3

Relations Prolog IV

3.1 Introduction

L'OBJET DU PRÉSENT CHAPITRE est de donner une description opérationnelle de l'interprétation par Prolog IV des divers symboles de relations réservés introduits dans les concepts de base.

Veuillez lire avec attention la présente introduction, qui rappelle quelques concepts de base importants pour une bonne compréhension du fonctionnement de Prolog IV, ainsi que les conventions et notations utilisées dans le reste du chapitre.

3.1.1 Rappels des concepts de base

Sous-domaines privilégiés

Un des concepts importants de Prolog IV est la définition de ses *sous-domaines privilégiés*. Ces sous-domaines privilégiés sont en fait des sous-ensembles d'ensembles d'arbres. La notion de sous-domaine est donc très similaire à la notion de type [1].

Bien entendu, Prolog IV n'est pas un langage fortement typé, c'est-à-dire que ses variables ne sont pas typées, et aucune vérification de typage n'est effectuée lors de la compilation. Toutefois, le fait qu'une variable soit contrainte à appartenir à un sous-domaine privilégié [2] permet souvent de déclencher certains traitements.

Les formes des sous-domaines privilégiés de Prolog IV sont les suivantes (les définitions proviennent du chapitre sur les concepts de base) :

1. l'ensemble de tous les arbres,
2. l'ensemble des arbres finis,
3. l'ensemble des arbres infinis,
4. l'ensemble des listes,

1. D'un point de vue théorique, on peut noter que l'ensemble de ces sous-domaines privilégiés forme un treillis de types.
2. Ce qui peut s'apparenter à un typage de la variable.

5. l'ensemble des arbres qui ne sont pas des listes,
6. pour chaque entier positif n, l'ensemble des listes de taille n,
7. pour chaque n-uplet $A_1, \ldots, A_n$ d'intervalles IEEE, l'ensemble des liste $[a_1, \ldots, a_n]$ avec $a_i \in A_i$,
8. l'ensemble des arbres d'un seul nœud,
9. l'ensemble des arbres de plus d'un nœud,
10. l'ensemble des identificateurs,
11. l'ensemble des arbres qui ne sont pas un identificateur,
12. pour chaque intervalle IEEE A, l'ensemble A,
13. l'ensemble des arbres qui ne sont pas un nombre réel,
14. pour chaque arbre doublement rationnel a, l'ensemble $\{a\}$,
15. l'ensemble vide.

Cette liste appelle un certain nombre de définitions et remarques :

– Un intervalle IEEE est un intervalle dont les bornes, si elles existent, sont des nombres flottants de la norme IEEE 754 (ou rationnels IEEE).

– Un arbre doublement rationnel est un arbre rationnel (dont le nombre de sous-arbres est fini), dont aucun nœud n'est étiqueté par un nombre réel non rationnel. En fait, c'est un arbre qui peut être décrit exactement avec Prolog IV.

– Il est important de noter l'absence de certains ensembles de cette liste. En particulier, l'ensemble des nombres entiers n'est pas un sous-domaine privilégié. On note également qu'il n'existe pas de sous-domaines correspondant aux ensembles du type «toutes les listes dont les éléments sont des identificateurs». Ces remarques sont importantes pour comprendre les algorithmes de résolution de Prolog IV.

Quand Prolog IV est en mode «unions d'intervalles», il faut remplacer dans les définitions des sous-domaines 7 et 12 la locution «intervalles IEEE» par la locution «unions d'intervalles IEEE».

Approximation

Dans la résolution des contraintes, Prolog IV utilise des approximations, basées sur les sous-domaines privilégiés définis ci-dessus. De plus, cette résolution consiste d'un point de vue théorique à l'application d'un nombre restreint d'axiomes. Toutes les réponses de Prolog IV à des requêtes peuvent être expliquées en fonction des définitions des sous-domaines privilégiés et des axiomes. Toutefois, il n'est pas toujours très simple d'arriver à ces explications. Nous avons essayé d'expliquer tout au long du présent chapitre certains des cas les plus classiques.

Ici, nous allons simplement voir quelques cas d'approximations. Nous utiliserons ici des pseudo-termes simples, que nous expliquerons de manière informelle par rapport aux divers types de sous-domaines privilégiés.

– `X \~ 1 .`
On a ici un cas très simple, définissant le singleton $\{1\}$, qui peut être

approximé par lui-même (formes 12 ou 14).

