Séminaires & Congrès

COLLECTION SMF

IMPLEMENTING GRÖBNER BASES FOR OPERADS

Vladimir Dotsenko & Mikael Vejdemo Johansson

OPERADS 2009

Numéro 26

Jean-Louis Loday & Bruno Vallette, ed.

SOCIÉTE MATHÉMATIQUE DE FRANCE

Séminaires & Congrès
26, 2011, p. 77–97

IMPLEMENTING GRÖBNER BASES FOR OPERADS

by

Vladimir Dotsenko & Mikael Vejdemo Johansson

Abstract. — We present an implementation of the algorithm for computing Gröbner bases for operads due to the first author and A. Khoroshkin. We discuss the actual algorithms, the choices made for the implementation platform and the data representation, and strengths and weaknesses of our approach.

***Résumé* (Comment implémenter les bases de Gröbner pour les opérades).** — Nous décrivons comment implémenter l'algorithme, dû au premier auteur et à A. Khoroshkin, qui calcule les bases de Gröbner pour les opérades. Nous étudions les algorithmes actuels, les choix affectués pour les plateformes d'implémentation et pour la représentation des données. Nous discutons aussi des forces et des faiblesses de notre approche.

1. Introduction

1.1. Summary of results. — In an upcoming paper [4], the first author and Anton Khoroshkin define the concept of a Gröbner basis for finitely presented operads. In that paper, they prove the diamond lemma, and demonstrate that for an operad, having a quadratic Gröbner basis is equivalent to the existence of a Poincaré–Birkhoff–Witt basis. As demonstrated by Eric Hoffbeck [5], an operad with a PBW basis is Koszul. Hence, an implementation of the Gröbner bases algorithm yields, in addition to a framework for exploration of operads by means of explicit calculation, a computer-aided tool for proving Koszulness.

In this paper, we present an implementation of the Gröbner basis algorithm in the Haskell programming language [10]. Being designed with categorical terms, Haskell provides a powerful framework for algorithms like that. What we end up with is a computer sofware package which allows to compute the Gröbner basis for a finitely presented operad, as well as bases and dimensions for components of such an operad.

2000 Mathematics Subject Classification. — Primary: 68W30; Secondary: 16S15, 18D50, 68N18.
Key words and phrases. — Buchberger algorithm, Gröbner bases, Haskell, shuffle operads.

One of the main goals of this paper is to help mathematicians who want to get familiar with this software package and use it for their needs, including changing some algorithms or adding more functionality.[1] Consequently, this is more of an invitation to experiment with this software than a report on what it is possible to compute. Let us comment briefly on the state of the art regarding computations. While working on the package, we have implemented several well known operads to test the performance. In the case when an operad is PBW, our package captures that right away. This already is a very important achievement: having implemented many different admissible orderings, one can check very fast whether or not an operad is PBW for at least one of them, thus proving the Koszulness in many cases. Note that the PBW property depends a lot not only on the choice of an admissible ordering, but also on the choice of ordering of generators of our operad; for example, for the operad of pre-Lie algebras, depending on the ordering, a Gröbner basis can vary from quadratic to seemingly infinite. On the other hand, for operads that do not have a quadratic Gröbner basis, we encountered subtle performance issues in many cases. For operads having a relatively small finite Gröbner basis, like the fake commutative operad AntiCom [4], the computation easily yields the correct result, while for many other cases, like the pre-Lie operad for a "wrong" ordering, computations with arity 6 and further take enormously long.

The actual implementation is distributed through the HackageDB repository for Haskell software projects at

`http://hackage.haskell.org/package/Operads`,

software distributed through this repository are available through the automated installation tool `cabal-install`.

The current documentation files are kept online at

`http://math.stanford.edu/~mik/operads/`.

1.2. Outline of the paper. — The paper is organized as follows. In Section 2, we recall relevant background information related to operads and Gröbner bases, on one hand, and to types and functions in Haskell, on the other hand. In Section 3, we discuss the way we chose to represent our data in Haskell. In Section 4, we present algorithms used in our implementation. Finally, in the appendix, we list Haskell constructions used throughout the paper.

1.3. Acknowledgements. — We wish to express our deep gratitude to Eric Hoffbeck and Henrik Strohmayer for both significant assistance in the construction of the software code, and analysis of the techniques we are using. Some of the hairier points of Haskell evaluation has been rendered clear by the helpful assistance of the many members of the `#haskell` IRC channel on the Freenode IRC network.

[1] The first author is a living example proving that it is possible; having been introduced to Haskell by the second author in the process of working on this package, he now has enough confidence to not only use the package, but to add new functions as well.

The first author was supported by an IRCSET research fellowship. The second author was supported by the Office of Naval Research, through grant N00014-08-1-0931.

We are grateful to Jean–Louis Loday and Bruno Vallette who organized the "Operads 2009" meeting in CIRM Luminy, where the work on this project was started. The second author wishes to thank Dublin Institute for Advanced Studies which hosted him as a visitor during the last stage of working on this paper.

2. Overview

For exhaustive information on symmetric operads, we refer the reader to monographs [8] and [9]. Here, we mainly concentrate on shuffle operads and their relationship with symmetric operads, and definitions in the symmetric case are chosen in the way that best suits this approach.

