RegressiveProduct (∨)

`RegressiveProduct[`x_1`, `x_2`, ...]`
denotes the regressive (or Vee) product ot the expressions x_1, x_2,....

Details

- The `RegressiveProduct` is `Listable`.
- `RegressiveProduct[x1, x2, ...]` and `Vee[x1, x2, ...]` are equivalent input forms for constructing the expression `x1 ∨ x2 ∨ ...`. They do not simplify or change the expression in any way.
- In *GrassmannAlgebra*, the `Vee` operator has been endowed with the extra attribute `Flat`. This ensures that the regressive product acts associatively. For example `Vee[x, Vee[y, z]]` is automatically reduced to `Vee[x, y, z]`.
- The vee symbol ∨ may be entered by using the *Common Operations* palette or the sequence ESC∨ESC.
- A regressive product □ ∨ □ may be entered by using the *Common Operations* palette. Continued regressive products may be entered by continued clicking on the □ ∨ □ button. For example, three successive clicks produces □ ∨ □ ∨ □ ∨ □. The □ are placeholders in which expressions may be entered successively using the Tab key.
- The regressive product is defined as the algebraic dual of the exterior product as explained in Chapter 3. The exterior product is zero if it overfills the space. The regressive product of two factors is zero if they underfill the space. The term "common factor" can be a bit confusing. The regressive product of two identical factors is zero if they don't fill the space.

Examples (5)

Basic Examples (1)

```
In[1]:= << GrassmannCalculus`
```

Setting some Book preferences:

```
In[2]:= ★A; ★ℬ5;
```

These inputs are equivalent:

```
In[3]:= {RegressiveProduct[x, y], Vee[x, y], x ∨ y}
Out[3]= {x ∨ y, x ∨ y, x ∨ y}
```

Because the regressive product is associative these inputs are also equivalent:

```
In[4]:= {RegressiveProduct[x, y ∨ z], Vee[Vee[x, y], z], Vee[x ∨ Vee[y] ∨ z]}
Out[4]= {x ∨ y ∨ z, x ∨ y ∨ z, x ∨ y ∨ z}
```

The regressive product is listable.

```
In[5]:= a ∨ {x, y, z}
Out[5]= {a ∨ x, a ∨ y, a ∨ z}
```

A principal use of the regressive product is to find intersections of spaces. These two 3-planes in 5-space intersect in a line but it is only know up to a congruence factor.

```
In[6]:= (e1 ∧ e4 ∧ e5) ∨ (e2 ∧ e3 ∧ e4)
        % // ToCommonFactor
Out[6]= e1 ∧ e4 ∧ e5 ∨ e2 ∧ e3 ∧ e4

Out[6]= ★c e4
```

A more intuitive example might be the intersection of a horizontal and vertical line in the GrassmannPlane. The use of `CongruenceSimplify` sets ★c == 1 and is appropriate in Projective or metric spaces.

```
In[7]:= SetActiveAssociation[PublicGrassmannAtlas[["Grassmann Plane"]]]

In[8]:= ((★𝟙 + 3 ey) ∧ ex) ∨ ((★𝟙 + 5 ex) ∧ ey)
        % // ★C
Out[8]= (★𝟙 + 3 ey) ∧ ex ∨ (★𝟙 + 5 ex) ∧ ey

Out[8]= ★𝟙 + 5 ex + 3 ey
```

For more examples of intersections see the Function page for `CongruenceFactor`.

Axioms of the Regressive Product (1)

```
In[1]:= ★A; ★P; ★D
Out[1]= 3
```

∨6: The regressive product of an *m*-element and a *k*-element is an (*m*+*k*–*n*)-element.

To be non-zero the vectors in an exterior product must not overflow the dimension of the space. For a regressive product they must equal or overflow the space.

```
In[2]:= Print["The regressive product of a 2-element and 3-element in 3-space"];
        step1 = (e1 ∧ e2) ∨ (e1 ∧ e2 ∧ e3)
        step2 = Grade@(List @@ step1)
        Print["The computed Grade of the regressive product"]
        Grade[step1]
        Print["The grade according to the axiom ∨6"]
        HoldForm[m + k - n]
        % /. {m → First[step2], k → Last[step2], n → ★D}
        % // ReleaseHold
        Print["To obtain the resulting 2-element we must use ToCommonFactor. ★𝒢 will not be sufficient."]
        step1 // ToCommonFactor
        Grade[%]
```

```
  The regressive product of a 2-element and 3-element in 3-space
```

```
Out[2]= (e1 ∧ e2) ∨ (e1 ∧ e2 ∧ e3)
```

```
Out[2]= {2, 3}
```

```
  The computed Grade of the regressive product
```

```
Out[2]= 2
```

```
  The grade according to the axiom ∨6
```

```
Out[2]= m + k - n
```

```
Out[2]= 2 + 3 - 3
```

```
Out[2]= 2
```

```
  To obtain the resulting 2-element we
    must use ToCommonFactor. ★𝒢 will not be sufficient.
```

```
Out[2]= ★c e1 ∧ e2
```

```
Out[2]= 2
```

∨7: The regressive product is associative.

