The PL package

Version 2.3

July 2014

Igor G. Korepanov
Alex I. Korepanov
Nurlan M. Sadikov

Nurlan M. Sadikov Email: ancleCharly@gmail.com

Copyright

Содержание

Chapter 1

Шаровые комплексы.

Piecewise-linear, or simply PL, ball complexes are, at least at this moment, the central objects with which our package *PL* deals. First, a ball complex is, simply speaking, a kind of cell complex but such where all closed cells (= balls) are embedded. In particular, their boundaries are genuine spheres, not crumpled/folded. The formal definition of *PL* ball complex reads: A PL ball complex is a pair (X,U), where X is a compact Euclidean polyhedron and U is a covering of X by closed PL-balls such that the following axioms are satisfied:

- the relative interiors of balls from U form a partition of X,
- the boundary of each ball from U is a union of balls from U.

We also call PL ball complexes "polytopes", for brevity, hence prefix "Pol" in the names of some of our functions.

1.1 Representation of a PL ball complex

A *PL*-ball complex is defined up to *PL*-homeomorphism only by the combinatorics of adjunctions of its balls. Due to this, we represent them combinatorially in the following way. First, we assume that all vertices in the complex are numbered from 1 to their total number N_0. Hence, in this sense, the 0-skeleton of the complex is described. Next, assuming that the k-skeleton is already given, which implies (in particular) the numeration of all k-cells, we describe the $(k+1)$-skeleton as the list of all $(k+1)$-cells, each of which, in its turn, is the set of numbers of k-cells in its boundary. Then we compose the list of length n, where n is the dimension of the complex, whose elements are lists of 1-,..., n-cells. Thus, a three-dimensional ball B^3 can be represented by the following *PL* ball complex with two vertices 1 and 2:

Пример

```
[
  [ [1,2], [1,2] ], # two one-dimensional simplexes, each with
                    # ends 1 and 2, of which the first is referred to
                    # in the next line as 1, the second - as 2;
  [ [1,2], [1,2] ], # two disks - bigons - bounded each by
                    # one-dimensional simplexes 1 and 2;
  [ [1,2] ]         # the three-ball bounded by bigons 1 and 2
]
```

ТУТ БЫ НАДО ВСТАВИТЬ КАРТИНКУ Actually, we add a list of vertices with their names or something like that in the beginning of the above ball complex representation. For instance, our function `ballAB`(n) calls them `"A"` and `"B"`. So, our GAP representation of the ball in Figure~\ref{fig:B3} is the following record:

Пример

```
gap> ballAB(3);
rec( vertices := [ "A", "B" ],
  faces := [ [ [ 1, 2 ], [ 1, 2 ] ], [ [ 1, 2 ], [ 1, 2 ] ], [ [ 1, 2 ] ] ]
 )
```

1.1.1 IsPolytope

▷ `IsPolytope(`*pol*`)` (функция)

Функция проверяет формальные признаки структуры *pol* в соответствии с правилами задания политопов. Если данные построены корректным образом, то функция выдаст *true* и *false* в противном случае.

Пример

```
gap> IsPolytope(T2);
true
```

Так как нет единого алгоритма распознавания $n-$мерной сферы, проверка того, что каждая клетка является шаром опушена. Это может породить ошибку когда данные удовлетворяют всем формальным признакам, но политоп не является шаровым. В данном случае программа, все равно выдаст *true* вместо положенного *false*.

1.2 Указывающие функции

В данном разделе представлены функции которые могут либо распознать многообразие по политопу, либо вычислить инвариант этого многообразия.

1.2.1 EulerNumber

▷ `EulerNumber(`*pol*`)` (функция)

функция вычисляет число Эйлера для шарового комплекса. Данная функция полиморфна и способна принимать для вычислений данные типа политоп (IsPolytope), именнованные списки по размерностям в котором содержится структура политопа и именованный список по размерностям элементами которого могут выступать количества клеток определенной размерности.

Пример

```
gap> EulerNumber(T2);
0
gap> EulerNumber(sphereAB(4));
2
gap> EulerNumber(sphereAB(3));
0
```

1.2.2 FundGroup

▷ `FundGroup(pol)` (функция)

Computes the fundamental group of the given polytope.

1.3 Общая информация о шаровом комплексе

1.3.1 LengthPol

▷ `LengthPol(pol)` (функция)

В *resul.d* указывается мощьность d-мерного остова в политопе *pol*.

Пример

```
gap> LengthPol(T2);
total16
rec( 0 := 4, 1 := 8, 2 := 4 )
```

1.3.2 PolBoundary

▷ `PolBoundary(pol)` (функция)

вычисляет границу политопа. На выход подает список в котором указаны только клетки размерности $(n-1)$ составляющие границу.

Пример

```
gap> PolBoundary(T2);
[ ]
gap> s1:=sphereAB(1);;
gap> d2:=ballAB(2);;
gap> ft2:=PolProduct(d2,s1);;
gap> PolBoundary(ft2);
[ 1, 2, 3, 4 ]
```

В данном примере $T2$ является тором, $s1$ - одномерная сфера, $d2$ - двумерный диск и $ft2$ - полноторие созданное как декартово произведение 2-диска и 1-сферы.

1.3.3 PolInnerFaces

▷ `PolInnerFaces(pol)` (функция)

Build index of inner faces of given polytope complex *returned*[*i*] - set of inner faces of dimensions (i-1). Any face is outer if it has at most 1 adjacent face of higher dimension of if it lies in the boundary of such a face. And inner faces are not outer faces.

Пример

```
gap> PolInnerFaces(T2);
[ [ 1 .. 4 ], [ 1 .. 8 ] ]
gap> PolInnerFaces(ft2);
[ [  ], [  ], [ 5, 6 ] ]
```

1.3.4 MaxTree

▷ `MaxTree(pol)` (функция)

finds a maximal tree in the 1-skeleton of a polytope as a list of edges.

1.3.5 CellOrient

▷ `CellOrient(pol)` (функция)

Provides inductively some orientations for cells of dimensions $1..n = dim(pol)$

Пример

```
gap> s3:=sphereAB(3);;
gap> CellOrient(s3);
[ [ [ -1, 1 ], [ -1, 1 ] ], [ [ -1, 1 ], [ -1, 1 ] ], [ [ -1, 1 ], [ -1, 1 ] ] ]
```

1.3.6 PolOrient

▷ `PolOrient(pol)` (функция)

If *pol* is orientable, gives a consistent orientation of $n-$faces ($n = dim(pol)$), otherwise returns $fail$.

1.3.7 OrientTriangulated

▷ `OrientTriangulated(pol)` (функция)

Function computes consistent orientation on simpleces of greatest dimension on a given triangulated complex *pol*. Returns array of -1, 1-s which correspond to the orientation of simplices of greatest dimension of *pol*.

1.3.8 dataPachner

▷ `dataPachner(dim, k)` (функция)

Выводит информацию о указанном движении Пахнера размерности *dim*, один из кластеров которого содержит *k* - симплексов, все информация принадлежащая данному кластеру условно обозначена буквой *l* (left). Второй кластер симплексов получающийся при преобразованиях Пахнера обозначен буквой *r* (right).

Пример

```
gap> Print(dataPachner(3,2));
rec(
  l := rec(
      pol := rec(
          faces :=
           [
              [ [ 1, 2 ], [ 1, 3 ], [ 2, 3 ], [ 1, 4 ], [ 2, 4 ], [ 3, 4 ],
                  [ 1, 5 ], [ 2, 5 ], [ 3, 5 ] ],
              [ [ 1, 4, 5 ], [ 1, 7, 8 ], [ 2, 4, 6 ], [ 2, 7, 9 ],
```

```
                [ 3, 5, 6 ], [ 3, 8, 9 ], [ 1, 2, 3 ] ],
            [ [ 1, 3, 5, 7 ], [ 2, 4, 6, 7 ] ] ],
        vertices := [ 1, 2, 3, 4, 5 ] ),
    sim := [ [ 1, 2, 3, 4 ], [ 1, 2, 3, 5 ] ],
    vnut := [ 7 ] ),
r := rec(
    pol := rec(
        faces :=
         [
            [ [ 1, 2 ], [ 1, 4 ], [ 2, 4 ], [ 1, 5 ], [ 2, 5 ], [ 4, 5 ],
                [ 1, 3 ], [ 3, 4 ], [ 3, 5 ], [ 2, 3 ] ],
            [ [ 1, 2, 3 ], [ 1, 4, 5 ], [ 2, 7, 8 ], [ 4, 7, 9 ],
                [ 3, 8, 10 ], [ 5, 9, 10 ], [ 2, 4, 6 ], [ 3, 5, 6 ],
                [ 6, 8, 9 ] ],
            [ [ 1, 2, 7, 8 ], [ 3, 4, 7, 9 ], [ 5, 6, 8, 9 ] ] ],
        vertices := [ 1, 2, 3, 4, 5 ] ),
    sim := [ [ 1, 2, 4, 5 ], [ 1, 3, 4, 5 ], [ 2, 3, 4, 5 ] ],
    vnut := [ 7, 8, 9 ] ) )
```

В каждой из этих двух записей *.l* или *.r* содержатся структуры *.pol* - задание кластера в виде шарового комплекса, *.sim* - представление кластера в виде симплексов, то есть задание симплексов через набор вершин на которые они натянуты и *.vnut* список внутренних $(n-1)-$мерных клеток шарового комплекса.