- `X \~ cc(1,2) .`
  On définit ici l'intervalle $[1, 2]$, qui peut être approximé par lui-même (forme 12).

- `X ~ cc(0,1/3) .`
  On définit ici l'intervalle $[0, \frac{1}{3}]$, qui n'est pas un sous-domaine privilégié, puisque $\frac{1}{3}$ n'est pas un rationnel IEEE. Son approximation est donc le plus petit intervalle IEEE le contenant.

- `X ~ [1,cc(1,2),real] .`
  On a ici une liste $[a, b, c]$, avec $a = 1$, $b \in [1, 2]$ et $c \in \mathbf{R}$. Les domaines de a, b et c peuvent être approximés exactement par des intervalles IEEE (le type 12 est applicable dans les trois cas). De ce fait, la liste $[a, b, c]$ est donc un sous-domaine privilégié de type 7.

- `X ~ [1,cc(1,2),Y] .`
  On a ici une liste $[a, b, c]$, avec $a = 1$, $b \in [1, 2]$ et c un arbre quelconque. Puisque c ne peut pas être approximé par un intervalle, la liste $[a, b, c]$ n'est pas un sous-domaine privilégié de type 7, mais seulement un sous-domaine privilégié de type 6, c'est-à-dire une liste de trois éléments (où les éléments ne sont pas qualifiés).

Passons maintenant à des pseudo-termes et contraintes un peu plus complexes :

- `X \~ oo(0,1/3) n oo(1/3,1) .`
  On définit ici l'intervalle $(0, \frac{1}{3}) \cap (\frac{1}{3}, 1)$, soit l'ensemble vide (les deux intervalles sont disjoints). Ici, Prolog IV effectue l'approximation des deux intervalles séparément, et les « élargit » donc un peu autour de la valeur $\frac{1}{3}$. Ensuite, en en prenant l'intersection, on obtient un intervalle non vide (en fait, le plus petit intervalle IEEE contenant la valeur $\frac{1}{3}$.

- `X \~ binlist(1,[1,cc(1,2),Y]) .`
  On a ici défini un booléen qui doit prendre la valeur 1 (vrai) si la valeur 1 appartient à la liste $[a, b, c]$, avec $a = 1$, $b \in [1, 2]$ et c un arbre quelconque. Intuituvement, on voit que cette condition est effectivement vérifiée. Or, Prolog IV est incapable d'atteindre cette conclusion. En fait, comme on l'a vu précédemment, l'ensemble dénoté par la liste $[a, b, c]$ n'est pas un sous-domaine privilégié, et il est approximé par l'ensemble des listes de 3 éléments. Avec cette approximation, il est impossible de conclure sur l'appartenance. Pour pouvoir conclure, il suffit que la variable `Y` du pseudo-terme soit contrainte à prendre une valeur réelle. La liste $[a, b, c]$ sera alors un sous-domaine privilégié de type 7, permettant ainsi à Prolog IV de déterminer la soution avec plus de précision.

Nous avons ici essayé de présenter quelques cas extrêmes, afin de montrer qu'il faut toujours garder à l'esprit les problèmes d'approximation. Toutefois, ces problèmes surgissent dans des cas relativement rares, et dans des programmes assez « pointus », et ne représentent pas une gêne quotidienne pour la programmation.

3.1.2 Conventions et notations

Notations

– Dans toutes les descriptions de nos relations, nous adopterons les lettres suivantes pour désigner des sous-ensembles importants d'arbres :
  A : ensemble des arbres,
  B : ensemble $\{0, 1\}$,
  F : ensemble des arbres d'un nœud,
  L : ensemble des listes,
  Z : ensemble des entiers,
  R : ensemble des nombres réels.
– Nous avons dans ce document utilisé la notation anglo-saxonne pour les intervalles. La table ci-dessous montre les équivalences entre les notations française et anglo-saxonne :

Anglo-saxonne	*Française*
$[a, b]$	$[a, b]$
$[a, b)$	$[a, b[$
$(a, b]$	$]a, b]$
(a, b)	$]a, b[$
$(-\infty, a]$	$]-\infty, a]$
$(-\infty, a)$	$]-\infty, a[$
$[a, +\infty)$	$[a, +\infty[$
$(a, +\infty)$	$]a, +\infty[$
$(-\infty, +\infty)$	$]-\infty, +\infty[$

– Dans tous les exemples présentés ci-dessous, nous supposerons que Prolog IV est dans sa configuration par défaut, soit en mode syntaxique spécifique et en mode «intervalles simple». Pour tous les exemples qui utilisent les unions d'intervalles, la commande permettant le passage en mode «unions d'intervalles» est explicitement placée dans le code.