This is implemented by the Flat Attribute of `Vee`.

```
In[3]:= Attributes[Vee]
Out[3]= {Flat, Listable, OneIdentity}

In[4]:= (e1 ∨ e2) ∨ e3 == e1 ∨ (e2 ∨ e3)
Out[4]= True

In[5]:= HoldForm[(e1 ∨ e2) ∨ e3]
        % // FullForm
        % // ReleaseHold // FullForm
Out[5]= (e1 ∨ e2) ∨ e3

Out[5]//FullForm= HoldForm[Vee[Vee[Subscript[e, 1], Subscript[e, 2]], Subscript[e, 3]]]

Out[5]//FullForm= Vee[Subscript[e, 1], Subscript[e, 2], Subscript[e, 3]]
```

∨8: There is a unit *n*-element which acts as an identity under the regressive product.

This is implemented by `GrassmannRule[3, 10]` and `GrassmannRule[3, 11]`. Note that the proper notation for `n` is ★`n`.

```
In[6]:= step1 = {1 ∨n e1 ∨ e2, e1 ∨ e2 ∨n 1}
         % /. GrassmannRule[3, 10]
         % /. GrassmannRule[3, 11]
Out[6]= {1 ∨n e1 ∨ e2, e1 ∨ e2 ∨n 1}

Out[6]= {e1 ∨ e2, e1 ∨ e2 ∨n 1}

Out[6]= {e1 ∨ e2, e1 ∨ e2}
```

If a numerical value is used on the graded 1, then `GrassmannRule[3, 12]` and `GrassmannRule[3, 13]` can be used.

```
In[7]:= {1 ∨3 e1 ∧ e2, e1 ∨ e2 ∨3 1}
         % /. GrassmannRule[3, 12]
         % /. GrassmannRule[3, 13]
Out[7]= {1 ∨3 (e1 ∧ e2), e1 ∨ e2 ∨3 1}

Out[7]= {e1 ∧ e2, e1 ∨ e2 ∨3 1}

Out[7]= {e1 ∧ e2, e1 ∨ e2}
```

∨9: Non-zero *n*-elements have inverses with respect to the regressive product. This is discussed in Section 3.3 in the Book. It is illustrated in The Unit n-element example section below.

∨10: The regressive product of elements of odd complementary grade is anti-commutative.

```
In[8]:= ★A; ★B5; ★D
Out[8]= 5
```

```
In[9]:= Print["Two elements:"]
        step1 = {element1, element2} = {e1 ∧ e2, e2 ∧ e3 ∧ e4 ∧ e5}
        Print["With grades:"]
        Grade[step1]
        Print["And odd complementary grades: ⋆D - grade:"]
        ⋆D - Grade[#] & /@ step1
        Print["Which can be checked with the command OddComplementaryGradeQ"]
        OddComplementaryGradeQ[step1]
        Print["So their regressive product is anti-commutative."]
        element1 ∨ element2 == -(element2 ∨ element1)
        ⋆𝒢@%
```

```
Two elements:
```

```
Out[9]= {e1 ∧ e2, e2 ∧ e3 ∧ e4 ∧ e5}
```

```
With grades:
```

```
Out[9]= {2, 4}
```

```
And odd complementary grades: ⋆D - grade:
```

```
Out[9]= {3, 1}
```

```
Which can be checked with the command OddComplementaryGradeQ
```

```
Out[9]= {True, True}
```

```
So their regressive product is anti-commutative.
```

```
Out[9]= (e1 ∧ e2) ∨ (e2 ∧ e3 ∧ e4 ∧ e5) == -((e2 ∧ e3 ∧ e4 ∧ e5) ∨ (e1 ∧ e2))
```

```
Out[9]= True
```

