00_basico

May 25, 2021

1 Uso de polymake: Construcciones básicas

1.1 Acerca de polymake y perl

Polymake es un software de código abierto para realizar investigación en geometría poliedral. Está disponible para algunos sistemas operativos de la familia linux y para el MacOS. También puede ser utilizado como una biblioteca para C++. Polymake puede ser obtenido en el siguiente enlace web:

https://polymake.org/doku.php/start

La sintaxis de polymake está basada en una variante (dialecto) del lenguaje de programación Perl. Aspectos básicos de la sintaxis pueden consultarse en la siguiente página web:

https://polymake.org/doku.php/user_guide/tutorials/perl_intro

1.2 Definir polítipos comunes

Se puede llamar a la función `simplex` para construir un **simplex** de una dimensión determinada. El siguiente código construye un simplex de dimensión 5 y lo asigna a la variable `$p`:

[2]:
```
$p = simplex(5);
```

La variable `$p` contiene un objeto que representa al simplex. Como es usual, este objeto tiene varias propiedades asociadas. Las propiedades disponibles de un objeto en polymake puede consultarse empleando el método `list_properties`:

[8]:
```
print $p->list_properties;
```

[8]:
```
VERTICESCONE_AMBIENT_DIMCENTEREDCONE_DIMN_VERTICESSIMPLICIALITYBOUNDEDFEASIBLEPO
INTED
```

Para facilitar la lectura de este arreglo, puede usarse la función `join` de perl:

[10]:
```
print join(", ", $p->list_properties);
```

[10]:
```
VERTICES, CONE_AMBIENT_DIM, CENTERED, CONE_DIM, N_VERTICES, SIMPLICIALITY,
BOUNDED, FEASIBLE, POINTED
```

Algunas propiedades toman valores de verdadero/falso y caracterizan al polítopo. En este ejemplo, el simplex de dimensión 5 es acotado (`BOUNDED`), no vacío (`FEASIBLE`) y con punta (`POINTED`), pero no está centrado (`CENTERED`):

[22]:
```
### es acotado?
print "BOUNDED: ";
print $p->BOUNDED;
print "\n";
### es no vacío?
print "FEASIBLE: ";
print $p->FEASIBLE;
print "\n";
### es polítopo con punta (tiene al menos un vértice)?
print "POINTED: ";
print $p->POINTED;
print "\n";
### tiene centro?
print "CENTERED: ";
print $p->CENTERED;
print "\n";
```

[22]:
```
BOUNDED: true
FEASIBLE: true
POINTED: true
CENTERED: false
```

Otras propiedades contienen información específica del polítopo: Por ejemplo, la propiedad `VERTICES` retorna los vértices del polítopo, y la propiedad `N_VERTICES` retorna la cantidad de vértices:

[16]:
```
### vértices del poliedro
print $p->VERTICES;
print("---\n");
### número de vertices del poliedro
print $p->N_VERTICES;
```

[16]:
```
(6) (0 1)
(6) (0 1) (1 1)
(6) (0 1) (2 1)
(6) (0 1) (3 1)
(6) (0 1) (4 1)
(6) (0 1) (5 1)
---
6
```

En este ejemplo el listado de vértices está formateado por defecto como una matriz dispersa (*sparse matrix*). Cada fila empieza con una indicación de su dimensión, seguida de pares ordenados que indican la posición y el valor de los elementos no nulos. Para transformar esta salida a la representación usual de matrices densas, podemos usar la función `dense`:

[17]:
```
### vértices del poliedro, en formato de matriz densa
print dense($p->VERTICES);
```

[17]:
```
1 0 0 0 0 0
1 1 0 0 0 0
1 0 1 0 0 0
1 0 0 1 0 0
1 0 0 0 1 0
1 0 0 0 0 1
```

Llamando al método **properties** pueden mostrarse los valores de todas las propiedades con un solo comando:

[19]:
```
$p->properties;
```

[19]:
```
name: p
type: Polytope<Rational>
description: standard simplex of dimension 5


BOUNDED
true

CENTERED
false

CONE_AMBIENT_DIM
6

CONE_DIM
6

FEASIBLE
true

N_VERTICES
6

POINTED
true

SIMPLICIALITY
5

VERTICES
(6) (0 1)
(6) (0 1) (1 1)
```

```
(6) (0 1) (2 1)
(6) (0 1) (3 1)
(6) (0 1) (4 1)
(6) (0 1) (5 1)
```