Nommage des relations

Le nommage des relations présentées dans le présent chapitre obéit à certaines règles précises. Certaines de ces règles sont rappelées ci-dessous.

– Les relations numériques de base existent toutes en deux versions : une est traitée par le solveur linéaire, et l'autre par le solveur sur les intervalles. Les noms des versions «linéaires» sont les mêmes que ceux des versions «intervalles», auxquels on a ajouté le suffixe «-lin». La table ci-dessous montre toutes ces relations, avec leurs raccourcis quand ceux-ci sont disponibles :

Intervalles	Raccourci	Linéaire	Raccourci
plus/3	.+.	pluslin/3	+
minus/3	.-.	minuslin/3	-
times/3	.*.	timeslin/3	*
div/3	./.	divlin/3	/
uplus/2	.+.	upluslin/2	+
uminus/2	.-.	uminuslin/2	-
ge/2		gelin/2	
gt/2		gtlin/2	
le/2		lelin/2	
lt/2		ltlin/2	

– Pour faire référence à l'appartenance ou à la non-appartenance à un intervalle, des relations différentes sont utilisées selon que l'intervalle est ouvert ou fermé de chaque côté. La convention de nommage utilisée est que le type de l'intervalle est représenté par deux lettres *gd*. La lettre *g* vaut `c` (pour *closed*) si l'intervalle est fermé à gauche, et `o` (pour *open*) si l'intervalle est ouvert à gauche, et la lettre *d* est définie similairement pour la droite. La table ci-dessous montre toutes les relations qui utilisent cette convention :

I	$x \in I$	$x \notin I$	$x \in I\,?$	$x \notin I\,?$
$[a, b]$	cc/3	outcc/3	bcc/4	boutcc/4
$[a, b)$	co/3	outco/3	bco/4	boutco/4
$(a, b]$	oc/3	outoc/3	boc/4	boutoc/4
(a, b)	oo/3	outoo/3	boo/4	boutoo/4

– Un certain nombre de relations de « typage » sont associées à une négation. Le symbole de cette relation négative est construit en ajoutant le préfixe `n` au symbole de la relation originale. La table ci-dessous en donne la liste :

Relation	*Négation*
identifier/1	nidentifier/1
int/1	nint/1
leaf/1	nleaf/1
list/1	nlist/1
prime/1	nprime/1
real/1	nreal/1
tree/1	ntree/1

Note : La plupart des relations ci-dessus correspondent à des sous-domaines privilégiés de Prolog IV, et sont donc susceptibles d'apparaître dans des réponses données par Prolog IV. Les exceptions sont int/1, nint/1, prime/1 et nprime/1.

– De nombreuses relations sur les intervalles sont associées à une pseudo-opération produisant un booléen qui est vrai pour les éléments de la relation initiale. Si on considère une relation r/n, cette pseudo-opération a donc une arité $n+1$, et son nom est `b`r (le préfixe `b` signifie *booléen*). La table ci-dessous liste toutes les relations qui ont une pseudo-opération associée :

Relation	*Pseudo*	*Relation*	*Pseudo*
and/2	band/3	nidentifier/1	bnidentifier/2
cc/3	bcc/3	nint/1	bnint/2
co/3	bco/3	nleaf/1	bnleaf/2
dif/2	bdif/3	nlist/1	bnlist/2
eq/2	beq/3	not/1	bnot/2
equiv/2	bequiv/3	nprime/1	bnprime/2
finite/1	bfinite/1	nreal/1	bnreal/2
ge/2	bge/3	oc/3	boc/4
gt/2	bgt/3	oo/3	boo/4
identifier/1	bidentifier/2	or/2	bor/3
impl/2	bimpl/3	outcc/3	boutcc/4
infinite/1	binfinite/2	outco/3	boutco/4
inlist/2	binlist/3	outlist/2	boutlist/3
int/1	bint/2	outoc/3	boutoc/4
le/2	ble/3	outoo/3	boutoo/4
leaf/1	bleaf/2	prime/1	bprime/2
list/1	blist/2	real/1	breal/2
lt/2	blt/3	xor/2	bxor/3