Симплициальный комплекс может быть задан как набор симплексов, каждый из которых представлен списком вершин на которые он натянут. При задании симплициальных многообразий достаточно указать симплексы размерности n. При этом существенным условием является то, что все симплексы должны быть натянуты на различные наборы вершин. Например граница четырехмерного симплекса может быть представлена как

Пример

```
gap> sim:=[ [ 1, 2, 3, 4 ], [ 1, 2, 3, 5 ], [ 1, 2, 4, 5 ],
                              [ 1, 3, 4, 5 ], [ 2, 3, 4, 5 ] ];
```

1.3.9 FromSimplexToPolytope

▷ `FromSimplexToPolytope(`*simplex*`)` (функция)

преобразует список симплексов в шаровой комплекс.

Пример

```
gap> FromSimplexToPolytope(sim);
rec(
  faces :=
    [ [ [ 1, 2 ], [ 1, 3 ], [ 2, 3 ], [ 1, 4 ], [ 2, 4 ], [ 3, 4 ], [ 1, 5 ],
          [ 2, 5 ], [ 3, 5 ], [ 4, 5 ] ],
      [ [ 1, 2, 3 ], [ 1, 4, 5 ], [ 2, 4, 6 ], [ 3, 5, 6 ], [ 1, 7, 8 ],
          [ 2, 7, 9 ], [ 3, 8, 9 ], [ 4, 7, 10 ], [ 5, 8, 10 ], [ 6, 9, 10 ] ]
        , [ [ 1, 2, 3, 4 ], [ 1, 5, 6, 7 ], [ 2, 5, 8, 9 ], [ 3, 6, 8, 10 ],
          [ 4, 7, 9, 10 ] ] ], vertices := [ 1, 2, 3, 4, 5 ] )
```

1.4 Работа с клетками

В данном разделе собраны все функции позволяющие узнать информацию связанную с какой-либо клеткой в pl-разбиении многообразия. Клетка в pl-многообразии указывается как пара чисел [dim,ind], первое из которых dim есть размерность клетки, второе ind - позиция клетки в списке pol.faces[dim]. Такую пару в описании программ мы будет называть адресом клетки или клеткой, обозначение *adr*.

1.4.1 PolBnd

▷ `PolBnd(`*pol, adr*`)` (функция)

Creating an index of boundary faces of face $adr = [dim, ind]$ of complex *pol*. *result*$[i]$ - index of (i-1)-dimensional faces of *pol* which are in the boundary of *adr*.

Пример

```
gap> PolBnd(T2,[2,3]);
[ [ 1, 2, 3, 4 ], [ 2, 4, 5, 7 ] ]
```

1.4.2 FaceComp

▷ `FaceComp(`*pol, adr*`)` (функция)

функция аналогичная функции *PolBnd*, выходные данные в функции собраны по размерностям. Сама клетка тоже включена в вывод как составляющая часть самой себя.

Пример

```
gap> FaceComp(T2,[2,3]);
rec( 0 := [ 1, 2, 3, 4 ], 1 := [ 2, 4, 5, 7 ], 2 := [ 3 ] )
```

Как видно из примера каждая именованное поле соответствует некоторой размерности.

1.4.3 StarFace

▷ `StarFace(`*pol, adr*`)` (функция)

Вычисляет все клетки которые содержат клетку *adr* в политопе *pol*. Другими словами функция вычисляет звезду указанной клетки. result.(i) содержит индексы $i-$клеток входящих в состав звезды.

Пример

```
gap> StarFace(T2,[0,3]);
rec( 1 := [ 2, 3, 6, 7 ], 2 := [ 1, 2, 3, 4 ] )
```

В данном примере 1-клетки с индексами 2, 3, 6 и 7 и 2-клетки с индексами 1, 2, 3 и 4 образуют звезду вершины 3 данного шарового разбиения.

1.4.4 PolCheckComb

▷ `PolCheckComb(`*pol*`, `*adr*`)` (функция)

check if a face of given complex is combinatorial complex that is, whether every its subface of lower dimension is uniquely determined by its vertices (and dimension)

1.5 Изменение *pl*−разбиения (не изменяющие)

Функции которые изменяют представленное разбиение без изменения многообразия.

1.5.1 PolTriangulate

▷ `PolTriangulate(`*pol*`)` (функция)

функция создает триангуляцию политопа *pol*. Заметим, что политоп после триангуляции может остаться не комбинаторным.

Пример

```
gap> PolTriangulate(sphereAB(2));
rec(
  faces := [ [ [ 1, 2 ], [ 1, 2 ], [ 1, 3 ], [ 2, 3 ], [ 1, 4 ], [ 2, 4 ] ],
      [ [ 1, 3, 4 ], [ 2, 3, 4 ], [ 1, 5, 6 ], [ 2, 5, 6 ] ] ],
  vertices := [ "A", "B", "V1", "V2" ] )
```

Triangulating a polytope.

1.5.2 FirstBoundary

▷ `FirstBoundary(`*pol*`)` (функция)

данная функция создает такое упорядочение клеток политопа, что клетки лежащие на границе имеют наименьшие индексы в списках pol.faces[dim]. Внутренний порядок клеток на границе остается неизменным (имеется в виду порядок клеток созданный индексацией внутри политопа).

1.5.3 PermFaces

▷ `PermFaces(`*pol*`, `*perm*`, `*dim*`)` (функция)

производит переупорядочение клеток размерности *dim* в политопе *pol* по перестановке *perm*.

Клетку размерности d в шаровом комплексе называем минимальной, если ее граница содержит только две клетки размерности $(d-1)$

1.5.4 ContractMiniFace

▷ `ContractMiniFace(`*pol*`, `*adr*`)` (функция)

В политопе *pol* стягивается минимальная клетка *adr*. Образом стягивания при этом является граничная клетка с меньшим индексом.

Пример

```
gap> s1:=sphereTriangul(1);
rec( faces := [ [ [ 1, 2 ], [ 1, 3 ], [ 2, 3 ] ] ], vertices := [ 1 .. 3 ] )
gap> s1:=ContractMiniFace(s1,[1,1]);
rec( faces := [ [ [ 1, 2 ], [ 1, 2 ] ] ], vertices := [ 1, 3 ] )
```

В некоторых случаях стягивание минимальной клетки невозможно, так как данные которые мы получим после стягивания могут не оказаться $pl-$комплексом. Программа проведет стягивание минимальной клетки в любом случае, даже если применение функции выведет нас из категории шаровых комплексов. По этому, применение этой функции требует доказательства того, что после стягивание не нарушится условие, что все клетки получившегося комплекса шары. Например, если в примере описанном выше попробовать стянуть в окружности $s1$ ребро, то получим следующее

Пример

```
gap> s1:=ContractMiniFace(s1,[1,1]);
rec( faces := [ [ [ 1 ] ] ], vertices := [ 1 ] )
gap> IsPolytope(s1);
false
```

Как видим выходные данные не являются политопом. По этому проверка допустимости данной операции остается за пользователем.

1.5.5 DivideFace

▷ `DivideFace(`*pol, adr, set*`)` (функция)

Пусть имеется клетка с адресом *adr* размерности $d = adr[1]$ и клетки *set* лежащие на границе клетки *adr* образуют $(d-2)-$мерную сферу S^{d-2}, тогда клетку *adr* можно разбить на две части натянув на сферу S^{d-2} диск D^{d-1} внутри клетки *adr*.

