familiar with this.

## 2.1 General pattern

Given a combinatorial class, we will explain how to implement a combinatorial Hopf algebra associated with it (if that makes sense...) with the following pattern of code:

```
from sage.structure.unique_representation import UniqueRepresentation
from sage.structure.parent import Parent
from sage.misc.bindable_class import BindableClass
from sage.combinat.free_module import CombinatorialFreeModule

class MyCHA(UniqueRepresentation, Parent):
```

```
    def __init__(self, R):
        r"""
        Here, we made the promess that our new object will be a combinatorial
        Hopf algebra with multiples bases.
        """
        from sage.categories.all import Rings
        assert(R in Rings())
        Parent.__init__(self, base=R,
            category=CombinatorialHopfAlgebras(R).WithRealizations())

    def _repr_(self):
        return "Combinatorial Hopf algebra MyCHA over %s" %self.base_ring()

    def basis_keys(self):
        r"""
        Returns the combinatorial class of objects which indices
        the bases of MyCHA.
        """
        return something

    def a_realization(self):
        r"""
        Return an example of a basis of MyCHA.
        """
        return MyFirstBasis

    class MyFirstBasis(CombinatorialFreeModule, BindableClass):
        def __init__(self, MyCHA):
            CombinatorialFreeModule.__init__(self,
                MyCHA.base_ring(), MyCHA.basis_keys(),
                prefix="something", latex_prefix="something",
                bracket=False, category=MyCHA.Bases())

        def product_on_basis(self, cba1, cba2):
            r"""
            Returns the element which is the product of the bases elements
            indexed by ``cba1`` and ``cba2``.
            """
            return something

        def coproduct_on_basis(self, cba):
            r"""
            Returns the element which is the coproduct of the basis element
            indexed by ``cba``.
```

```
            """
            return something

        def dual_basis(self):
            r"""
            Returns the dual of the MyFirstBasis basis.

            If not implemented, it will be automatically computes
            """
            return something

    # another basis
    class OtherBasis(CombinatorialFreeModule, BindableClass):

        def __init__(self, MyCHA):
            CombinatorialFreeModule.__init__(self,
                MyCHA.base_ring(), MyCHA.basis_keys(),
                prefix="something", latex_prefix="something",
                bracket=False, category=MyCHA.Bases())

        # if not implemented, the operations of product and coproduct will
        # be automatically deduce from another basis by coercion (if it can)
        def _build_morphisms(self):
            r"""
            register coercions between different bases whose the one is
            OtherBasis
            """
```

## 2.2 Free quasi-symmetric functions

The combinatorial Hopf algebras of free quasi-symmetric functions **FQSym** [9, 2] is a module whose bases are labelled by permutations. The following code allows to implement this abstract combinatorial Hopf algebra **FQSym**.

```
class FreeQuasiSymmetricFunctions(UniqueRepresentation, Parent):
    def __init__(self, R):
        from sage.categories.all import Rings
        assert(R in Rings())
        Parent.__init__( self, base=R,
            category=CombinatorialHopfAlgebras(R).WithRealizations())

    def _repr_(self):
        cha = "Combinatorial Hopf algebra of Free Quasi-Symmetric Functions "
        return  cha + "over %s" %self.base_ring()
```

```
    def basis_keys(self):
        return Permutations()
```

We can build this abstract combinatorial Hopf algebra **FQSym** in Sage:

```
sage: FQSym = FreeQuasiSymmetricFunctions(ZZ); FQSym
```
Combinatorial Hopf algebra of Free Quasi-Symmetric Functions over Integer Ring

Several bases are implemented in this abstract algebra.

### 2.2.1 *Fundamental basis of* FQSym

The first basis: the fundamental basis. Let us add this new class for the definition of the fundamental basis.

```
class Fundamental(CombinatorialFreeModule, BindableClass):
    def __init__(self, FQSym):
        CombinatorialFreeModule.__init__(self,
            FQSym.base_ring(), FQSym.basis_keys(),
            prefix="F", latex_prefix="{\\bf F}",
            bracket=False, category=FQSym.Bases())

    def product_on_basis(self, sigma1, sigma2):
        from sage.combinat.words.word import Word
        R = self.base_ring()
        d = {self.basis().keys()(list(w)): R(1)
            for w in Word(sigma1).shifted_shuffle(Word(sigma2))}
        return self._from_dict(d)

    def coproduct_on_basis(self, sigma):
        from sage.combinat.permutation import to_standard
        std = to_standard, R = self.base_ring()
        d = {(std(sigma[:i]), std(sigma[i:])): R(1)
            for i in xrange(sigma.size() + 1)}
        return self.tensor_square()._from_dict(d)

# simple shortland to use FreeQuasiSymmetricFunctions(ZZ).F()
F = Fundamental
```