Dependiendo del polítopo, pueden haber propiedades que no se crean directamente al construir el objeto, sino que se calculan "sobre la marcha" cuando son consultadas. Por ejemplo, la propiedad `FACETS` retorna las desigualdades que definen las facetas del polítopo:

[23]:
```
print $p->FACETS;
```

[23]:
```
0 1 0 0 0 0
0 0 1 0 0 0
0 0 0 1 0 0
0 0 0 0 1 0
0 0 0 0 0 1
1 -1 -1 -1 -1 -1
```

Cada desigualdad es del tipo `>=` y tiene la forma $a_0 + a_1 x_1 + \cdots + a_n x_n >= 0$. Las desigualdades también pueden mostrarse en un formato más amigable empleando el comando `print_constraints`:

[24]:
```
print_constraints $p->FACETS;
```

[24]:
```
0: x1 >= 0
1: x2 >= 0
2: x3 >= 0
3: x4 >= 0
4: x5 >= 0
5: -x1 - x2 - x3 - x4 - x5 >= -1
```

Al calcular la propiedad `FACETS` se han calculado además otras propiedades adicionales:

[25]:
```
$p->properties;
```

[25]:
```
name: p
type: Polytope<Rational>
description: standard simplex of dimension 5


AFFINE_HULL


BOUNDED
```

```
true

CENTERED
false

COMBINATORIAL_DIM
5

CONE_AMBIENT_DIM
6

CONE_DIM
6

FACETS
0 1 0 0 0 0
0 0 1 0 0 0
0 0 0 1 0 0
0 0 0 0 1 0
0 0 0 0 0 1
1 -1 -1 -1 -1 -1


FEASIBLE
true

FULL_DIM
true

LINEALITY_DIM
0

LINEALITY_SPACE


N_VERTICES
6

POINTED
true

SIMPLICIALITY
5

VERTICES
(6) (0 1)
(6) (0 1) (1 1)
```

```
(6) (0 1) (2 1)
(6) (0 1) (3 1)
(6) (0 1) (4 1)
(6) (0 1) (5 1)
```

La propiedad `VERTICES_IN_FACETS` indica qué vertices están contenidos en cada faceta del polítopo. Esta propiedad es importante, pues la misma determina la estructura combinatoria de un polítopo:

[27]:
```
print $p->VERTICES_IN_FACETS;
```

[27]:
```
{0 2 3 4 5}
{0 1 3 4 5}
{0 1 2 4 5}
{0 1 2 3 5}
{0 1 2 3 4}
{1 2 3 4 5}
```

Otra clase relevante de polítopos son los **hipercubos**. Pueden construirse empleando la función `cube`:

[26]:
```
$c = cube(6);
$c->properties;
```

[26]:
```
name: c
type: Polytope<Rational>
description: cube of dimension 6


AFFINE_HULL


BOUNDED
true

CONE_AMBIENT_DIM
7

CONE_DIM
7

FACETS
(7) (0 1) (1 1)
(7) (0 1) (1 -1)
(7) (0 1) (2 1)
(7) (0 1) (2 -1)
```

```
(7) (0 1) (3 1)
(7) (0 1) (3 -1)
(7) (0 1) (4 1)
(7) (0 1) (4 -1)
(7) (0 1) (5 1)
(7) (0 1) (5 -1)
(7) (0 1) (6 1)
(7) (0 1) (6 -1)


VERTICES_IN_FACETS
{0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54
56 58 60 62}
{1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55
57 59 61 63}
{0 1 4 5 8 9 12 13 16 17 20 21 24 25 28 29 32 33 36 37 40 41 44 45 48 49 52 53
56 57 60 61}
{2 3 6 7 10 11 14 15 18 19 22 23 26 27 30 31 34 35 38 39 42 43 46 47 50 51 54 55
58 59 62 63}
{0 1 2 3 8 9 10 11 16 17 18 19 24 25 26 27 32 33 34 35 40 41 42 43 48 49 50 51
56 57 58 59}
{4 5 6 7 12 13 14 15 20 21 22 23 28 29 30 31 36 37 38 39 44 45 46 47 52 53 54 55
60 61 62 63}
{0 1 2 3 4 5 6 7 16 17 18 19 20 21 22 23 32 33 34 35 36 37 38 39 48 49 50 51 52
53 54 55}
{8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31 40 41 42 43 44 45 46 47 56 57 58
59 60 61 62 63}
{0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 32 33 34 35 36 37 38 39 40 41 42 43 44 45
46 47}
{16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63}
{0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31}
{32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63}
```