3.2 Liste alphabétique

DANS LES PAGES qui suivent, on trouvera la liste de tous les symboles r de relation auxquels Prolog IV associe une relation $\pi_4(r)$ bien précise. Pour faciliter les recherches, cette liste est donnée dans l'ordre alphabétique. La distinction entre symboles de relation et relations est nécessaire : deux symboles distincts – comme par exemple **plus** et **pluslin** – peuvent représenter une même relation mais être traités différemment dans les algorithmes de résolution de contraintes.

abs/2 — Valeur absolue

Définition : **abs/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a = |b|\}$.

Description : **abs/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique et sa valeur absolue.

Exemples :

```
>> abs(A, B).
B ~ real,
A ~ ge(0).
>> abs(1, B).
B ~ cc(-1,1).
>> abs(A, -1).
A = 1.
>> A = abs(2).
A = 2.
>> A ~ abs(cc(-2,1)).
A ~ cc(0,2).
```

Voir également : ceil/2, floor/2.

and/2 — Et

Définition : **and/2** définit la relation $\{(a, b) \in \mathbf{B}^2 \mid a = 1 \text{ et } b = 1\}$.

Description : **and/2** est une relation sur les booléens, qui est traitée par le solveur sur les intervalles. Une contrainte **and**(x, y) réussit si ses deux arguments valent 1.

Cette contrainte est normalement utilisée pour construire une conjonction d'autres contraintes exprimées par des pseudo-opérations produisant des booléens.

Autre notation : **and**(a, b) s'écrit aussi a **and** b.

Exemples :

```
>> and(A, B).
B = 1,
A = 1.
>> and(A, 0).
false.
>> and(A, bnot(A)).
false.
>> and(bcc(A,1,4), boc(A,2,5)).
A ~ oc(2,4).
```

Voir également : band/3, equiv/2, impl/2, not/1, or/2, xor/2.

arccos/2 — Arc-cosinus

Définition : **arccos/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid 0 \leq a \leq \pi \text{ et } b = \cos a\}$.

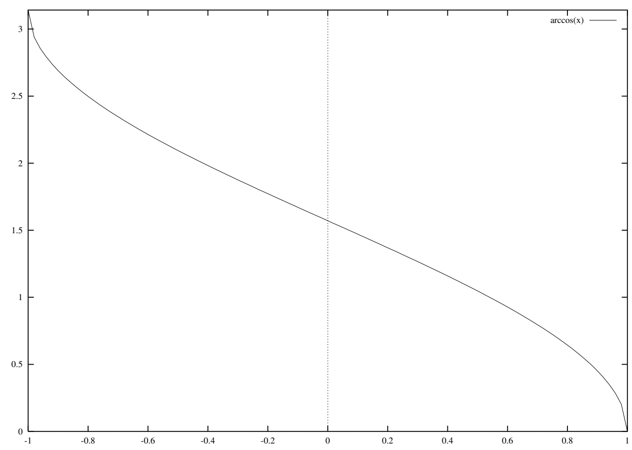
Description : **arccos/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique et son arccosinus.

Cette relation diffère de la relation **cos/2** par le fait que son premier argument est contraint à appartenir à l'intervalle $[0, \pi]$. Elle n'est donc pas sujette aux problèmes des fonctions périodiques.

Exemples :

```
>> arccos(A, B).
B ~ cc(-1,1),
A ~ co(0,`>3.1415927`).
>> arccos(0, B).
B = 1.
>> arccos(pi./.2, B).
B ~ oo(-`>4.371139e-8`,`>7.54979e-8`).
>> arccos(pi, B).
B ~ co(-1,-`>0.9999999`).
>> arccos(A, -1).
A ~ co(`>3.1415922`,`>3.1415927`).
>> A = 2.*.arccos(0).
A ~ cc(`>3.1415925`,`>3.1415927`).
>> arccos(A, 1).
A = 0.
>> A = 4.*.arccos(sqrt(2)./.2).
A ~ cc(`>3.1415922`,`>3.1415927`).
>> arccos(A, A).
A ~ cc(`>0.739085`,`>0.7390852`).
>> arccos(2.*.pi,B).
false.
```

Graphe :



Voir également : arcsin/2, arctan/2, cos/2, cot/2, pi/2, sin/2, tan/2.

arcsin/2 — Arc-sinus

Définition : **arcsin/2** définit la relation $\{(a,b) \in \mathbf{R}^2 \mid -\pi/2 \leq a \leq \pi/2 \text{ et } b = \sin a\}$.