Пример

```
gap> octahedron;
rec(
  faces :=
        [ [ [ 1, 2 ], [ 2, 3 ], [ 3, 4 ], [ 1, 4 ], [ 1, 5 ], [ 2, 5 ], [ 3, 5 ],
                  [ 4, 5 ], [ 1, 6 ], [ 2, 6 ], [ 3, 6 ], [ 4, 6 ] ],
          [ [ 1, 5, 6 ], [ 2, 6, 7 ], [ 3, 7, 8 ], [ 4, 5, 8 ], [ 1, 9, 10 ],
                  [ 2, 10, 11 ], [ 3, 11, 12 ], [ 4, 9, 12 ] ], [ [ 1 .. 8 ] ] ],
        vertices := [ 1 .. 6 ] )
gap> DivideFace(octahedron,[3,1],[1,2,3,4]);
rec(
  faces :=
         [ [ [ 1, 2 ], [ 2, 3 ], [ 3, 4 ], [ 1, 4 ], [ 1, 5 ], [ 2, 5 ], [ 3, 5 ],
                  [ 4, 5 ], [ 1, 6 ], [ 2, 6 ], [ 3, 6 ], [ 4, 6 ] ],
          [ [ 1, 5, 6 ], [ 2, 6, 7 ], [ 3, 7, 8 ], [ 4, 5, 8 ], [ 1, 9, 10 ],
            [ 2, 10, 11 ], [ 3, 11, 12 ], [ 4, 9, 12 ], [ 1, 2, 3, 4 ] ],
          [ [ 1, 4, 2, 3, 9 ], [ 5, 6, 7, 8, 9 ] ] ],
```

```
                vertices := [ 1 .. 6 ] )
```

В случае когда разбивается одномерная клетка, указанная клетка дробится на две части новой вершиной, вместо множества *set* указывается имя новой вершины разбиения.

Пример

```
gap> s1:=sphereAB(1);
rec( faces := [ [ [ 1, 2 ], [ 1, 2 ] ] ], vertices := [ "A", "B" ] )
gap> DivideFace(s1,[1,1],3);
rec( faces := [ [ [ 1, 3 ], [ 1, 2 ], [ 2, 3 ] ] ], vertices := [ "A", "B", 3 ] )
```

1.5.6 UnionFace

▷ `UnionFace(`*pol, kl1, kl2*`)` (функция)

На вход функции посылаются две клетки одной и той же размерности для объединения их в одну клетку. По сути данной функцией реализуется обратная операция к функции *DivideFace* разбивающей клетку на две части.

Пример

```
gap> UnionFace(p3,[3,1],[3,2]);
rec(
  faces :=
    [ [ [ 1, 2 ], [ 1, 3 ], [ 2, 3 ], [ 1, 4 ], [ 2, 4 ], [ 3, 4 ], [ 1, 5 ],
          [ 2, 5 ], [ 3, 5 ], [ 4, 5 ] ],
      [ [ 2, 4, 6 ], [ 2, 7, 9 ], [ 4, 7, 10 ], [ 3, 5, 6 ], [ 3, 8, 9 ],
          [ 5, 8, 10 ], [ 1, 4, 5 ], [ 1, 7, 8 ] ],
      [ [ 1, 2, 4, 5, 7, 8 ], [ 3, 6, 7, 8 ] ] ],
  vertices := [ 1, 2, 3, 4, 5 ] )
```

Объединение двух клеток D_1^k и D_2^k в одну в политопе *pol* можно провести только в том случае если звезда пересечения этих клеток состоит только из D_1^k и D_2^k. Для проведения объединения функция проверяет, что данные клетки пересекаются по одному диску. Если после объединения указанных клеток комплекс перестанет быть шаровым разбиением, то функция не будет проводить объединение, на выход будет подан начальный политоп и информационная строка с пояснением почему функция отказывается работать.

Пример

```
gap> UnionFace(T2,[2,1],[2,3]);;
This faces intersected on some balls or not intersected.
I cannot union the faces in a polytope.
```

1.5.7 PolSimplify

▷ `PolSimplify(`*pol*`)` (функция)

Проводит упрощение политопа *pol* фукцией UnionFaces. Данная функция перебирает все возможности начиная с клеток максимальной размерности. Вновь появившиеся

возможности функция не исследует. По этому функцию можно запускать несколько раз если целью стоит максимально упростить политоп.

Пример

```
gap> a:=ballTriangul(3);;
gap> PolSimplify(a);
rec( faces := [ [ [ 1, 2 ], [ 1, 2 ] ], [ [ 1, 2 ], [ 1, 2 ] ], [ [ 1, 2 ] ]
     ], vertices := [ 1, 2 ] )
```

1.6 Непосредственное создание политопов

Для создания $n-$мерного диска и сферы имеются стандартные функции включенные в библиотеку

1.6.1 ballAB

▷ `ballAB(`*dim*`)` (функция)

Создает минимально возможное шаровое разбиение диска размерности *dim*.

1.6.2 sphereAB

▷ `sphereAB(`*dim*`)` (функция)

Создает минимально возможное pl-разбиение сферы размерности *dim*.

Пример

```
gap> sphereAB(3);
rec(
  faces := [ [ [ 1, 2 ], [ 1, 2 ] ], [ [ 1, 2 ], [ 1, 2 ] ],
      [ [ 1, 2 ], [ 1, 2 ] ] ], vertices := [ "A", "B" ] )
```

В библиотеку так же включена функция создающая задание диска и сферы в виде триангулированных шаровых комплексов

1.6.3 ballTriangul

▷ `ballTriangul(`*dim*`)` (функция)

Создает симплекс размерности *dim*

Пример

```
gap> ballTriangul(2);
rec( faces := [ [ [ 1, 2 ], [ 1, 3 ], [ 2, 3 ] ], [ [ 1 .. 3 ] ] ],
  vertices := [ 1 .. 3 ] )
```

1.6.4 sphereTriangul

▷ `sphereTriangul(dim)` (функция)

Триангулированная сфера создается как граница $(dim+1)-$мерного симплекса

Пример

```
gap> sphereTriangul(1);
rec( faces := [ [ [ 1, 2 ], [ 1, 3 ], [ 2, 3 ] ] ], vertices := [ 1 .. 3 ] )
```

1.7 Топологические операции

1.7.1 FreeUnionPol

▷ `FreeUnionPol(pol1, pol2)` (функция)

Свободное объединение политопов.

Пример

```
gap> FreeUnionPol(s1,s1);
rec( faces := [ [ [ 1, 2 ], [ 1, 2 ], [ 3, 4 ], [ 3, 4 ] ] ],
  vertices := [ [ 1, "A" ], [ 1, "B" ], [ 2, "A" ], [ 2, "B" ] ] )
```

Объединение происходит путем слияния соответствующих списков. При этом индексы второго политопа увеличиваются.

1.7.2 PolDoubleCone

▷ `PolDoubleCone(pol)` (функция)

Make a double cone with vertices V1 and V2 over the given polytope *pol*.

Пример

```
gap> PolDoubleCone(s1);
rec(
  faces := [ [ [ 1, 2 ], [ 1, 2 ], [ 1, 3 ], [ 2, 3 ], [ 1, 4 ], [ 2, 4 ] ],
      [ [ 1, 3, 4 ], [ 2, 3, 4 ], [ 1, 5, 6 ], [ 2, 5, 6 ] ] ],
  vertices := [ "A", "B", "V1", "V2" ] )
```

1.7.3 ConnectedSum

▷ `ConnectedSum(N, M)` (функция)

Фукнция создает связную сумму двух политопов N и M одинаковой размерности.

Пример

```
gap> s2:=sphereAB(2);;
gap> ConnectedSum(s2,s2);
rec(
  faces := [ [ [ 1, 2 ], [ 1, 2 ], [ 1, 2 ], [ 1, 2 ] ],
      [ [ 1, 3 ], [ 1, 2 ], [ 3, 4 ], [ 2, 4 ] ] ],
```

```
  vertices := [ [ 1, "A" ], [ 1, "B" ] ] )
```

1.7.4 PolProduct

▷ `PolProduct(`*pol1, pol2*`)` (функция)

Функция вычисляет декартово произведение двух шаровых комплексов pol1 и pol2. Декартово произведение для двух шаровых комплексов определяется как шаровой комплекс составленный из всевозможных шаров $D_i^s \times D_j^t$, где D_i^s и D_j^t клетки комплексов M и N, соответственно.

