Component-free vector algebra in Aldor

Songxin Liang, David J. Jeffrey and Stephen M. Watt
Ontario Research Centre for Computer Algebra, University of Western Ontario,
London, Ontario, Canada

Abstract

An implementation of a component-free symbolic vector algebra in Aldor is presented. This package provides two powerful functions: simplification of vector expressions and the proof of vector identities. The implementation benefits greatly from Aldor's strong typing, which allows several simplification problems that have defeated previous implementations to be solved.

1 Introduction

Vector algebra and vector calculus have many applications throughout science and mathematics. Vector analysis often simplifies the derivation of mathematical theorems and the statements of physical laws, while vector notation can often clearly convey geometric or physical interpretations that greatly facilitate understanding. Physicists and engineers prefer to formulate their equations of motion using abstract vectors rather than components. Thus they prefer to write the velocity of a rotating body as $\omega \wedge \mathbf{r}$, rather than in some component form as

$$[\omega_2 z - \omega_3 y, \omega_3 x - \omega_1 z, \omega_1 y - \omega_2 x] \ .$$

Almost all well-known computer algebra systems provide vector operations, and some have add-on vector analysis packages, for example, the VectorAnalysis package for Mathematica [4]. However, all these packages only perform component-dependent operations, not component-independent ones. This means that before one can do any vector operations, one must set the components for all vectors involved. This is inconvenient when one wants to deal with problems containing many vectors and one only wants to know the relationships among them. Compared with component-dependent systems, which have a systematic method for deriving mathematical statements, component-independent systems are more difficult and challenging, because vector algebra has a strange and intriguing structure [3]. Denoting the vector and scalar products of a, b by $a \wedge b$ and $a \cdot b$, we see:

- neither operation is associative. If p, q, r are vectors, then $p \wedge (q \wedge r) \neq (p \wedge q) \wedge r$, whereas $p \cdot (q \cdot r)$ and $(p \cdot q) \cdot r$ are invalid.
- neither operation has a multiplicative unit. There does not exist a fixed vector u such that for any vector p, $u \wedge p = p$ or $p \wedge u = p$ or $u \cdot p = p$ or $p \cdot u = p$.

- both admit zero divisors. For any vector p, $p \wedge p = 0$; if q is a vector perpendicular to p, then $p \cdot q = 0$.
- the two operations are connected by the strange side relation $p \wedge (q \wedge r) = (p \cdot r)q - (p \cdot q)r$

Packages by Fiedler [2] and Stoutemyer [3] implement component-free vector operations. However, the emphasis of these packages is still on component-based operations, and only the package by Stoutemyer provides non-trivial simplification examples. Stoutemyer's package left simplification problems unsolved. When he tried to simplify the vector expression

$$(a \wedge b) \wedge (b \wedge c) \cdot (c \wedge a) - (a \cdot (b \wedge c))^2$$

which should simplify to zero, he only got

$$a \cdot (a \cdot b \wedge c.b) \wedge c - (a \cdot b \wedge c)^2 \ .$$

The reason was that although the scalar factor $a \cdot b \wedge c$ could be factored out, revealing the expression to be zero, the built-in scalar-factoring-out mechanism could not recognize that $a \cdot b \wedge c$ is a scalar despite its vector components.

Because of space limitations, this description of the program is necessarily brief, and omits most details. However, the source code for the program will be made available on the Aldor web site, from where it may be downloaded and inspected.

2 Mathematical strategy

In this section we describe the mathematical strategy of the package, while the next section describes the Aldor implementation. Simplification is achieved by defining a canonical form for a vector expression and transforming all input expressions into this form.

2.1 Expression representation

A vector expression is a sum of terms, with each term being a product of scalar and vector. The scalar is further divided into scalars from the coefficient field and scalars formed because of a scalar product of vectors. A vector expression is represented as: `Rep==List Term,` and

```
Term==Record (coe:R, sca:  List List String, vec:List String),
```

where `coe, sca, vec` are respectively the coefficient, scalar part and vector part of a term. For example, the term $-2(a.b)(a \wedge b \cdot c)(c \wedge d)$ is represented as

```
[-2, [["a","b"], ["a","b","c"]], ["c","d"]].
```

Note that the vector triple product $(a \wedge b) \cdot c$ is an important scalar that is represented by the three-element bracket. Because of the equality $(a \wedge b) \cdot c = a \cdot (b \wedge c)$, the product can be represented as the triple `["a","b","c"]`.