One obtains a first basis that we can explicitely manipulate

```
sage: F = FQSym.F(); F
```
Combinatorial Hopf algebra of Free Quasi-Symmetric Functions over Integer Ring in the Fundamental basis

In this practical realization, one can build an element indexed by a permutation:

```
sage: u = F(Permutation([1, 2])); u
```
$\mathbf{F}_{12}$

In the fundamental basis, the product rule is given by the shifted shuffle of permutations:

```
sage: F([1, 2]) * F([1, 3, 2])
```
$\mathbf{F}_{12354} + \mathbf{F}_{13254} + \mathbf{F}_{13524} + \mathbf{F}_{13542} + \mathbf{F}_{31254} + \mathbf{F}_{31524} + \mathbf{F}_{31542} + \mathbf{F}_{35124} + \mathbf{F}_{35142} + \mathbf{F}_{35412}$

and the coproduct rule is the deconcatenation of words, followed by standardization of all prefixes and suffixes:

```
sage: F([3, 1, 4, 5, 2]).coproduct()
```
$1 \otimes \mathbf{F}_{31452} + \mathbf{F}_{1} \otimes \mathbf{F}_{1342} + \mathbf{F}_{21} \otimes \mathbf{F}_{231} + \mathbf{F}_{213} \otimes \mathbf{F}_{21} + \mathbf{F}_{2134} \otimes \mathbf{F}_{1} + \mathbf{F}_{31452} \otimes 1$

The definition of this operator is extended by linearity and the coproduct is an algebra morphism:

```
sage: u = F([1, 2])
sage: v = F([2, 1])
sage: (u * v).coproduct() == u.coproduct() * v.coproduct()
True
```

The previous computation illustrates the fact that the following diagram commutes:

$$\begin{array}{ccc} \mathbf{FQSym} \otimes \mathbf{FQSym} & \xrightarrow{\mu} & \mathbf{FQSym} \\ \downarrow{\scriptstyle \Delta\otimes\Delta} & & \downarrow{\scriptstyle \Delta} \\ \mathbf{FQSym}^{\otimes 2} \otimes \mathbf{FQSym}^{\otimes 2} & \xrightarrow[\mu^{(2)}]{} & \mathbf{FQSym} \otimes \mathbf{FQSym} \end{array}$$

since **FQSym** is a combinatorial Hopf algebra, where $\mu$ and $\Delta$ are respectively the product and the coproduct defined in the fundamental basis and $\mu^{(2)}$ is defined on pure tensors by

$$\mu^{(2)}\left((x_1 \otimes y_1) \otimes (x_2 \otimes y_2)\right) = \mu(x_1 \otimes x_2) \otimes \mu(y_1 \otimes y_2).$$

The antipode is then automatically deduced from the coproduct:

```
sage: F([1, 3, 4, 2]).antipode()
```
$(-1)\,\mathbf{F}_{2143} + (-1)\,\mathbf{F}_{2413} + (-1)\,\mathbf{F}_{2431} + \mathbf{F}_{3142} + \mathbf{F}_{3214} + \mathbf{F}_{3241} + \mathbf{F}_{3412} + 2\mathbf{F}_{3421} + (-1)\,\mathbf{F}_{4213} + (-1)\,\mathbf{F}_{4231}$

### 2.2.2 *Elementary basis of* **FQSym**

Most basis transformations are expressed by an order relation on objects indexing the bases. The elementary basis, expressed through the fundamental basis, is a sum over an interval of the weak order:

```
class Elementary(CombinatorialFreeModule, BindableClass):
    def __init__(self, FQSym):
        CombinatorialFreeModule.__init__(self,
            FQSym.base_ring(), FQSym.basis_keys(),
            prefix="E", latex_prefix="{\\bf E}",
            bracket=False, category=FQSym.Bases())

    def _build_morphisms(self):
        F = self.realization_of().F(); E = self
        R = self.realization_of().base_ring()
```

```
        SumOfGreater = lambda sigma: F._from_dict({s: R(1)
            for s in sigma.permutohedron_greater()})
        E_to_F = E.module_morphism(on_basis=SumOfGreater,
                category=CombinatorialHopfAlgebras(R),
                codomain=F, triangular="lower", unitriangular=True)
        E_to_F.register_as_coercion()
        (~E_to_F).register_as_coercion()

    def product_on_basis(self, s1, s2):
        n = s1.size()
        R = self.realization_of().base_ring()
        d = {self.basis().keys()(s1[:] + map(lambda t: t+n, s2[:])): R(1)}
        return self._from_dict(d)

E = Elementary
```