Пример

```
gap> PolProduct(s1,s1);
rec(
  faces :=
    [ [ [ 1, 2 ], [ 1, 2 ], [ 3, 4 ], [ 3, 4 ], [ 1, 3 ], [ 2, 4 ], [ 1, 3 ],
          [ 2, 4 ] ],
      [ [ 1, 3, 5, 6 ], [ 2, 4, 5, 6 ], [ 1, 3, 7, 8 ], [ 2, 4, 7, 8 ] ] ],
  vertices := [ [ "A", "A" ], [ "A", "B" ], [ "B", "A" ], [ "B", "B" ] ] )
```

1.7.5 ImageInPolProduct

▷ `ImageInPolProduct(`*pol1, pol2, kl1xkl2*`)` (функция)

Вспомогательная функция для декартова произведения, вычисляет адрес клетки составленной как произведение клеток $kl1$ и $kl2$ из политопов $pol1$ и $pol2$, соответственно. На вход функции адреса клеток подаются объединенные в список.

Пример

```
gap> t2:=PolProduct(s1,s1);;
gap> ImageInPolProduct(s1,s1,[[1,2],[1,1]]);
[ 2, 3 ]
```

В качестве входных данных функции могут поступить не только политопы pol1 и pol2, но и именованные списки по размерностям, каждой размерности k в котором сопоставлена мощность клеток размерности k. Данная возможность позволяет не вычислять эту информацию, при частом вызове функции для одного и того же декартова произведения.

Пример

```
gap> ls1:=rec(0:=2, 1:=2);
rec( 0 := 2, 1 := 2 )
gap> ImageInPolProduct(ls1,ls1,[[1,2],[1,1]]);
[ 2, 3 ]
```

Обратим внимание, что функция вычисляет клетку только в комплексе составленном функцией PolProduct. Если произошло какое-либо изменение комплекса, указанный адрес может оказаться не корректным.

1.7.6 PreimageInPolProduct

▷ `PreimageInPolProduct(`*pol1, pol2, imageface*`)` (функция)

По заданному образу в декартовом произведении политопов pol1 и pol2 указываем из каких клеток была составлена данная клетка imageface.

1.7.7 PolProductSyms

▷ `PolProductSyms()` (функция)

Cartesian product of two polytopes with symmetries of multipliers transferred to it. First go the symmetries of the first multiplier, then - the second.

1.7.8 PolProductSymsDict

▷ `PolProductSymsDict()` (функция)

Cartesian product of two polytopes with symmetries of multipliers transferred to it. First go the symmetries of the first multiplier, then - the second. Also returns the face dictionary.

1.7.9 PolFactorInvolution

▷ `PolFactorInvolution(`*pol, invol*`)` (функция)

pol is polytope with symmetries, *invol* is such a list of some of its symmetries (repetitions possible) that it is known that the product of symmetries in s is an involution returns the factored polytope.

1.8 Основные принципы перестроек шаровых комплексов

В данном разделе рассказывается об основных принципах перестройки шаровых комплексов связанных с изменением индексации клеток. Изменение индексации клеток является одной из основных трудностей для быстрого и удобного построения программ. Во второй версии пакета *PL* мы пользовались следующей идеей. По возможности адреса клеток которые не участвуют в операции должны остаться неизменными. Тем не менее это не всегда возможно, когда, например, клетка удаляется из политопа. Если не удается избежать смещение индексов, то по возможности необходимо сделать так, что бы индексы изменились у минимального количества индексов. Та особенность, что все построение политопа привязана к позиции соответствующих клеток в списках pol.faces[k] осложняет работу связанную с перестройкой комплексов. Начиная с третьей версии пакета *PL* будет организована возможность жесткой индексации клеток, когда позиция клетки в списке pol.faces[k] является одновременно и именем этой клетки. Это, например, будет означать, что в списках pol.faces[k] могут присутствовать пустые адреса. На данный момент присутствуют следующие функции позволяющие корректно изменять структуру политопа и прикрепленную к нему информацию.

1.8.1 DelFace

▷ `DelFace(`*pol*`, `*adr*`)` (функция)

клетка *adr* корректным образом исключается из политопа *pol*. После проведения данной операции корректность данных может быть нарушена.

Пример

```
gap> pol:=DelFace(t2,[1,2]);;
gap> IsPolytope(pol);
false
```

Функция содержит наиболее часто встречающийся код в функциях который позволяет корректно удалить все упоминания о данной клетке и подготовить данные для дальнейшей работы алгоритма.

1.8.2 wasDelFace

▷ `wasDelFace(`*pol*`, `*adr*`)` (функция)

Функция корректирует сопуствующую информацию прикрепленную к политопу *pol* которая должна была измениться после удаления одной из клеток. На вход функции подается политоп и адрес той клетки которая была удалена. Напомним, что после удаления клетки индексы больших клеток данной размерности понижаются на единицу, это изменение индексации должно быть отображено в той информации которая сопутствует данному шаровому комплексу. Если из какой-либо сопуствующей информации была удалена клетка, то будет выведено соответствующее сообщение, но изменения будут проведены. При обработке информации по 2-узле в политопе будет, в случае если удаляется 2-клетка, будет выведено соответствующее сообщение, индекс 2-клетки будет удален из списка .2knot.sheets, если на данную 2-клетку есть ссылка в .2knot.dpoints.(1kl), то соответсвующая позиция будет очищена.

Chapter 2

Подполитопы, вложения

Множество *subpol* клеток размерности *k* будем называть подполитопом, если они задают некоторое вложенное многообразие $N \subset M$. Самый простой пример подполитопа это список из одного элемента любой размерности, который задает диск внутри политопа *pol*. Тем не менее мы не будем создавать какой либо жесткой структуры обозначающей подполитоп, так как его использование будет ясным из контекста программ.

2.1 Распознающие

2.1.1 SetOfFacesBoundary

▷ `SetOfFacesBoundary(`*pol*`, `*subpol*`, `*dim*`)` (функция)

Функция выводит границу подполитопа *subpol*. На выходе будут указаны клетки внутри политопа *pol* образующие границу подполитопа *subpol*. Размерность указанных клеток равняется $dim - 1$.

Пример

```
gap> SetOfFacesBoundary(t2,[2,1],2);
[ 1, 2, 3, 4 ]
```

2.1.2 SubPolytope

▷ `SubPolytope(`*pol*`, `*subpol*`, `*dim*`)` (функция)

выделяем указанный подполитоп как самостоятельный. Отношение порядка клеток в создаваемом политопе наследуется из объемлющего политопа *pol*.

Пример

```
gap> SubPolytope(t2,[1,2],1);
rec( faces := [ [ [ 1, 2 ], [ 1, 2 ] ] ],
  vertices := [ [ "A", "A" ], [ "A", "B" ] ] )
```

2.1.3 ParallelSimplify

▷ `ParallelSimplify(`*pol, subpol, dim*`)` (функция)

В политопе *pol* производится упрощение с помощью функции UnionFaces, параллельно с этим упрощается подполитоп *subpol* размерности *dim*. Информация о клетках подполитопа помещается в список .subpol. Функция работает только для подполитопов чья размерность меньше размерности объемлющего пространства. Если вложенное подмногообразие A той же размерности, что и политоп M, можно провести параллельное упрощение с пересечением $C = A \cap (\overline{M})$.

2.1.4 PolMinusFace

▷ `PolMinusFace(`*pol, adr*`)` (функция)

cuts out a neighborhood of a face with given adress from polytope. Функция построена таким образом, что после вырезания клетки позиции старых клеток в списках pol.faces[i] и pol.vertices не изменяются.

2.1.5 PolMinusFaceDoublingMethod

▷ `PolMinusFaceDoublingMethod(`*pol, adr*`)` (функция)

Допустим, некоторая окрестность k-клетки a в pl-комплексе M обладает окрестностью эквивалентной нескольким копиям n-дисков D^n_i склеенных по представленной клетке a. Тогда можно осуществить разрез по данной клетке в pl-многообразии более экономичным способом, создав для каждого n-диска D^n_i свою копию клетки a. В качестве примера приведем букет трех отрезков, склеенный по вершине [0,1].