In addition to the basic data structure, there is a set of ordering rules to two occurrences of the same term will be represented by the same list.

2.2 Transformation rules

The transformation rules used to reduce expressions to canonical form are taken from standard textbooks. In the current version of the program, the basic rules are not all independent of each other. For example, the program applies separately the rule

$$a \wedge a = 0 \ ,$$

even though it can be deduced from the rule

$$a \wedge b = -b \wedge a \ ,$$

because setting $b = a$ gives $a \wedge a = -a \wedge a$. Note that in physics books, the rule is proved by using the fact that the angle between parallel vectors is 0.

3 Implementation in Aldor

Aldor is a strongly typed, imperative programming language with a two-level object model of categories and domains [1]. Here we give an overview of a few of the top level constructs. We first define a vector space category as follows:

```
define VectorSpcCategory(R:Join(ArithmeticType, ExpressionType), n:MI==3):
Category==with
{
  *: (R,%)->%;
  *: (%,R)->%;
  +: (%,%)->%;
  -: (%,%)->%;
  -:     %->%;
  =: (%,%)->Boolean;
  default
  {
    import from R;
    (x:%)-(y:%):%==x+(-1)*y;
    (x:%)*(r:R):%==r*x;
    -(x:%):%==(-1)*x;
  }
}.
```

Here, n is the dimension of the space. Based on VectorSpcCategory, we define the vector algebra category VectorAlgCategory as follows.

```
define VectorAlgCategory(R:Join(ArithmeticType, ExpressionType)):
Category== VectorSpcCategory(R) with
{
  vector:      Symbol->%;
  scalarZero:  ()->%;
  vectorZero:  ()->%;
  realVector?: %->Boolean;
  simplify:    (%,UseAdvancedRules:Boolean==false)->%;
  identity?:   (%,%)->Boolean;
  s3p:   (%,%,%)->%;
  *:     (%,%)->%;
  apply: (%,%)->%;
  ^:     (%,%)->%;
  <<:    (TextWriter,%)->TextWriter;
  default
  {
    s3p(x:%,y:%,z:%):%==apply(x^y,z);
  }
}.
```

With these categories, we can implement a vector algebra domain `VectorAlg`.

4 An Example

Stoutemyer's unsolved problem from section 1 is solved as follows.

$$\begin{aligned}
((a \wedge b) \wedge (b \wedge c)) \cdot (c \wedge a) - (a \cdot (b \wedge c)) * (a \cdot (b \wedge c)) &\Rightarrow \\
((a \wedge b \cdot c) * b - (a \wedge b \cdot b) * c) \cdot (c \wedge a) - (a \cdot (b \wedge c)) * (a \cdot (b \wedge c)) &\Rightarrow \\
((a \wedge b \cdot c) * b - 0) \cdot (c \wedge a) - (a \cdot (b \wedge c)) * (a \cdot (b \wedge c)) &\Rightarrow \\
(a \wedge b \cdot c) * (b \cdot (c \wedge a)) - (a \cdot (b \wedge c)) * (a \cdot (b \wedge c)) &\Rightarrow \\
(a \wedge b \cdot c) * (b \wedge c) \cdot a - (a \cdot (b \wedge c)) * (a \cdot (b \wedge c)) &\Rightarrow \\
(a \wedge b \cdot c) * (a \cdot (b \wedge c)) - (a \cdot (b \wedge c)) * (a \cdot (b \wedge c)) &\Rightarrow \\
(a \cdot (b \wedge c) * (a \cdot (b \wedge c)) - (a \cdot (b \wedge c)) * (a \cdot (b \wedge c)) &\Rightarrow \quad 0
\end{aligned}$$

References

[1] Aldor Compiler User Guide, `http://www.aldor.org`

[2] Fiedler, B., 1997. Vectan 1.1. Manual Math. Inst., Univ. Leipzig, 1-22.

[3] Stoutemyer, D. R., 1979. Symbolic computer vector analysis. Computers & Mathematics with Applications, v 5, n 1, 1979, p 1-9.

[4] Wolfram, S., 1996. The Mathematica Book, 3rd ed.. Wolfram Media/Cambridge University Press.