∨10 can also be implemented with `GrassmannRule[3, 9]` and by setting ⋆n to the dimension of the vector space.

```
In[10]:= element1 ∨ element2
         % /. GrassmannRule[3, 9]
         % /. ⋆n → ⋆D
```

```
Out[10]= (e1 ∧ e2) ∨ (e2 ∧ e3 ∧ e4 ∧ e5)
```

```
Out[10]= (-1)^((-4+⋆n) (-2+⋆n)) (e2 ∧ e3 ∧ e4 ∧ e5) ∨ (e1 ∧ e2)
```

```
Out[10]= -((e2 ∧ e3 ∧ e4 ∧ e5) ∨ (e1 ∧ e2))
```

∨11: Additive identities act as multiplicative zero elements under the regressive product. GrassmannAlgebra does not maintain a formal graded zero. Since the result is 0 regardless of the grade, one can either use an ordinary 0 or the special symbol ⋆0, which stands for `GradeOf Zero`.

```
In[11]:= {0 ∨ e1 ∧ e2, ⋆0 ∨ e1 ∧ e2}
         ⋆𝒢[%]
```

```
Out[11]= {0 ∨ (e1 ∧ e2), ⋆0 ∨ (e1 ∧ e2)}
```

```
Out[11]= {0, 0}
```

```
In[12]:= ⋆A; ⋆P;
```

∨12: The regressive product is both left and right distributive under addition. This axiom is built into the definitions of ExpandRegressiveProducts, which in turn are automatically called by ★𝒢.

```
In[13]:= step1 = (e1 + e2) ∨ (e1 ∧ e2 ∧ e3)
         % // ExpandRegressiveProducts
         step1 // ★𝒢
Out[13]= (e1 + e2) ∨ (e1 ∧ e2 ∧ e3)

Out[13]= e1 ∨ (e1 ∧ e2 ∧ e3) + e2 ∨ (e1 ∧ e2 ∧ e3)

Out[13]= e1 ∨ (e1 ∧ e2 ∧ e3) + e2 ∨ (e1 ∧ e2 ∧ e3)
```

∨13: Scalar multiplication commutes with the regressive product. This axiom is built into the definitions of SimplifyRegressiveProducts, which in turn are automatically called by ★𝒢.

```
In[14]:= {e1 ∨ (a (e1 ∧ e2 ∧ e3)), (a e1) ∨ (e1 ∧ e2 ∧ e3), a (e1 ∨ (e1 ∧ e2 ∧ e3))}
         % // SimplifyRegressiveProducts
         Equal @@ %
Out[14]= {e1 ∨ (a e1 ∧ e2 ∧ e3), (a e1) ∨ (e1 ∧ e2 ∧ e3), a e1 ∨ (e1 ∧ e2 ∧ e3)}

Out[14]= {a e1 ∨ (e1 ∧ e2 ∧ e3), a e1 ∨ (e1 ∧ e2 ∧ e3), a e1 ∨ (e1 ∧ e2 ∧ e3)}

Out[14]= True
```

The Unit n-element (1)

We set the default 3-dimensional vector space and show precedence.

```
In[1]:= ★A; ★P
```

All n-elements, where n is the dimension of the underlying vector space, are congruent. However, there are various symbolic entities used to express this congruence. If we have a metric the congruence may be expressed exactly. This can all lead to a bit of confusion but nevertheless n-elements are a good vehicle for understanding the relations between various congruent elements. Congruent elements represent the same space.

Let's look first at the various types of n-elements that might appear in Grassmann algebra. Two rather standard n-elements, to which other n-elements are often referred are the *unit n-element* and the *basis n-element*.

The unit n-element is represented symbolically as $\underset{\star n}{1}$ and is the unit identity in regressive products. It is defined in axiom ∨8, Equation 3.4.

The symbolic form will simplify in regressive products but not necessarily in other calculations, such as ToCommonFactor. An alternative to the use of ★n is the use of ★D, which will immediately simplify to the dimension of the current vector space.

```
In[2]:= {1 ∨ x, 1 ∨ x}
         % // ★𝒢
Out[2]= {1 ∨ x, 1 ∨ x}

Out[2]= {x, x}
```

On the other hand, the basis n-element is the cobasis of 1, Equation 2.25 in Section 2.6. It is also available on the Cobasis palette. It is simply the product of the vector basis elements.

```
In[3]:= 1
         % /. UnderBar → CobasisElement
Out[3]= 1

Out[3]= e1 ∧ e2 ∧ e3
```

