

Re: what's it worth to write a short program for polynomial multiplication?

# Re: what's it worth to write a short program for polynomial multiplication?

---

*Source:* <http://sci.tech-archive.net/Archive/sci.math.symbolic/2008-06/msg00038.html>

---

- *From:* rjf <fateman@xxxxxxxx>
  - *Date:* Sat, 7 Jun 2008 07:09:50 -0700 (PDT)
- 

On Jun 7, 5:05 am, Christopher Creutzig <christop...@xxxxxxxx> wrote:

rjf wrote:

Why should not the object-oriented choice of operator use the types of all the operands?

Good question. Most OO programming languages work that way. CLOS, the Common Lisp Object System is the one exception I am aware of, but perhaps there are more.

In C++, you can overload functions (including operators) on all arguments, sure:

```
template <typename coeffT>
SparsePoly<coeffT> operator*(SparsePoly<coeffT> &a const,
SparsePoly<coeffT> &b const);
```

```
template <typename coeffT>
SparsePoly<coeffT> operator*(DensePoly<coeffT> &a const,
SparsePoly<coeffT> &b const);
```

```
template <typename coeffT>
SparsePoly<coeffT> operator*(SparsePoly<coeffT> &a const,
DensePoly<coeffT> &b const);
```

```
template <typename coeffT>
DensePoly<coeffT> operator*(DensePoly<coeffT> &a const,
DensePoly<coeffT> &b const);
```

This is also possible for types only determined at runtime, but requires more code. The idiom is called a double dispatch pattern. Oh, and I should point out that the choice of return types above is of course not a good idea in general. Using the double dispatch pattern and

Re: what's it worth to write a short program for polynomial multiplication?

determining whether to return a sparse or dense polynomial based on the actual data found is likely to be much better. But I haven't done any tests on this yet.

There are quite a few papers on this topic of how to do multiple dispatch in different languages, added on in python, C++, etc. Usually (always?) this falls short of the generality of common lisp, and may also fall short of the efficiency of common lisp, which can at run time not only call methods whose details are determined by run-time tests on the operands, but can also at run time define new methods. If the types of arguments can be determined at compile time, that is an optimization that is available.

see for example

[http://en.wikipedia.org/wiki/Multiple\\_dispatch](http://en.wikipedia.org/wiki/Multiple_dispatch)

... to quote:

"In "conventional", i.e. single dispatch, object-oriented programming languages, when you invoke a method ("send a message" in Smalltalk, "call a member function" in C++) one of its arguments is treated specially and used to determine which of the (potentially many) methods of that name is to be applied. In many languages, the "special" argument is indicated syntactically; for example, a number of programming languages put the special argument before a dot in making a method call: special.meth(other,args,here).

In languages with multiple dispatch, all the arguments are treated symmetrically in the selection of which method to call. Matching recognizes first, second, third, etc. position within the call syntax, but no one argument "owns" the function/method carried out in a particular call.

The Common Lisp Object System is an early and well-known example of multiple dispatch.

...end quote.

This is somewhat off topic; I do not myself consider unadorned common lisp to be an ideal language for expression of mathematics for applications. It is pretty handy for programming, though, and I think that is illustrated by the short programs for multiplication of polynomials in the referenced paper. If shorter programs can be written in other languages, (other than Tim's x\*y) I'd like to see them.

Re: what's it worth to write a short program for polynomial multiplication?

Re: what's it worth to write a short program for polynomial multiplication?

RJF