Пример

```
gap> buket:=rec( vertices := [ 1, 2, 3, 4 ],
faces := [ [ [ 1, 2 ], [ 1, 3 ], [ 1, 4 ] ] ] );;
gap> PolMinusFaceDoublingMethod(buket,[0,1]);
rec( vertices := [ 1, 2, 3, 4, 5, 6 ],
 faces:=[ [ [ 1, 2 ], [ 3, 5 ], [ 4, 6 ] ] ] )
```

Каждый диск D^n_i может иметь свое подразбиение внутри pl-многообразия M. При этом очевидно, что в pl-комплексе существует такое подразбиение шарового комплекса, что линк клетки a распадается на несколько связных компонент (по числу дисков D^n_i). Именно это свойство взято в качестве критерия нахождения и разделения дисков D^n_i. Данное вырезание не затрагивает границу клетки, оставляя ее на месте, при этом индексы не участвующих в разрезании клеток не изменяются.

2.1.6 PolMinusPol

▷ `PolMinusPol(`*pol, subpol, dim*`)` (функция)

Функция вырезает подполитоп *subpol* из политопа *pol*. По возможности выбирается такой способ вырезания, который будет более экономичным.

2.1.7 GlueFaces

▷ `GlueFaces(`*pol, face1, face2*`)` (функция)

Склеить две клетки в многообразие у которых общая граница. Замечание: функция не проверяет действительно ли у клеток одинаковая граница. Так же проверку, что после склейки получаются корректные данные возлагаем на пользователя. Например, функнция будет работать в следующем случае некорректно:

Пример

```
gap>d2:=ballAB(2);;
gap>GlueFaces(d2,[1,2],1);
rec(faces:=[ [ [1,2] ], [ [1] ] ],
        vertices:= ["A","B"]
```

Как мы видим из примера полученные данные уже не являются $pl-$разбиением.

2.1.8 VerticesRullGlueFace

▷ `VerticesRullGlueFace(`*pol, para, dim*`)` (функция)

Производится склейка двух клеток политопа. Клетки индексы клеток помещаются в список para, размерность dim клеток указывается отдельно.

2.1.9 VerticesRullGluePol

▷ `VerticesRullGluePol(`*pol, subpol1, subpol2, dim*`)` (функция)

В политопе указывается два набора связных подполитопов, которые необходимо склеить. Правила склеики подполитопов указываются в именах вершин (вершины с одинаковыми именами склеиваются в одну), далее все это индуцируется на клетки большей размерности. Для работы данного алгоритма необходимо, что бы у подполитопов были одинаковые pl-разбиения, а также, что бы в этом pl-разбиении содержалась хоть одна n-клетка натянутая на хотябы на (n+1) вершин.

Chapter 3

Погружения и узлы

Пакет *PL* предоставляет возможность работы с одномерными узлами и двумерными заузленными поверхностями.

3.1 Классические узлы

В пакете PL имеется возможность задания диаграммы узла двумя различными способами каждый из которых имеет свои плюсы и свои минусы.

3.1.1 Способы задания диаграммы узла.

Первый способ основан на том, что диаграмме узла сопоставляются атрибуты которые однозначно характеризуют диаграмму. Для этого присвоим имена каждой двойной точке, которые назовем образующими и выберем на узле произвольным образом начальную точку и положительное направление, причем диаграмму можно построить так, что бы отмеченная точка не стала при проекции двойной точкой. При обходе диаграммы начиная от отмеченной точки по выбранному направлению можно составить слово по следующему правилу: на начальном этапе имеем пустое слово, далее каждая встреченная двойная точка приписывается к слову справа в степени -1 если мы пришли в эту точку снизу и в степени 1 если пришли сверху. Когда мы вернемся в начальную точку, создание слова заканчивается. Дополнительно, каждая двойная точка имеет свою ориентацию которую образует выбранное нами направление. Ориентацию двойной точки предлагается строить так. В двойной точке строится два вектора v_1 и v_2, где v_1 — касательный вектор к верхней дуге диаграммы на плоскости, v_2 — к нижней. В качестве ориентации этой точки выбирается знак построенного репера. Воспользуемся данным описанием и составим необходимые атрибуты для задания узла трилистника. Для трилистника будет составлено слово $ac^{-1}ba^{-1}cb^{-1}$ и для каждой двойной точки диаграммы каждая двойная точка этой диаграммы имеет отрицательную ориентацию. Если данные ориентации заменить на противоположные, то будет создана зеркальная диаграмма к указанной. Ниже представлена диаграмма узла трилистника включенная в библиотеку пакета *PL*.

Пример

```
gap> Trefoil;
rec( kod := [ [ "a", 1 ], [ "c", -1 ], [ "b", 1 ],
              [ "a", -1 ], [ "c", 1 ], [ "b", -1 ] ],
```

```
    orient := [ [ "a", -1 ], [ "b", -1 ], [ "c", -1 ] ] )
```

Как видно из примера вся необходимая информация собрана в списках .kod и .orient. Слово которое составляется при обходе узла представлено списком .kod который содержит двухэлементные списки, первым элементом которого является имя двойной точки, вторым ± 1 в зависимости от степени соответствующей образующей в слове. Сравните слово $ac^{-1}ba^{-1}cb^{-1}$ составленное по трилистнику со списком Trefoil.kod. Далее, список .orient составлен из списка пар, первый элемент пары — имя двойной точки, второй ориентация. Задание таким образом зацеплений несколько усложняется тем, что необходимо отдельно указывать компоненты зацепления. В случае, если стоит необходимость вручную задать диаграмму обычного узла данный способ является лучшим поскольку информация легко проверяется. Предполагается, что данная диаграмма лежит на двумерной плоскости, отсюда мы будем называть этот способ задания – плоским, что бы различать два способа описания диаграмм узлов. Второй способ задания диаграммы узла основан на той идее, что диаграмма узла естественным образом создает разбиение двумерной плоскости, а следовательно аналогичное разбиение можно выбрать и на сфере S^2 после одноточечной компактификации. Отсюда, для удобства, этот способ далее будем называть сферическим. Итак, этот способ описания диаграммы заключается в создании такого шарового разбиения двумерной сферы по клеткам которого проходят дуги диаграммы узла. То есть дополнительно к $pl-$разбиению присоединяется информация о дугах диаграммы узла и о том как ведут себя дуги в районе двойной точки. В случае классических узлов достаточно указать все 1-клетки по которым проходят клетки узла. Двойные точки вычисляются уже из представленной информации, но для каждой двойной точки необходимо знать какая дуга выше, какая ниже.

Пример

```
rec(
  1knot :=
    rec(
      dpoints := rec( 1 := [ 6, 1, 3, 4 ], 2 := [ 2, 3, 5, 6 ],
          3 := [ 4, 5, 1, 2 ] ), sheets := [ 1 .. 6 ] ),
  faces := [ [ [ 1, 3 ], [ 2, 3 ], [ 1, 2 ], [ 1, 3 ], [ 2, 3 ], [ 1, 2 ] ],
      [ [ 1, 3, 5 ], [ 1, 4 ], [ 2, 4, 6 ], [ 2, 5 ], [ 3, 6 ] ] ],
  vertices := [ "a", "b", "c" ] )
```

В данном примере приведена диаграмма узла трилистника расположенная на двумерной сфере. Сначала, обратим внимание, что вся информация касающаяся узла помещена в именованный список .1knot, в котором содержатся списки .sheets всех 1-клеток $pl-$разбиения по которому проходит диаграмма и .dpoints который в качестве именованных полей содержит индексы вершин, а в качестве соответствующих объектов списки из четырех элементов в которых первая пара элементов это индексы 1-клеток входящих в верхнюю дугу диаграммы в этой точке, последняя пара — индексы 1-клеток входящих в нижнюю дугу диаграммы в данной точке. Данная конструкция неудобна для описания ее напрямую по диаграмме узла поскольку содержит излишнюю информацию, но с другой стороны предложенный способ обладает двумя преимуществами. Во-первых, это задание может напрямую работать с диаграммами зацеплений узлов. Во-вторых,

такое задание легко обобщается на двумерные заузленные поверхности, о чем будет рассказано ниже.

3.1.2 Knot1OnSphere2

▷ `Knot1OnSphere2(`*knot*`)` (функция)

По диаграмме узла создается двумерная сфера S^2, в разбиении которой указана данная диаграмма. Узел содержится в прикрепленном именованном списке .1knot, который построен в стиле задания диаграмм двумерных заузленных поверхностей.

