Operadic Gröbner Bases: an implementation

Mikael Vejdemo-Johansson and Vladimir Dotsenko

14 September 2010

Outline

Definition

A *symmetric (resp. shuffle) operad* is a collection $O(n)$ of vector spaces, one for each n, together with linear maps

$$\circ_\sigma : O(n) \times O(m_1) \times \cdots \times O(m_n) \to O(m_1 + \cdots + m_n)$$

called composition maps. The composition maps are parametrized by arbitrary permutations in $S_{m_1+\cdots+m_n}$ (resp. by shuffle permutations of type $(m_1, \ldots, m_n)$) that provide symmetry actions to the operad. These maps are required to fulfill associativity conditions and allow for a unit for the composition.

Mental model

Key to internalizing any kind of algebraic construction is the model for free objects.

Free commutative rings Polynomial rings

Free monoids Strings in an alphabet

Free categories Paths in a graph

Free operads Rooted trees; with internal nodes labelled by a generating set, and leaves ordered according to any permutation acting.

Why the shuffle operads?

The objects of most interest are the symmetric operads – why shuffle operads?

For symmetric operads, several good properties for Gröbner bases cannot be fulfilled: dimension counting using initial ideals fails badly with full symmetries.

Theorem [Dotsenko–Khoroshkin]

The forgetful functor from symmetric operads to shuffle operads is monoidal and injective on objects.

This suffices for us to be able to compute with shuffle operads instead of symmetric operads.

Enabling Gröbner bases

Picking out a shuffle operad corresponding to an interesting symmetric operad allows us to work computationally without losing information to excessive symmetries.

Theorem [Dotsenko–Khoroshkin]

The Diamond Lemma holds for shuffle operads.

Outline

What I did last summer

Starting at the CIRM workshop OPERADS 09, I worked with Dotsenko on a reference implementation for the approach by Dotsenko–Khoroshkin.

The `Math.Operad` Haskell module is provided by the package `Operads` available at `http://hackage.haskell.org/package/Operads`.

The `Operads` package works well with the Haskell Platform `http://hackage.haskell.org/platform/` version 2010.2.0.0. It is provided under a BSD license.

Getting the software running

1. Download and install the Haskell platform
   `http://hackage.haskell.org/platform/`
2. Download and unpack the `Operads` package
   `http://hackage.haskell.org/package/Operads`
3. From the directory of `Operads`, run:
   ```
   ghc --make Setup
   ./Setup configure && ./Setup build &&
            ./Setup install
   ```
4. `Operads` is now available in your Haskell platform installation. Example code for a number of examples resides in the `examples/` directory in the current directory.

Example session

```
% ghci -cpp Math.Operad
*Math.Operad> let v = corolla 2 [1,2]
*Math.Operad> let [g1t1,g1t2,g2t2] =
    [shuffleCompose 1 [1,2,3] v v,
     shuffleCompose 2 [1,2,3] v v,
     shuffleCompose 1 [1,3,2] v v]
*Math.Operad> let ac =
    [(oet g1t1) + (oet g1t2), (oet g2t2) - (oet g1t2)]
    :: [OperadElement Integer Rational PathPerm]
*Math.Operad> let acGB = operadicBuchberger ac
*Math.Operad> length acGB
3
```

Why Haskell?

- Easy treeshaped datatypes

```
data Tree nodeType leafType =
    Leaf leafType
  | Node nodeType [Tree nodeType leafType]
```

- Easy and fast recursion; all the benefits of a functional programming language.
- Algebraic approaches to programming; most mathematics can be translated word-by-word into workable code.

Data type choices

Monomials Trees:

```
data Tree nodeType leafType =
    Leaf leafType
  | Node nodeType [Tree nodeType leafType]
```

Elements Polynomials of trees:

```
type OperadElement n l r =
    Map (Tree n l) r
```

Associative array (stored as balanced binary tree) of coefficients keyed by monomials.

Algebra and analysis with datatypes

In classical Buchberger, reduction and S-polynomials are given by:

$$S_{f,g} = \frac{\operatorname{lcm}(\operatorname{lm} f, \operatorname{lm} g)}{\operatorname{lm} f} f - \frac{\operatorname{lcm}(\operatorname{lm} f, \operatorname{lm} g)}{\operatorname{lm} g} g$$

Algebra and analysis with datatypes

In non-commutative Buchberger, reduction and S-polynomials are given by, for each overlap $O_{\mathrm{lm}\, f,\mathrm{lm}\, g}$:

$$S_{f,g} = (O_{\mathrm{lm}\, f,\mathrm{lm}\, g}/\,\mathrm{lm}\, f)f - g(\mathrm{lm}\, g \backslash O_{\mathrm{lm}\, f,\mathrm{lm}\, g})$$

Algebra and analysis with datatypes

In operadic Buchberger:

- One S-polynomial for each overlap of two initial tree monomials.
- Embedding of each tree into the common multiple most relevant.
- S-polynomials and reductions rely on being able to remove factor from common multiple; and apply the outside of the factor tree to the other trees in a polynomial.

This forms a map $m_{S,T}$ that takes polynomials to multiplied up polynomials, and thus, for a specific common multiple C of $\operatorname{lm} f, \operatorname{lm} g$,

$$S_{f,g} = m_{\operatorname{lm} f, C}(f) - m_{\operatorname{lm} g, C}(g)$$

Derivatives of datatypes

Data type construction needed well known in Combinatorial Species; as well as in the Functional Languages community:

A data type is a functor. Formal derivatives of these functors give the data type with a single element removed.

Lists are given by the functor equation $L(x) = 1 + x \times L(x)$. Implicit derivative is $L'(x) = x \times L'(x) + L(x) = L(x) + L(x)$.

Operad trees are given by the functor equation $OT_\ell(x) = 1 + x \times L(OT_\ell(x))$. Implicit derivative is $OT'_\ell(x) = L(OT_\ell(x)) + x \times L'(OT_\ell(x)) \times OT'_\ell(x)$.

This final equation can be used as is for a data type declaration in Haskell.

Outline

Comparisons

Most monomial orderings proposed by Dotsenko–Khoroshkin rely on a tree traversal to collect data needed for ordering the trees.

Balanced binary trees perform a lot of comparisons between keys.

As a result, our first implementation spent almost all computational time traversing monomial trees.

Comparisons – Bag datastorage

As a first approach, we tried using a Bag datatype

```
data Bag key value =
  Bag (key, value) [(key, value)]
```

which performs addition by appending key/value pairs except for if the leading terms match. This triggers a consolidation where all the added pairs are added up, thus amortizing additions over time.

Comparisons – Bag datastorage

Inspiration from homology computations, where it has been used to noticable benefit.

However: Gröbner basis computations do very many arithmetic operations with matching leading terms. Thus, the Bag datatype never managed to delay additions.

Comparisons – Caching wrapper around Map

The solution we settled on has a wrapper around `Map` that provides caching of the tree traversal data for the monomial trees.

This certainly speeds the code up significantly; and demotes the comparisons in the storage from their leading position among the timesinks in profiling.