Notar que, por defecto, el cubo se construye en la “forma H”, definido a partir de un sistema de desigualdades:

[28]:
```
print_constraints $c->FACETS;
```

[28]:
```
0: x1 >= -1
1: -x1 >= -1
2: x2 >= -1
3: -x2 >= -1
```

```
4: x3 >= -1
5: -x3 >= -1
6: x4 >= -1
7: -x4 >= -1
8: x5 >= -1
9: -x5 >= -1
10: x6 >= -1
11: -x6 >= -1
```

Al consultar alguna propiedad que requiera de la “forma V”, se invocan automáticamente los algoritmos de transformación correspondientes. Para polítopos grandes, esta operación puede ser muy costosa computacionalmente:

[29]:
```
print $c->VERTICES;
```

[29]:
```
1 -1 -1 -1 -1 -1 -1
1 1 -1 -1 -1 -1 -1
1 -1 1 -1 -1 -1 -1
1 1 1 -1 -1 -1 -1
1 -1 -1 1 -1 -1 -1
1 1 -1 1 -1 -1 -1
1 -1 1 1 -1 -1 -1
1 1 1 1 -1 -1 -1
1 -1 -1 -1 1 -1 -1
1 1 -1 -1 1 -1 -1
1 -1 1 -1 1 -1 -1
1 1 1 -1 1 -1 -1
1 -1 -1 1 1 -1 -1
1 1 -1 1 1 -1 -1
1 -1 1 1 1 -1 -1
1 1 1 1 1 -1 -1
1 -1 -1 -1 -1 1 -1
1 1 -1 -1 -1 1 -1
1 -1 1 -1 -1 1 -1
1 1 1 -1 -1 1 -1
1 -1 -1 1 -1 1 -1
1 1 -1 1 -1 1 -1
1 -1 1 1 -1 1 -1
1 1 1 1 -1 1 -1
1 -1 -1 -1 1 1 -1
1 1 -1 -1 1 1 -1
1 -1 1 -1 1 1 -1
1 1 1 -1 1 1 -1
1 -1 -1 1 1 1 -1
1 1 -1 1 1 1 -1
```

```
1 -1 1 1 1 1 -1
1 1 1 1 1 1 -1
1 -1 -1 -1 -1 -1 1
1 1 -1 -1 -1 -1 1
1 -1 1 -1 -1 -1 1
1 1 1 -1 -1 -1 1
1 -1 -1 1 -1 -1 1
1 1 -1 1 -1 -1 1
1 -1 1 1 -1 -1 1
1 1 1 1 -1 -1 1
1 -1 -1 -1 1 -1 1
1 1 -1 -1 1 -1 1
1 -1 1 -1 1 -1 1
1 1 1 -1 1 -1 1
1 -1 -1 1 1 -1 1
1 1 -1 1 1 -1 1
1 -1 1 1 1 -1 1
1 1 1 1 1 -1 1
1 -1 -1 -1 -1 1 1
1 1 -1 -1 -1 1 1
1 -1 1 -1 -1 1 1
1 1 1 -1 -1 1 1
1 -1 -1 1 -1 1 1
1 1 -1 1 -1 1 1
1 -1 1 1 -1 1 1
1 1 1 1 -1 1 1
1 -1 -1 -1 1 1 1
1 1 -1 -1 1 1 1
1 -1 1 -1 1 1 1
1 1 1 -1 1 1 1
1 -1 -1 1 1 1 1
1 1 -1 1 1 1 1
1 -1 1 1 1 1 1
1 1 1 1 1 1 1
```

Los polítopos de cruz pueden construirse empleando la función `cross`:

[30]:
```
$r= cross(5);
$r->properties;
```

[30]:
```
name: r
type: Polytope<Rational>
description: cross-polytope of dimension 5


BOUNDED
```

```
true

CENTERED
true

CONE_AMBIENT_DIM
6

CONE_DIM
6

N_VERTICES
10

VERTICES
(6) (0 1) (1 1)
(6) (0 1) (1 -1)
(6) (0 1) (2 1)
(6) (0 1) (2 -1)
(6) (0 1) (3 1)
(6) (0 1) (3 -1)
(6) (0 1) (4 1)
(6) (0 1) (4 -1)
(6) (0 1) (5 1)
(6) (0 1) (5 -1)


VERTICES_IN_FACETS
{0 2 4 6 8}
{1 2 4 6 8}
{0 3 4 6 8}
{1 3 4 6 8}
{0 2 5 6 8}
{1 2 5 6 8}
{0 3 5 6 8}
{1 3 5 6 8}
{0 2 4 7 8}
{1 2 4 7 8}
{0 3 4 7 8}
{1 3 4 7 8}
{0 2 5 7 8}
{1 2 5 7 8}
{0 3 5 7 8}
{1 3 5 7 8}
{0 2 4 6 9}
{1 2 4 6 9}
{0 3 4 6 9}
```

```
{1 3 4 6 9}
{0 2 5 6 9}
{1 2 5 6 9}
{0 3 5 6 9}
{1 3 5 6 9}
{0 2 4 7 9}
{1 2 4 7 9}
{0 3 4 7 9}
{1 3 4 7 9}
{0 2 5 7 9}
{1 2 5 7 9}
{0 3 5 7 9}
{1 3 5 7 9}
```

Por defecto, los polítopos de cruz se crean en la forma V.

[31]:
```
print dense($r->VERTICES);
```

[31]:
```
1 1 0 0 0 0
1 -1 0 0 0 0
1 0 1 0 0 0
1 0 -1 0 0 0
1 0 0 1 0 0
1 0 0 -1 0 0
1 0 0 0 1 0
1 0 0 0 -1 0
1 0 0 0 0 1
1 0 0 0 0 -1
```

Al consultar cualquier propiedad relativa a las facetas, el polítopo es automáticamente transformado a la forma H:

[33]:
```
print_constraints($r->FACETS);
```

[33]:
```
0: -x1 - x2 - x3 - x4 - x5 >= -1
1: x1 - x2 - x3 - x4 - x5 >= -1
2: -x1 + x2 - x3 - x4 - x5 >= -1
3: x1 + x2 - x3 - x4 - x5 >= -1
4: -x1 - x2 + x3 - x4 - x5 >= -1
5: x1 - x2 + x3 - x4 - x5 >= -1
6: -x1 + x2 + x3 - x4 - x5 >= -1
7: x1 + x2 + x3 - x4 - x5 >= -1
8: -x1 - x2 - x3 + x4 - x5 >= -1
9: x1 - x2 - x3 + x4 - x5 >= -1
10: -x1 + x2 - x3 + x4 - x5 >= -1
```

```
11: x1 + x2 - x3 + x4 - x5 >= -1
12: -x1 - x2 + x3 + x4 - x5 >= -1
13: x1 - x2 + x3 + x4 - x5 >= -1
14: -x1 + x2 + x3 + x4 - x5 >= -1
15: x1 + x2 + x3 + x4 - x5 >= -1
16: -x1 - x2 - x3 - x4 + x5 >= -1
17: x1 - x2 - x3 - x4 + x5 >= -1
18: -x1 + x2 - x3 - x4 + x5 >= -1
19: x1 + x2 - x3 - x4 + x5 >= -1
20: -x1 - x2 + x3 - x4 + x5 >= -1
21: x1 - x2 + x3 - x4 + x5 >= -1
22: -x1 + x2 + x3 - x4 + x5 >= -1
23: x1 + x2 + x3 - x4 + x5 >= -1
24: -x1 - x2 - x3 + x4 + x5 >= -1
25: x1 - x2 - x3 + x4 + x5 >= -1
26: -x1 + x2 - x3 + x4 + x5 >= -1
27: x1 + x2 - x3 + x4 + x5 >= -1
28: -x1 - x2 + x3 + x4 + x5 >= -1
29: x1 - x2 + x3 + x4 + x5 >= -1
30: -x1 + x2 + x3 + x4 + x5 >= -1
31: x1 + x2 + x3 + x4 + x5 >= -1
```

El polítopo cíclico de dimensión d con n vértices puede construirse llamando a la función `cyclic(d, n)`. Recordar que debe cumplirse $n > d$:

[36]:
```
$cy = cyclic(3, 6);
$cy->properties;
```

[36]:
```
name: cy
type: Polytope<Rational>
description: Cyclic 3-polytope on 6 vertices


BOUNDED
true

CONE_AMBIENT_DIM
4

CONE_DIM
4

N_VERTICES
6
```

```
VERTICES
1 0 0 0
1 1 1 1
1 2 4 8
1 3 9 27
1 4 16 64
1 5 25 125
```

Por defecto, el polítopo cíclo se construye en la forma V. Al consultar la propiedad `VERTICES_IN_FACETS` se calcula automáticamente la forma H:

[37]:
```
print $cy->VERTICES_IN_FACETS;
```

[37]:
```
{0 1 2}
{0 2 3}
{0 3 4}
{3 4 5}
{0 4 5}
{2 3 5}
{1 2 5}
{0 1 5}
```

Cuando un polítopo tiene dimensión 3, llamando a la propiedad `VISUAL` puede producirse una representación gráfica del mismo:

[39]:
```
$c = cube(3);
$c->VISUAL;
```

La propiedad `SCHLEGEL` construye un diagrama del Schlegel de un polítopo:

[40]:
```
$c->SCHLEGEL;
```

Los diagramas de Schlegel son muy útiles para estudiar las propiedades combinatorias de polítopos de dimensión 4:

[41]:
```
$s = simplex(4);
$s->SCHLEGEL;
```

1.3 Definir un poliedro en la forma V

Podemos crear un poliedro a partir de la envolvente convexa de un conjunto finito de puntos. Por ejemplo, definiremos a `$p` como la envolvente convexa de los puntos $(-1,-1),(-1,1),(1,-1),(1,1),(0,0)$:

[42]:
```
$p=new Polytope(POINTS=>[[1,-1,-1],[1,-1,1],[1,1,-1],[1,1,1],[1,0,0]]);
```

Notar que es necesario añadir una componente igual a 1 al inicio de cada punto, porque polymake utiliza coordenadas homogéneas para representar los objetos.

Una vez que el poliedro ha sido creado, podemos consultar sus vértices. Notar que el punto $(0, 0)$ no es un vértice de `$p$`, pues puede expresarse como combinación convexa de los otros puntos.

[43]:
```
print $p->POINTS;
print "----\n";
print $p->VERTICES;
```

[43]:
```
1 -1 -1
1 -1 1
1 1 -1
1 1 1
1 0 0
----
1 -1 -1
1 -1 1
1 1 -1
1 1 1
```

Invocando al método `DIM` se puede consultar la dimensión del poliedro:

[44]:
```
print $p->DIM;
```

[44]:
```
2
```

Para visualizar una representación gráfica de un polítopo, puede invocarse al método `VISUAL`:

[45]:
```
$p->VISUAL;
```

Para obtener información acerca de las desigualdades que definen las facetas de `$p` es necesario primero especificar un algoritmo a utilizar para la transformación entre representaciones V y H. Especificaremos el algoritmo de búsqueda en reversa lrs):

[46]:
```
prefer "lrs";
```

Ahora podemos consultar las desigualdades que definen las facetas de `$p`:

[47]:
```
print $p->FACETS;
```

[47]:
```
1 1 0
1 0 1
1 -1 0
1 0 -1
```

Cada desigualdad es del tipo `>=` y tiene la forma $a_0 + a_1x_1 + \cdots + a_nx_n >= 0$. Las desigualdades también pueden mostrarse en un formato más amigable empleando el comando `print_constraints`:

[48]:
```
print_constraints($p->FACETS);
```

[48]:
```
0: x1 >= -1
1: x2 >= -1
2: -x1 >= -1
3: -x2 >= -1
```

Para consultar cómo están ubicados los vértices en las facetas de `$p`, utilizamos el método `VERTICES_IN_FACETS`:

[49]:
```
print $p->VERTICES_IN_FACETS;
```

[49]:
```
{0 1}
{0 2}
{2 3}
{1 3}
```

1.3.1 Ejemplo 2: Permutaedro Π_3

Construyamos el permutaedro $\Pi_3 \subset \mathbb{R}^4$. Este polítopo está generado por todos los puntos cuyas coordenadas son permutaciones del conjunto $\{1, 2, 3, 4\}$:

[50]:
```
$Pi3 = new␣
↪Polytope(POINTS=>[[1,1,2,3,4],[1,1,2,4,3],[1,1,3,2,4],[1,1,3,4,2],[1,1,4,2,3],[1,1,4,3,2],
                                    ␣
↪[1,2,1,3,4],[1,2,1,4,3],[1,2,3,1,4],[1,2,3,4,1],[1,2,4,1,3],[1,2,4,3,1],
                                    ␣
↪[1,3,1,2,4],[1,3,1,4,2],[1,3,2,1,4],[1,3,2,4,1],[1,3,4,1,2],[1,3,4,2,1],
                                    ␣
↪[1,4,1,2,3],[1,4,1,3,2],[1,4,2,1,3],[1,4,2,3,1],[1,4,3,1,2],[1,4,3,2,1]]);
```

Podemos verificar que todos los puntos empleandos en la combinación convexa son vértices de Π_3:

[51]:
```
print $Pi3->VERTICES;
```

[51]:
```
1 1 2 3 4
1 1 2 4 3
1 1 3 2 4
1 1 3 4 2
1 1 4 2 3
1 1 4 3 2
1 2 1 3 4
1 2 1 4 3
1 2 3 1 4
1 2 3 4 1
```

```
1 2 4 1 3
1 2 4 3 1
1 3 1 2 4
1 3 1 4 2
1 3 2 1 4
1 3 2 4 1
1 3 4 1 2
1 3 4 2 1
1 4 1 2 3
1 4 1 3 2
1 4 2 1 3
1 4 2 3 1
1 4 3 1 2
1 4 3 2 1
```

Este polítopo tiene dimensión 3:

[52]:
```
print($Pi3->DIM);
```

[52]:
```
3
```

Calculemos ahora las desigualdades que definen las facetas de Π_3:

[53]:
```
print_constraints($Pi3->FACETS);
```

[53]:
```
0: -1/4 x1 >= -1
1: -1/7 x1 - 1/7 x2 >= -1
2: -1/9 x1 - 1/9 x2 - 1/9 x3 >= -1
3: x3 >= 1
4: -1/7 x1 - 1/7 x3 >= -1
5: -1/4 x2 >= -1
6: 1/6 x1 + 1/6 x2 + 1/6 x3 >= 1
7: x1 >= 1
8: 1/3 x1 + 1/3 x2 >= 1
9: 1/3 x1 + 1/3 x3 >= 1
10: -1/7 x2 - 1/7 x3 >= -1
11: -1/4 x3 >= -1
12: 1/3 x2 + 1/3 x3 >= 1
13: x2 >= 1
```

Al consultar la distribución de los vértices en las facetas, comprobamos que las facetas de Π_3 son cuadrados y hexágonos:

[54]:
```
print($Pi3->VERTICES_IN_FACETS);
```

```
[54]: {18 19 20 21 22 23}
{16 17 22 23}
{9 11 15 17 21 23}
{8 10 14 16 20 22}
{13 15 19 21}
{4 5 10 11 16 17}
{0 2 6 8 12 14}
{0 1 2 3 4 5}
{0 1 6 7}
{2 4 8 10}
{3 5 9 11}
{1 3 7 9 13 15}
{12 14 18 20}
{6 7 12 13 18 19}
```

No podemos graficar directamente Π_3 con el método `VISUAL`, porque es un objeto de dimensión 3 en $\mathbb{R}^4$. Pero podemos tomar una proyección del mismo en $\mathbb{R}^3$, por ejemplo, eliminando la última componente de cada vértice.

```
[55]: $q = new␣
↪Polytope(POINTS=>[[1,1,2,3],[1,1,2,4],[1,1,3,2],[1,1,3,4],[1,1,4,2],[1,1,4,3],
␣
↪[1,2,1,3],[1,2,1,4],[1,2,3,1],[1,2,3,4],[1,2,4,1],[1,2,4,3],
␣
↪[1,3,1,2],[1,3,1,4],[1,3,2,1],[1,3,2,4],[1,3,4,1],[1,3,4,2],
␣
↪[1,4,1,2],[1,4,1,3],[1,4,2,1],[1,4,2,3],[1,4,3,1],[1,4,3,2]]);
$q->VISUAL;
```

1.4 Definir un polítopo en la forma H

Puede especificarse un polítopo a través de un sistema de desigualdades empleando el método `INEQUALITIES`. El vector $[a_0, a_1, \ldots, a_n]$ representa a la desigualdad $a_0 + a_1x_1 + \cdots + a_nx_n \geq 0$.

```
[56]: $p = new Polytope(INEQUALITIES=>[[1,1,0],[1,0,1],[1,-1,0],[1,0,-1],[2,1,1]]);
```