Пример

```
gap> Knot1OnSphere2(Figure8);
rec(
  1knot :=
    rec(
      dpoints := rec( 1 := [ 8, 1, 4, 3 ], 2 := [ 4, 5, 8, 7 ],
          3 := [ 2, 3, 5, 6 ], 4 := [ 6, 7, 1, 2 ] ), sheets := [ 1 .. 8 ] ),
  faces :=
    [ [ [ 1, 4 ], [ 3, 4 ], [ 1, 3 ], [ 1, 2 ], [ 2, 3 ], [ 3, 4 ], [ 2, 4 ],
          [ 1, 2 ] ],
      [ [ 1, 3, 6 ], [ 1, 4, 7 ], [ 2, 5, 7 ], [ 2, 6 ], [ 3, 5, 8 ], [ 4, 8 ]
        ] ], vertices := [ "a", "b", "c", "d" ] )
```

3.1.3 KnotInS3

▷ `KnotInS3(`*knot*`)` (функция)

Создает трехмерную сферу в которую вложен узел. Клетки узла указаны в result.knot.

3.1.4 Reidemeister10Everywhere

▷ `Reidemeister10Everywhere(`*knot*`)` (функция)

Проверяет диаграмму узла на наличие в ней свободных петель, которые можно убрать первым движением Райдемайстера R_0^{-1}. Число 10 в названии обозначает, что применяется движение Райдемайстера, которое берет локально одну вершину и создает из него ноль вершин, если такое движение применимо.

3.1.5 ZeroLinkFromKnot

▷ `ZeroLinkFromKnot(`*knot*`)` (функция)

Не изменяя самого узла, данная функция создает такую диаграмму, которая будет иметь нулевой коэффициент зацепления с диаграммой составленной следующим образом. Для узла создается трубчатая окрестность в трехмерном пространстве. Проекция узла параллельно оси Oz на трубчатую окрестность создает новый узел (по сути дела тот же самый). Два узла созданных по предложенной диаграмме образуют зацепление с нулевым коэффициентом.

3.1.6 KnotGroup

▷ `KnotGroup(knot)` (функция)

Вычисляется фундаментальная группа узла, которая определяется как фундаментальная группа дополнения узла в трехмерной сфере S^3. Для создания группы используются соотношения Виртингера.

3.1.7 TorusKnot

▷ `TorusKnot(q, p)` (функция)

Создается диаграмма торического узла (q, p), если q и p взаимнопростые, если это не так, то будет создано соответствующее зацепление. Параметр $q > 0$ соответствует количеству нитей, а параметр p соответствует количеству оборотов.

Пример

```
gap> k:=TorusKnot(2,3);
rec( kod := [ [ 1, 1 ], [ 2, -1 ], [ 3, 1 ], [ 1, -1 ], [ 2, 1 ], [ 3, -1 ] ],
  orient := [ [ 1, 1 ], [ 2, 1 ], [ 3, 1 ] ] )
```

3.1.8 ZeifertSurface

▷ `ZeifertSurface(knot)` (функция)

создает поверхность Зейферта узла, вложенную в трехмерную сферу. Все 2-клетки отвечающие поверхности Зейферта собраны в списке .zeifert прикрепленному к политопу.

Пример

```
gap> pol:=ZeifertSurface(Knot7_7);;
gap> zeif:=SubPolytope(pol, pol.zeifert, 2);;
gap> PolOrient(zeif);
[ 1, 1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, 1 ]
```

В данном примере мы создали сферу с указанной в ней поверхностью Зейферта, выделили ее как самостоятельный политоп и проверили является ли эта поверхность ориентируемой. Тем самым мы проиллюстрировали ориентируемость полученного объекта.

3.1.9 ZeifertSurfaceWithSimplyBoundary

▷ `ZeifertSurfaceWithSimplyBoundary(knot)` (функция)

создает поверхность Зейферта узла *knot*, чья граница состоит только из двух 1-клеток (соответственно по которым и проходит сам узел).

Пример

```
gap> pol:=ZeifertSurfaceWithSimplyBoundary(Knot7_7);;
gap> SetOfFacesBoundary(pol,pol.zeifert,2);
[ 49, 159 ]
```

3.2 Двумерные заузленные поверхности

Под заузленной двумерной поверхностью (surface-knot) мы понимаем связную или не связную поверхность помещенную в четырехмерное евклидово пространство $\mathbb{R}^4$. Проекция $\pi : \mathbb{R}^4 \to \mathbb{R}^3$ общая (is generic) для заузленной поверхности T если если образ $\pi(T)$ из $\mathbb{R}^3$ локально гомеоморфен 1) простому диску 2) двум трансверсально пересекающимся дискам 3) трем трансверсально пересекающимся дискам или 4) зонтику Уитни. Точки имеющие окрестности с 2) по 4) называются двойными точкаи, тройными точками и точками ветвления общей проекции, соответственно. Множество всех точек типа 2),3) и 4) называется графом двойных точек (singularity set). Диаграммой поверхности T называют образ $\pi(T)$ с добавлением информации пересечения на графе двойных точек. Для каждой лини двойных точек должно быть определено отношение высот для листов которые входят в это ребро.

3.2.1 Задание двумерных заузленных поверхностей

Аналогично одномерным узлам, диаграммы двумерных заузленных поверхностей могут быть рассмотрены уже внутри разбиения трехмерной сферы S^3. В качестве такой диаграммы мы понимает шаровое разбиение трехмерной сферы по клеткам которой проходят листы двумерной заузленной поверхности. Вся информация о диаграмме внутри политопа собирается в прикрепленном именованном списке .2knot. Список .2knot содержит список .sheets в котором собраны индексы всех 2-клеток шарового разбиения по которым проходит диаграмма заузленной поверхности и именованный список .dpoints в котором именами выступают ребра двойных точек. В списке .2knot.dpoints каждому двойному ребру поставлен в соответствие четырехэлементный список индексов 2-клеток диаграммы лежащих в звезде данного двойного ребра. На четрыхэлементном списке выбрано следующее упорядочение, первые два индекса соответствуют 2-клеткам в звезде ребра лежащие на верхнем листе, последние два это индексы 2-клеток на нижнем. Данной информации о диаграмме двумерной заузленной поверхности достаточно чтобы вычислить тройные точки и точки ветвления. Внутри программ мы не будем проводить четкого разделения между двумерными заузленными поверхностями и двумерными узлами. Вся информация о данных структурах будет собираться в прикрепленном списке .2knot.

3.2.2 PolSimplifyWith2Knot

▷ `PolSimplifyWith2Knot(`*pol*`)` (функция)

Упрощает политоп содержащий 2-узел. В качестве упрощающей функции была выбрана функция UnionFaces, которая объединяет два шара в политопе. Так же как и функция PolSimplify данная функция не проверяет возможны ли дальнейшие упрощение политопа *pol*. Для этой проверки необходимо еще раз запустить эту функцию на вновь полученных данных.

3.2.3 SingularitySet2Knot

▷ `SingularitySet2Knot(`*pol*`)` (функция)

Singularity set — граф двойных точек диаграммы 2-узла. В графе перечислены все 1-клетки объемлющего политопа *pol* которые содеражт описываемый граф. Так же в соответствии каждой вершине сопоставлен список. Данный список строится из звезды $Star(v)$ в описываемом графе упорядоченной таким образом. Первая пара элементов принадлежит верхней линии двойных точек в этой вершине (получаемая на пересечении верхнего и среднего листов), вторая пара элементов принадлежит средней линии двойных точек (пересечение верхнего и нижнего листов) и третья пара принадлежит нижней линии двойных точек (пересечение среднего и нижнего листов). Таким образом соответсвующие списки для тройной точки будут состоять из шести элементов, для двойной точки из двух и для точки ветвления из одного.