sage: `E = FQSym.E(); E`

Combinatorial Hopf algebra of Free Quasi-Symmetric Functions over Integer Ring in the Elementary basis

sage: `F(E([4, 1, 2, 3]))`

$\mathbf{F}_{4123} + \mathbf{F}_{4132} + \mathbf{F}_{4213} + \mathbf{F}_{4231} + \mathbf{F}_{4312} + \mathbf{F}_{4321}$

The definition of the elementary basis ensures that the base change between the elementary basis and the fundamental basis is triangular. With some manipulations of a technical nature, we compute, using the `Sage` labelled matrix framework of A. MATHAS, the transformation matrix from the fundamental basis to the elementary basis restricted in size 3:

| | $[1,2,3]$ | $[1,3,2]$ | $[2,1,3]$ | $[2,3,1]$ | $[3,1,2]$ | $[3,2,1]$ |
|---|---|---|---|---|---|---|
| $[1,2,3]$ | 1 | . | . | . | . | . |
| $[1,3,2]$ | 1 | 1 | . | . | . | . |
| $[2,1,3]$ | 1 | . | 1 | . | . | . |
| $[2,3,1]$ | 1 | . | 1 | 1 | . | . |
| $[3,1,2]$ | 1 | 1 | . | . | 1 | . |
| $[3,2,1]$ | 1 | 1 | 1 | 1 | 1 | 1 |

This basis is a multiplicative basis. Using the product rule in the fundamental basis, we compute the product in the elementary basis:

```
sage: sigma = Permutation([2, 3, 1])
sage: tau = Permutation([4, 2, 1, 3])
sage: E(F(E(sigma)) * F(E(tau)))
```

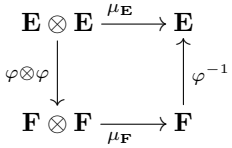
$\mathbf{E}_{2317546}$

This composition is indeed the product in the elementary basis:

```
sage: E(F(E(sigma)) * F(E(tau))) == E(sigma) * E(tau)
```

True

which illustrates the commutativity of the following diagram:



since the basis transformation is an algebra isomorphism, where $\mu_{\mathbf{E}}$ and $\mu_{\mathbf{F}}$ are respectively the product rule in the elementary basis and the fundamental basis and $\varphi$ is the algebra isomorphism from $\mathbf{E}$ to $\mathbf{F}$. The coproduct in the elementary basis is automatically computed by using the coproduct defined in the fundamental basis in the following way:

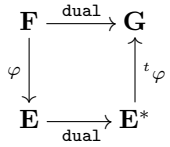
```
sage: tensor((E, E))(F(E([1, 2, 3])).coproduct())
```
$1 \otimes \mathbf{E}_{123} + 3\mathbf{E}_1 \otimes \mathbf{E}_{12} + 3\mathbf{E}_{12} \otimes \mathbf{E}_1 + \mathbf{E}_{123} \otimes 1$

### 2.2.3 *Duality in* $\mathbf{FQSym}$

The combinatorial Hopf algebra $\mathbf{FQSym}$ is self-dual and the dual basis of $\mathbf{F}_\sigma$ is $\mathbf{G}_\sigma$. The code of the dual basis of the fundamental basis is easy. The information that these bases are dual is given to each of them and the product and coproduct rule is induced by the isomorphism $\mathbf{F}_\sigma \mapsto \mathbf{G}_{\sigma^{-1}}$.

```
sage: G = FQSym.G(); G is F.dual_basis()
```
True

For the dual basis of the elementary basis, we are in the following case. The previous bases are dual and we know the isomorphism $\varphi$ from the fundamental basis to the elementary basis. So, the dual basis to the elementary basis is completely determined by $\varphi$:



When the dual bases are not implemented, they are then instantiated by constructing isomorphisms induced by duality:

```
sage: E = FQSym.E()
sage: Edual = E.dual_basis(); Edual
```
Combinatorial Hopf algebra of Free Quasi-Symmetric Functions over Integer Ring expressed in the dual basis to the Elementary basis

```
sage: Edual([2, 1])**2
```
$2\mathbf{E}^*_{2143} + \mathbf{E}^*_{2413} + \mathbf{E}^*_{2431} + \mathbf{E}^*_{3142} + \mathbf{E}^*_{3241} + \mathbf{E}^*_{4132} + \mathbf{E}^*_{4213} + 2\mathbf{E}^*_{4231} + \mathbf{E}^*_{4321}$

```
sage: Edual([2, 1, 4, 3]).coproduct()
```
$1 \otimes \mathbf{E}^*_{2143} + \mathbf{E}^*_{21} \otimes \mathbf{E}^*_{21} + \mathbf{E}^*_{2143} \otimes 1$

In the same way as for the elementary basis, the homogeneous basis is obtained from the fundamental basis as a sum over the weak order. By transitivity, the dual bases to the elementary and homogeneous

bases are isomorphic. Hence, any element expressed in the dual basis to the homogeneous basis can be expressed in the dual basis to the elementary basis:

```
sage: Hdual = H.dual_basis(); Hdual
```

Combinatorial Hopf algebra of Free Quasi-Symmetric Functions over Integer Ring expressed in the dual basis to the Homogeneous basis

```
sage: Edual(Hdual([2, 3, 1]))
```

$(-1)\,\mathbf{E}^*_{132} + (-1)\,\mathbf{E}^*_{312} + (-1)\,\mathbf{E}^*_{321}$

The basis transformation between the elementary and homogeneous bases and the one between the dual bases to the elementary and homogeneous bases are dual. Here is the matrix transformation from $\mathbf{E}$ to $\mathbf{H}$:

| | $[1,2,3]$ | $[1,3,2]$ | $[2,1,3]$ | $[2,3,1]$ | $[3,1,2]$ | $[3,2,1]$ |
|---|---|---|---|---|---|---|
| $[1,2,3]$ | . | . | . | 1 | 1 | 1 |
| $[1,3,2]$ | . | . | . | . | $-1$ | . |
| $[2,1,3]$ | . | . | . | $-1$ | . | . |
| $[2,3,1]$ | . | $-1$ | . | . | $-1$ | $-1$ |
| $[3,1,2]$ | . | . | $-1$ | $-1$ | . | $-1$ |
| $[3,2,1]$ | 1 | 1 | 1 | 1 | 1 | 1 |

Here is the matrix transformation from $\mathbf{H}^*$ to $\mathbf{E}^*$:

| | $[1,2,3]$ | $[1,3,2]$ | $[2,1,3]$ | $[2,3,1]$ | $[3,1,2]$ | $[3,2,1]$ |
|---|---|---|---|---|---|---|
| $[1,2,3]$ | . | . | . | . | . | 1 |
| $[1,3,2]$ | . | . | . | $-1$ | . | 1 |
| $[2,1,3]$ | . | . | . | . | $-1$ | 1 |
| $[2,3,1]$ | 1 | . | $-1$ | . | $-1$ | 1 |
| $[3,1,2]$ | 1 | $-1$ | . | $-1$ | . | 1 |
| $[3,2,1]$ | 1 | . | . | $-1$ | $-1$ | 1 |

which emphasizes the fact that the matrix representing the two bases transformations at each graded component are transposed.

# 3 Status of the code and further development

## *3.1 Status of the code*

Our code is currently constituted by 9 `Python` modules for a total of 4500 lines of code. This number of lines is underestimated as we are adding documentation and tests as the code begin to be final.

We implemented from scratch the combinatorial family of packed words using the `Parent/Element` structures, the `DisjointUnionEnumeratedSets` features and the `SetWithGrading` category. We implemented two new categories to model abstract combinatorial Hopf algebras with realizations and abstract bidendriform combinatorial Hopf algebras with realizations. All generic code and method live in these two modules. Each Hopf algebra ($\mathbf{FQSym}$, $\mathbf{WQSym}$, $\mathbf{PQSym}$, $\mathbf{PBT}$, $\mathbf{FSym}$) is a new module in which the `python` class modeling the abstract algebra inherit (`Python` inheritance) from our two

categories. Products, coproducts and morphisms are always only defined on basis elements and extended on whole algebras by linearity.