Without a metric these two standard n-elements are related only up to a congruence. Any measure of the size of the basis vectors is unknown to the algebra. So they are related by Equation 3.10 in Section 3.3. The congruence factor is represented symbolically by ★c.

```
In[4]:= e1 ∧ e2 ∧ e3 == ★c 1
                          ★D
Out[4]= e1 ∧ e2 ∧ e3 == ★c 1
                           3
```

That is also the answer we obtain if we calculate the intersection of these two spaces using ToCommonFactor:

```
In[5]:= (e1 ∧ e2 ∧ e3) ∨ 1
                         3
        % // ToCommonFactor
Out[5]= (e1 ∧ e2 ∧ e3) ∨ 1
                         3

Out[5]= ★c 1
           3
```

The third kind of n-element is one composed of general elements. For example we could use the symbolic vectors {*x*, *y*, *z*} and calculate the intersection.

```
In[6]:= (x ∧ y ∧ z) ∨ 1
                      ★D
        % // ToCommonFactor
Out[6]= (x ∧ y ∧ z) ∨ 1
                      3

Out[6]= ★c ⟨x ∧ y ∧ z⟩ 1
              3
```

The AngleBracket expression, ⟨*x*∧*y*∧*z*⟩, represents the coefficient of *x*∧*y*∧*z* when expressed in terms of the **basis n-vector**. *x*∧*y*∧*z*== ⟨x∧y∧z⟩ *e*1∧*e*2∧*e*3. This is again a kind of congruence factor so it may seem like too much to have two of them. In that case you can use CongruenceSimplify, with the alias ★*C*, instead of ToCommonFactor.

```
In[7]:= x ∧ y ∧ z ∨ 1
                    ★D
        % // ★C
Out[7]= (x ∧ y ∧ z) ∨ 1
                      3

Out[7]= ⟨x ∧ y ∧ z⟩ 1
                    3
```

That eliminated the ★c. Notice that a regressive product is not commutative with respect to the ToCommonFactor and CongruenceSimplify operations. We obtain the cofficient of the first factor times the n-element of the second element. The coefficient is always with repsect to the basis n-element.

```
In[8]:= 1 ∨ (x ∧ y ∧ z)
        3
        % // ★C
Out[8]= 1 ∨ (x ∧ y ∧ z)
        3

Out[8]= ⟨1⟩ x ∧ y ∧ z
        ⟨3⟩
```

AngleBrackets and CongruenceSimplify were introduced into GrassmannAlgebra for convenience in working with projective spaces, Section 4.12, p 226 in the book.

In summary ★c is the congruence with respect to the unit n-element (for regressive products) and AngleBracket is the congruence factor with respect to the n-basis (as an exterior product).

The following table summarizes the various basis and unit elements. Different notations are used for the unit n-element. The book often uses $\underset{n}{1}$ but this is not recognized in calculations. The $\underset{\star n}{1}$ notation is recognized in the Dual command and some rules, but this is not generally recognized elsewhere, especially in ToCommonFactor where $\underset{\star D}{1}$ is necessary, where ★D is the Dimension of the underlying linear space.

Out[9]=

Unit scalar	Value == 1 $1 \wedge \underset{m}{\alpha} == \underset{m}{\alpha}$, Axiom ∧8
Basis n−element	Value == $\underline{1}$, Eq 2.25 $\underline{1}$ == CobasisElement[1] == Wedge @@ Basis
Unit n−element	Value == $\underset{\star n}{1}$ $\underline{1} == \star c \underset{\star n}{1}$, Eq 3.10 $\underset{\star n}{1} == \underset{\star D}{1}$ Value == $\overline{1} == \underset{\star n}{1}$, With metric and axiom $\overline{5}$

Fill and Fold (1)

Experiments (1)

See Also

ToCommonFactor ▪ **RegressiveProductQ** ▪ **ExpandRegressiveProducts** ▪ **SimplifyRegressiveProducts** ▪ **ExpandAndSimplifyRegressiveProducts** ▪ **ExteriorToRegressive** ▪ **RegressiveToExterior** ▪ **InteriorToRegressive** ▪ **RegressiveToInterior** ▪ **ToRegressiveProducts** ▪ **RegressiveUnitRules** ▪ **ComposeRegressiveGradedForm** ▪ **GradeDistributeRegressiveProducts**

ExteriorProduct ▪ **InteriorProduct** ▪ **CliffordProduct** ▪ **GeneralizedProduct** ▪ **HypercomplexProduct**

Related Guides

- Expressions
- Grassmann Calculus
- RegressiveProducts