Пример

```
gap> SingularitySet2Knot(TurnKnot(Trefoil,2));
        ...
rec( graf := [ 7, 4, 5, 6, 19, 20, 21, 22, 17, 18, 23, 8, 9, 10, 12, 45, 32,
      31, 25, 26, 51, 54, 55, 64, 112, 113, 114, 115, 134, 136, 137, 138,
      131, 130, 129, 119, 109, 118, 127, 106, 124, 105, 123, 104, 122, 102,
      99, 98, 79, 97, 78, 53, 94, 52, 59, 93, 92, 57, 73, 56, 72, 90, 70, 69,
      68, 67, 66, 65 ],
  order := rec( 1 := [ 137, 92 ], 10 := [ 12, 102 ], 11 := [ 10, 12 ],
      13 := [ 4 ], 14 := [ 22, 17, 19, 20, 104, 97 ],
      15 := [ 19, 18, 21, 22, 105, 98 ], 16 := [ 20, 21, 18, 23, 106, 99 ],
      19 := [ 25 ], 2 := [ 138, 94 ], 20 := [ 32, 31, 25, 26, 109, 102 ],
      21 := [ 23, 26 ], 23 := [ 17, 32 ], 24 := [ 31, 112 ],
      25 := [ 113, 104 ], 26 := [ 114, 105 ], 27 := [ 115, 106 ],
      30 := [ 45, 109 ], 34 := [ 45, 112 ], 36 := [ 113, 118 ],
      37 := [ 115, 119 ], 40 := [ 51, 56, 54, 53, 118, 122 ],
      41 := [ 53, 52, 55, 56, 114, 123 ], 42 := [ 54, 55, 52, 57, 119, 124 ],
      45 := [ 127, 59 ], 46 := [ 59, 57 ], 48 := [ 51 ],
      49 := [ 64, 69, 67, 66, 129, 122 ], 5 := [ 4, 9, 7, 6, 97, 92 ],
      50 := [ 66, 65, 69, 68, 130, 123 ], 51 := [ 68, 67, 70, 65, 131, 124 ],
      54 := [ 72 ], 55 := [ 79, 78, 73, 72, 134, 127 ], 56 := [ 73, 70 ],
      58 := [ 64, 79 ], 59 := [ 136, 78 ], 6 := [ 5, 6, 8, 9, 98, 93 ],
      60 := [ 137, 129 ], 61 := [ 130, 93 ], 62 := [ 138, 131 ],
      64 := [ 134, 90 ], 67 := [ 136, 90 ], 7 := [ 7, 8, 5, 10, 99, 94 ] ) )
```

3.2.4 TripleDoubleBranchPoints

▷ `TripleDoubleBranchPoints(`*pol*`)` (функция)

Для шарового разбиения многообразия, внутри которого указана диаграмма заузленной поверхнсоти вычисляются те вершины политопа, которые являются тройными точками, двойными точками или точками ветвления. По возможности для каждой точки указываются 2-клетки диаграммы группированные по принадлежности различным листам. Функция выводит именованный список с полями .triple, .double и .branch. В именованном списке result.triple каждой вершине v под списком поля .u можно узнать 2-клетки звезды на верхнем листе в тройной точке v, под списком .m — 2-клетки на среднем листе и под списком .d — на нижнем. Для двойной точки указываются только списки .u и .d. Для точки ветвления указан только список 2-клеток узла лежащих в звезде этой вершины.

Пример

```
gap> pol:=TurnKnot(Trefoil,1);;
...
gap> TripleDoubleBranchPoints(pol);
rec( branch := rec( 13 := [ 2, 6, 22, 30 ], 19 := [ 13, 16, 37, 44 ] ),
  double :=
    rec( 1 := rec( d := [ 24, 25, 41, 57 ], u := [ 22, 57, 51, 27 ] ),
      10 := rec( d := [ 28, 37, 38 ], u := [ 2, 30, 7 ] ),
      11 := rec( d := [ 28, 36, 38 ], u := [ 2, 6, 7 ] ),
      2 := rec( d := [ 23, 28, 55, 52, 55 ], u := [ 25, 41, 26 ] ),
      21 := rec( d := [ 36, 38, 43 ], u := [ 8, 12, 14 ] ),
      23 := rec( d := [ 30, 39, 48 ], u := [ 8, 12, 16 ] ),
      24 := rec( d := [ 7, 46, 47 ], u := [ 13, 44, 14 ] ),
      25 := rec( d := [ 24, 40, 41 ], u := [ 27, 39, 51 ] ),
      26 := rec( d := [ 26, 42, 27 ], u := [ 23, 24, 40 ] ),
      27 := rec( d := [ 23, 43, 52 ], u := [ 26, 42, 41 ] ),
      30 := rec( d := [ 47, 48, 51, 57 ], u := [ 19, 44, 20, 43 ] ),
      34 := rec( d := [ 46, 54, 57, 47 ], u := [ 14, 20, 19, 44 ] ) ),
  triple :=
    rec(
      14 := rec( d := [ 32, 33, 40, 41 ], m := [ 27, 30, 35, 39 ],
          u := [ 8, 9, 11, 12 ] ),
      15 := rec( d := [ 27, 34, 35, 42 ], m := [ 23, 31, 32, 40 ],
          u := [ 9, 10, 11, 12 ] ),
      16 := rec( d := [ 23, 31, 36, 43 ], m := [ 33, 34, 41, 42 ],
          u := [ 8, 9, 10, 12 ] ),
      20 := rec( d := [ 37, 38, 43, 44 ], m := [ 7, 30, 47, 48 ],
          u := [ 8, 13, 14, 16 ] ),
      5 := rec( d := [ 24, 25, 32, 33 ], m := [ 22, 27, 30, 35 ],
          u := [ 2, 3, 5, 6 ] ),
      6 := rec( d := [ 26, 27, 34, 35 ], m := [ 23, 24, 31, 32 ],
          u := [ 3, 4, 5, 6 ] ),
      7 := rec( d := [ 23, 28, 31, 36 ], m := [ 25, 26, 33, 34 ],
          u := [ 2, 3, 4, 6 ] ) ) )
```

3.2.5 IsDiagrammOf2Kont

▷ `IsDiagrammOf2Kont(`*pol*`)` (функция)

Проверяет диаграмму 2-узла вложенную в трехмерное многообразие. Для этого проверяются: 1) корректность всех ссылок на клетки политопа, 2) граф двойных точек, 3) отсутствие точек самокасания.

Пример

```
gap> pol:=TurnKnot(Figure8,-1);;

 All good!

gap> IsDiagrammOf2Kont(pol);
true
```

3.2.6 SurfaceOf2Knot

▷ `SurfaceOf2Knot(`*M3*`)` (функция)

Для заузленной поверхности указанной внутри некоторого 3-многообразия M^3 создается прообраз этой поверхности с указанием прообразов двойных ребер, тройных точек и точек ветвления. Прообраз создается как pl-комплекс, к которому прикреплена дополнительная информация содеражащаяся в именнованном списке *.preimage*, который является списком дублированных прообразов. В него входят список *.rebras* и *.points*. Список *.rebras* это список пар содержий прообразы для каждого двойного ребра, первым элементом пары является ребро-прообраз лежащее на нижнем листе, вторым, соответственно, ребро-прообраз лежащее на верхнем листе образа в диаграмме. Список *.points* состоит из списков длины 1 и 3, которые являются списками прообразов точек ветвления и тройных точек, соответственно. Причем, для тройных точек прообраз точки, лежащий на верхнем листе будет третьим в списке, на среднем - вторым и на нижнем, соответственно, первым.

Пример

3.2.7 TurnKnot

▷ `TurnKnot(`*knot, number*`)` (функция)

Создается диаграмма 2-узла вложенная в трехмерную сферу S^3 с помощью алгоритма SpunTwist на основании диаграммы одномерного узла *knot*. Число *number* задает количество оборотов данной диаграммы при осуществлении twist-движения, при этом отрицательный знак данного числа задает обращение узла в противоположном направлении, при этом если *number* указать равным нулю, тогда twist-оборотов в диаграмме не будет и мы получим простую spun-диаграмму.

Пример

```
gap> TurnKnot(Trefoil,0);
I'm trying simplify a polytope.

  ...

 All good!

rec(
  2knot :=
    rec( dpoints := rec( 11 := [ 12, 7, 9, 10 ], 12 := [ 8, 9, 11, 12 ],
          13 := [ 10, 11, 13, 8 ], 14 := [ 18, 7, 15, 16 ],
          15 := [ 14, 15, 17, 18 ], 16 := [ 16, 17, 13, 14 ] ),
      sheets := [ 7, 8, 14, 9, 15, 10, 16, 11, 17, 12, 18, 13 ] ),
  faces :=
    [ [ [ 2, 3 ], [ 1, 2 ], [ 1, 3 ], [ 2, 3 ], [ 1, 2 ], [ 5, 6 ], [ 4, 5 ],
          [ 4, 6 ], [ 5, 6 ], [ 4, 5 ], [ 1, 4 ], [ 2, 5 ], [ 3, 6 ],
          [ 1, 4 ], [ 2, 5 ], [ 3, 6 ] ],
      [ [ 1, 3, 5 ], [ 1, 4 ], [ 2, 5 ], [ 6, 8, 10 ], [ 6, 9 ], [ 7, 10 ],
          [ 11, 14 ], [ 1, 6, 12, 13 ], [ 2, 7, 11, 12 ], [ 3, 8, 11, 13 ],
          [ 4, 9, 12, 13 ], [ 5, 10, 11, 12 ], [ 13, 16 ], [ 1, 6, 15, 16 ],
```

```
          [ 2, 7, 14, 15 ], [ 3, 8, 14, 16 ], [ 4, 9, 15, 16 ],
          [ 5, 10, 14, 15 ] ],
      [ [ 7, 10, 13, 16 ], [ 1, 4, 8, 10, 12 ], [ 2, 5, 8, 11 ],
          [ 3, 6, 9, 12 ], [ 1, 4, 14, 16, 18 ], [ 2, 5, 14, 17 ],
          [ 3, 6, 15, 18 ] ] ],
  vertices := [ [ 1, "a" ], [ 1, "b" ], [ 1, "c" ], [ 2, "a" ], [ 2, "b" ],
      [ 2, "c" ] ] )
```

3.2.8 2KnotInS4

▷ `2KnotInS4(`*pol*`)` (функция)

По трехмерному политопу *pol* в котором содержится диаграмма заузленной двумерной поверхности T создается вложение этой поверхности T в четырехмерную сферу S^4.