Podemos mostrar estas desigualdades en un formato amigable empleando la función `print_constraints`:

```
[57]: print_constraints($p->INEQUALITIES);
```

```
[57]: 0: x1 >= -1
1: x2 >= -1
2: -x1 >= -1
3: -x2 >= -1
4: x1 + x2 >= -2
5: 0 >= -1
```

Notar que algunas desigualdades son redundantes. Para mostrar solamente aquellas que definen facetas, llamamos al método `FACETS`:

[58]:
```
print_constraints($p->FACETS);
```

[58]:
```
0: x1 >= -1
1: x2 >= -1
2: -x1 >= -1
3: -x2 >= -1
```

Mostramos ahora la dimensión del polítopo empleando el método `DIM`:

[59]:
```
print($p->DIM);
```

[59]:
```
2
```

Con el método `VISUAL` podemos dibujar `$p`:

[60]:
```
$p->VISUAL;
```

Podemos consultar los vértices de `$p` con el método `VERTICES`. En este caso, `polymake` ejecutará el algoritmo que se haya seleccionado para la transformación entre representaciones:

[61]:
```
print($p->VERTICES);
```

[61]:
```
1 1 1
1 -1 1
1 1 -1
1 -1 -1
```

Llamando al método `VERTICES_IN_FACETS` mostramos la distribución de los vértices de `$p` en sus facetas:

[62]:
```
print($p->VERTICES_IN_FACETS);
```

[62]:
```
{1 3}
{2 3}
{0 2}
{0 1}
```

1.4.1 Ejemplo 4: Hipercubo C_4

Conocemos que el hipercubo de dimensión cuatro C_4 puede definirse como la solución del sistema de desigualdades

$$C_4 := \left\{x \in \mathbb{R}^4 : -1 \leq x_i \leq 1, \forall i \in [4]\right\}.$$

Especificamos estas desigualdades utilizando el método `INEQUALITIES`:

[63]:
```
$c4 = new
↪Polytope(INEQUALITIES=>[[1,1,0,0,0],[1,0,1,0,0],[1,0,0,1,0],[1,0,0,0,1],
↪[1,-1,0,0,0],[1,0,-1,0,0],[1,0,0,-1,0],[1,0,0,0,-1]]);
```

Podemos constatar que todas las desigualdades definen facetas:

[64]:
```
print_constraints($c4->FACETS);
```

[64]:
```
0: x1 >= -1
1: x2 >= -1
2: x3 >= -1
3: x4 >= -1
4: -x1 >= -1
5: -x2 >= -1
6: -x3 >= -1
7: -x4 >= -1
```

Escribimos la dimensión del hipercubo:

[65]:
```
print($c4->DIM);
```

[65]:
```
4
```

Al llamar al método `VISUAL` de un polítopo de dimensión 4, `polymake` dibuja su representación por medio de un diagrama de Schlegel:

[66]:
```
$c4->VISUAL;
```

Verifiquemos que los vértices del hipercubo corresponden a todos los elementos del conjunto $\{-1, 1\}^4$:

[67]:
```
print($c4->VERTICES);
```

[67]:
```
1 1 1 1 1
1 -1 1 1 1
1 1 -1 1 1
1 -1 -1 1 1
1 1 1 -1 1
```

```
1 -1 1 -1 1
1 1 -1 -1 1
1 -1 -1 -1 1
1 1 1 1 -1
1 -1 1 1 -1
1 1 -1 1 -1
1 -1 -1 1 -1
1 1 1 -1 -1
1 -1 1 -1 -1
1 1 -1 -1 -1
1 -1 -1 -1 -1
```

Finalmente, examinemos la distribución de los vértices de `$c4` en sus facetas:

[68]:
```
print($c4->VERTICES_IN_FACETS);
```

[68]:
```
{1 3 5 7 9 11 13 15}
{2 3 6 7 10 11 14 15}
{4 5 6 7 12 13 14 15}
{8 9 10 11 12 13 14 15}
{0 2 4 6 8 10 12 14}
{0 1 4 5 8 9 12 13}
{0 1 2 3 8 9 10 11}
{0 1 2 3 4 5 6 7}
```

[ ]:
```
```