2.1. Operads. — We denote by Ord the category of nonempty finite ordered sets (with order-preserving bijections as morphisms), and by Fin — the category of nonempty finite sets (with bijections as morphisms). Also, we denote by Vect the category of vector spaces (with linear operators as morphisms; unlike the first two cases, we do not require a map to be invertible).

Definition 1. — 1. A *(nonsymmetric) collection* is a contravariant functor from the category Ord to the category Vect.
2. A *symmetric collection* (or an $\mathbb{S}$-*module*) is a contravariant functor from the category Fin to the category Vect.

For either type of collections, we can consider the category whose objects are collections of this type (and morphisms are morphisms of the corresponding functors). The natural forgetful functor ${}^f\colon$ Ord $\to$ Fin, $I \mapsto I^f$ leads to a forgetful functor ^{f} from the category of symmetric collections to the category of nonsymmetric ones, $\mathscr{P}^f(I) := \mathscr{P}(I^f)$. For simplicity, we let $\mathscr{P}(k) := \mathscr{P}([k])$.

We use the convention $[k] = \{1, 2, \dots, k\}$ in this paper.

Definition 2. — – Let $\mathscr{P}$ and $\mathscr{Q}$ be two nonsymmetric collections. Define their *shuffle composition* $\mathscr{P} \circ_{sh} \mathscr{Q}$ by the formula

$$(\mathscr{P} \circ_{sh} \mathscr{Q})(I) := \bigoplus_k \mathscr{P}(k) \otimes \left(\bigoplus_{f\colon I \twoheadrightarrow [k]} \mathscr{Q}(f^{-1}(1)) \otimes \cdots \otimes \mathscr{Q}(f^{-1}(k)) \right),$$

where the sum is taken over all shuffling surjections f, that is surjections for which $\min f^{-1}(i) < \min f^{-1}(j)$ whenever $i < j$.

– Let $\mathscr{P}$ and $\mathscr{Q}$ be two symmetric collections. Define their *(symmetric) composition* $\mathscr{P}\circ\mathscr{Q}$ by the formula

$$(\mathscr{P}\circ\mathscr{Q})(I) := \bigoplus_k \mathscr{P}(k)\otimes_{S_k}\left(\bigoplus_{f\colon I\twoheadrightarrow[k]}\mathscr{Q}(f^{-1}(1))\otimes\cdots\otimes\mathscr{Q}(f^{-1}(k))\right),$$

where the sum is taken over all surjections f.

Each of these compositions gives a structure of a monoidal category on the category of the corresponding collections. The same definitions can be given if we replace Vect by another symmetric monoidal category. For our purposes, an important replacement for Vect will be the category of finite sets (with arbitrary mappings as morphisms).

Definition 3. — 1. A *shuffle operad* is a monoid in the category of nonsymmetric collections with the monoidal structure given by the shuffle composition.
2. A *symmetric operad* is a monoid in the category of symmetric collections with the monoidal structure given by the (symmetric) composition.

Definition 4. — A *shuffle permutation* of the type $(k_1,\dots,k_n)$ is a permutation in the symmetric group $S_{k_1+\dots+k_n}$ which preserves the order of the first k_1 elements, the second k_2 elements,..., the last k_n elements, and satisfies

$$\sigma(1)<\sigma(k_1+1)<\sigma(k_1+k_2+1)<\cdots<\sigma(k_1+\cdots+k_{n-1}+1).$$

Proposition 1. — *The number of shuffle permutations of the type $(k_1,\dots,k_n)$ is equal to*

$$\frac{k_1k_2\cdot\dots\cdot k_n}{(k_1+k_2+\cdots+k_n)(k_2+\cdots+k_n)\cdot\dots\cdot k_n}\binom{k_1+k_2+\cdots+k_n}{k_1,k_2,\dots,k_n}.$$

When implementing shuffle permutations, one can use the following simple idea: In a shuffle permutation, the number whose image is $k_1+\cdots+k_n$ should clearly be the maximal one in its block. Moreover, if this block is of size 1, it should be the last one to comply with the ordering condition on the first elements of blocks. This implies an obvious recursive algorithm to generate a list of shuffle permutations with given sizes of blocks: put the maximal image in the end of each allowed block, and for each such choice list all shuffle permutations where the corresponding block contains one element less than prescribed.

Definition 5. — 1. Let $\mathscr{O}$ be a shuffle operad, $\beta\in\mathscr{O}(n)$, $\alpha_1\in\mathscr{O}(k_1)$, ..., $\alpha_n\in\mathscr{O}(k_n)$. Assume that $\sigma\in S_{k_1+\dots+k_n}$ is a shuffle permutation of the type $(k_1,\dots,k_n)$. Denote by B_s, $s=1,\dots,n$, the s^{th} block of $[k_1+\cdots+k_n]$ (on which σ is monotonous). Then we define

$$\beta(\alpha_1,\dots,\alpha_n)_\sigma=\circ(\beta\otimes\sigma(\alpha_1)\otimes\cdots\otimes\sigma(\alpha_n))\in\mathscr{O}(k_1+\cdots+k_n),$$

where $\sigma(\alpha_s)$ is the image of α_s under the isomorphism between $\mathscr{O}(k_s)$ and $\mathscr{O}(\sigma(B_s))$, and $\circ\colon\mathscr{O}\circ_{sh}\mathscr{O}\to\mathscr{O}$ is the monoid product map.