Chapter 4

Структуры рядом

4.1 Списки

Следующие списки являются полезными при работе с политопами.

4.1.1 ConnectedSubset

▷ `ConnectedSubset(`*`lists`*`)` (функция)

Пусть список list состоит из списков. Функция ConnectedSubset выведет все индексы внутренних элементов, которые пересекаются или могут быть соединены цепочкой пересекающихся элементов с первым элементом списка lists.

Пример

```
gap> ConnectedSubset(T2.faces[1]);
[ 1, 4, 5, 8, 2, 6, 3, 7 ]
```

Наиболее частое применение данная функция находит в разделении на связные компоненты многообразий, отсюда и название этой функции.

4.1.2 SortCircle

▷ `SortCircle(`*`list`*`)` (функция)

Входной список list должен состоять из двухэлементных списков которые можно рассматривать как ребра графа. Функция *SortCircle* проведет сортировку списка *list* исходя из предположения, что описываемый им граф является циклом.

Пример

```
gap> list:=T2.faces[1]{T2.faces[2][3]};
[ [ 2, 3 ], [ 1, 4 ], [ 1, 2 ], [ 3, 4 ] ]
gap> SortCircle(list);
[ [ 2, 3 ], [ 1, 2 ], [ 1, 4 ], [ 3, 4 ] ]
gap> list;
[  ]
```

4.2 Рациональные функции

Для удобств преобразований было решено рациональные функции хранить в виде именованного списка содержащего три поля. Первые два поля для рациональной функции f это поля *f.numerator* и *f.denominator* соответствующие числителю и знаменателю рациональной фукции. Каждое из этих полей является разложением на неприводимые множители числителя и знаменателя соответственно. Третье поле это *f.coef* которое содержит числовой коэффициент рациональной функции составленный как отношение коэффициента при старшей степени числителя к коэффициенту при старшей степени знаменателя. Для неизвестных $x, y, \ldots$ используется лексикографическое упорядочение. Например, для функции $f = \frac{2x^2-8y^2}{3x^2+2y^3}$ получим следующее представление

Пример

```
gap> SimplifyRationalFunction(f);
rec( coef := 2/3, denominator := [ x_1^2+2/3*x_2^2 ],
  numerator := [ x_1-2*x_2, x_1+2*x_2 ] )
```

Для работы с данным форматом рациональных функций реализованы основные арифметические операции с использованием следующего приципа. Функции могут примать на вход как стандартный формат записи функций, так и разложенный на неприводимые множители, на выход будет подан только формат предлагаемый нами. Так как такое представление потенциально занимает много компьютерной помяти по возможности все данные всегда упрощаются.

4.2.1 SimplifyRationalFunction

▷ `SimplifyRationalFunction(`*f*`)` (функция)

Функция переводит в формат неприводимых множителей и одновремено упрощает функцию f.

4.2.2 GcdPolynomial

▷ `GcdPolynomial(`*f*`, `*g*`)` (функция)

Находится наибольший общий делитель для двух полиномов. Результат выдается в виде списка неприводимых многочленов.

Пример

```
gap> a:=x^5+y^5;;
gap> b:=(x^7+y^7)*(x+y)^2;;
gap> GcdPolynomial(a,b);
[ x_1+x_2 ]
```

4.2.3 LcmPolynomial

▷ `LcmPolynomial(`*f*`, `*g*`)` (функция)

Находится наименьшее общее кратное для двух полиномов. Результат выдается в виде списка неприводимых многочленов.

Пример

```
gap> a:=x^5+y^5;;
gap> b:=(x^7+y^7)*(x+y)^2;;
gap> LcmPolynomial(a,b);
[ x_1+x_2, x_1+x_2, x_1+x_2, x_1^4-x_1^3*x_2+x_1^2*x_2^2-x_1*x_2^3+x_2^4,
  x_1^6-x_1^5*x_2+x_1^4*x_2^2-x_1^3*x_2^3+x_1^2*x_2^4-x_1*x_2^5+x_2^6 ]
```

4.2.4 ProductRationalFunctions

▷ `ProductRationalFunctions(`*f*`,` *g*`)` (функция)
▷ `DivideRationalFunction(`*f*`,` *g*`)` (функция)

Операции умножения и деления над рациональными функциями.

4.3 Матрицы

4.3.1 Pfaffian

▷ `Pfaffian(`*mat*`)` (функция)

Вычисляется пфаффиан кососимметирческой матрицы *mat*. Для вычисления используется модифицированный аналог алгоритма Гаусса.

Пример

```
gap> PrintArray(mat);
[ [   0,  -3,  -1,   1 ],
  [   3,   0,  -1,   2 ],
  [   1,   1,   0,   3 ],
  [  -1,  -2,  -3,   0 ] ]
gap> Pfaffian(mat);
-8
```

4.4 Грассманнова алгебра

На данный момент не реализовано какой либо формальной структуры с помощью которой можно объявить грассмановы переменные как таковые. Тем не менее для задания мономов и функций на грассмановых переменных используется следующая структура. Предполагается, что грассмановы переменные нумерованы и любое упорядочение описываемое ниже является лексикографическим, если не оговорено обратное. Простой моном на грассмановых переменных задается как двухэлементный список первым элементом которого является список индексов грассмановых переменных образующих моном, втором коэффициент этого монома над некоторой алгеброй (полем). Функция на грассмановых переменных $f(x_1,x_2,\ldots,x_k) = \sum a_{i_1,\ldots i_s} x_{i_1}\cdots x_{i_s}$ описывается в как именованный список с двумя полями f.monomials и f.coeffs которые содержат слово из грассмановых переменных и коэффициент данного слова, соответственно, при этом коэффициент f.coeffs[j] принадлежит слову f.monomials[j].

4.4.1 GrassmannMonomialsProduct

▷ `GrassmannMonomialsProduct(`*mon1, mon2*`)` (функция)

Вычисляется произведение двух грассмановых мономов. Проиллюстрируем функцию на следующих примерах. $(2\ a_1a_2)*(3\ a_5a_4a_6) = -6\ a_1a_2a_3a_4a_5a_6$ и $(2\ a_1a_2)*(3\ a_5a_2a_6) = 0$.

Пример

```
gap> GrassmannMonomialsProduct([[1,2],2],[[5,4,6],3]);
[ [ 1, 2, 4, 5, 6 ], -6 ]
gap> GrassmannMonomialsProduct([[1,2],2],[[5,2,6],3]);
[ [  ], 0 ]
```

4.4.2 GrassmannProduct

▷ `GrassmannProduct(`*f, g*`)` (функция)

Произведение грассмановых функций f и g.

4.4.3 GrassmanSum

▷ `GrassmanSum(`*f, g*`)` (функция)

Сложение грассманвых функций f и g.

4.4.4 BerezinIntegral

▷ `BerezinIntegral(`*f, a*`)` (функция)

Вычисляется интеграл Березина от функции f по грассмановой образующей a (на вход функции поступает индекс соответствующей грассмановой образующей).

4.4.5 BerezinMultipleIntegral

▷ `BerezinMultipleIntegral(`*f, list*`)` (функция)

Вычисляется кратный интеграл Березина от функции f по некоторому набору грассмановых переменных указанных в списке *list*. Порядок следование индексов грассмановых переменных в списке *list* определяет порядок интегрирования.

Ссылки

Индекс