The code is available on the `Sage-combinat` server `http://combinat.sagemath.org/`. This document has been automatically generated with the `Sagetex` package (embedding of `Sage` results in tex document). As we are currently using experimental new features, this documents depends on:

- a patch for labeled matrices indexed by any `Sage` objects from A. MATHAS,
- a patch to enhance the coercion between tensor products of combinatorial free module,
- a patch for the category of set with a grading,
- a patch to enhance the category of graded morphism between graded modules with basis,
- a patch modifying the `Sage` permutations adding them a grading.

All of these patches present the technical issues for our code to be ready to be submitted to `Sage`. As all of these points has been largely discussed, we expect a short term integration.

### *3.2 Further improvements*

In combinatorics research, computer exploration is of great importance. Part of the program has been designed in this direction and is a continuation of the `Sage` category framework of N. THIÉRY. It remains to improve a finer management of the morphisms, of the duality in order to, for example, construct on the fly any realization.

## Acknowledgements

We thank N. BORIE initially for all his help and support during the development of this work. We would like to thank F. HIVERT, N. THIÉRY, J.-C. NOVELLI and J.-Y. THIBON to have influenced this work. We would like to thank G. CHÂTEL, S. GIRAUDO, M. JOSUAT-VERGÈS, V. PONS and V. VONG to have encouraged this project.

This research was driven by computer exploration using the open-source mathematical software `Sage` [14]. In particular, we perused its algebraic combinatorics features developed by the `Sage-Combinat` community [13].

## References

[1] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system I: The user language. *J. Symbolic Comput.*, 1997.

[2] G. Duchamp, F. Hivert, and J.-Y. Thibon. Noncommutative Symmetric Functions VI: Free Quasi-Symmetric Functions and Related Algebras. *Int. J. of Algebra and Computation*, 12:671–717, 2002.

[3] L. Foissy. Bidendriform bialgebras, trees, and free quasi-symmetric functions. *J. Pure Appl. Algebra*, 209(2):439–459, 2007.

[4] F. Hivert, J.-C. Novelli, and J.-Y. Thibon. An analogue of the plactic monoid for binary search trees. *Comptes-Rendus Mathematique*, 335, Number 7:577–580, 2002.

[5] F. Hivert, J.-C. Novelli, and J.-Y. Thibon. The Algebra of Binary Search Trees. *Theor. Comput. Sci.*, 339, Issue 1:129–165, 2005.

[6] Florent Hivert and Nicolas M. Thiéry. MuPAD-Combinat, an open-source package for research in algebraic combinatorics. *Sém. Lothar. Combin.*, 51:Art. B51z, 70 pp. (electronic), 2004. `http://mupad-combinat.sf.net/`.

[7] A. Lascoux and M.-P. Schützenberger. Le monoïde plaxique. In *Noncommutative structures in algebra and geometric combinatorics (Naples, 1978)*, volume 109 of *Quad. "Ricerca Sci."*, pages 129–156. CNR, Rome, 1981.

[8] J.-L. Loday and M. Ronco. Hopf Algebra of the Planar Binary Trees. *Adv. Math.*, 139:293–309, 1998.

[9] C. Malvenuto and C. Reutenauer. Duality between quasi-symmetric functions and the Solomon descent algebra. *J. Algebra*, 177(3):967–982, 1995.

[10] J.-C. Novelli and J.-Y. Thibon. Polynomial realizations of some trialgebras. *FPSAC*, 2006.

[11] J.-C. Novelli and J.-Y. Thibon. Hopf algebras and dendriform structures arising from parking functions. *Fundamenta Mathematicae*, 193:189–241, 2007.

[12] S. Poirier and C. Reutenauer. Algèbres de Hopf de tableaux. *Ann. Sci. Math. Québec*, 19:79–90, 1995.

[13] The Sage-Combinat community. Sage-Combinat: enhancing Sage as a toolbox for computer exploration in algebraic combinatorics, 2013.

[14] W. A. Stein et al. *Sage Mathematics Software (Version 5.7)*. The Sage Development Team, 2013. `http://www.sagemath.org`.

[15] The MuPAD Group, Benno Fuchssteiner et al. *MuPAD User's Manual - MuPAD Version 1.2.2*. John Wiley and sons, Chichester, New York, first edition, march 1996. includes a CD for Apple Macintosh and UNIX.

[16] S. Veigneau. ACE, an Algebraic Combinatorics Environment for the computer algebra system MAPLE: User's Reference Manual, Version 3.0. Report 98–11, IGM, 1998.