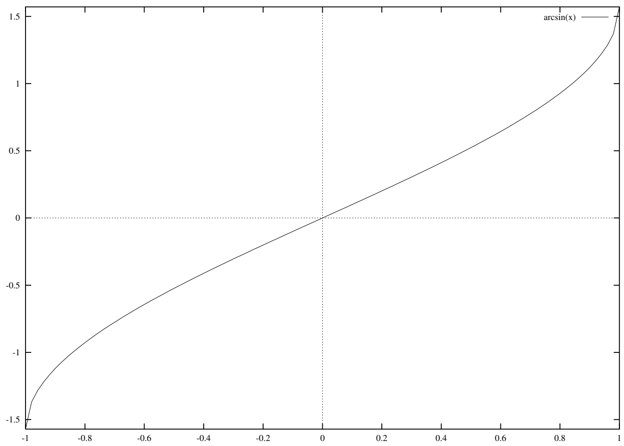
Description : **arcsin/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique et son arcsinus.

Cette relation diffère de la relation **sin/2** par le fait que son premier argument est contraint à appartenir à l'intervalle $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Elle n'est donc pas sujette aux problèmes des fonctions périodiques.

Exemples :

```
>> arcsin(A, B).
B ~ cc(-1,1),
A ~ oo(-`>1.5707963`,`>1.5707963`).
>> arcsin(0, B).
B = 0.
>> arcsin(pi./.2, B).
B ~ oc(`>0.9999999`,1).
>> arcsin(A, 0).
A = 0.
>> A = 2.*.arcsin(1).
A ~ oo(`>3.1415925`,`>3.1415927`).
>> A = 4.*.arcsin(sqrt(2)./.2).
A ~ oo(`>3.1415922`,`>3.141593`).
>> arcsin(A, A).
A ~ oo(-`>0.017607333`,`<0.01740003`).
>> arcsin(pi,B).
false.
```

Graphe :



Voir également : arccos/2, arctan/2, cos/2, cot/2, pi/2, sin/2, tan/2.

arctan/2 — Arc-tangente

Définition : **arctan/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid -\pi/2 < a < \pi/2 \text{ et } b = \tan a\}$.

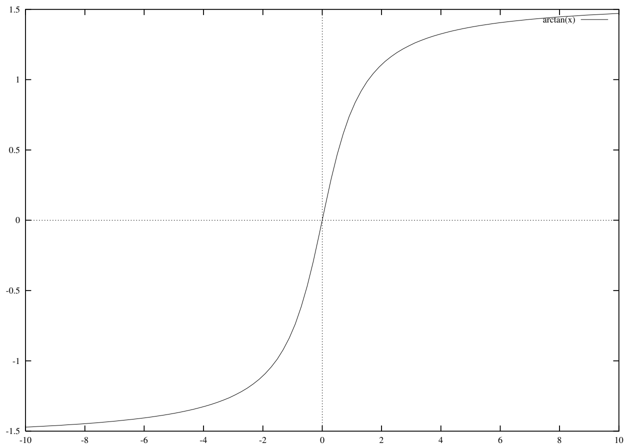
Description : arctan/2 est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique et son arctangente.

Cette relation diffère de la relation **tan/2** par le fait que son premier argument est contraint à appartenir à l'intervalle $(-\frac{\pi}{2}, \frac{\pi}{2})$. Elle n'est donc pas sujette aux problèmes des fonctions périodiques.

Exemples :

```
>> arctan(A, B).
B ~ real,
A ~ oo(-`>1.5707963`,`>1.5707963`).
>> arctan(0, B).
B = 0.
>> arctan(pi./.4, B).
B ~ oo(`<0.9999999`,`>1.0000001`).
>> arctan(A, 0).
A = 0.
>> A = 4.*.arctan(1).
A ~ cc(`>3.1415925`,`>3.1415927`).
>> arctan(pi,B).
false.
```

Graphe :



Voir également : arccos/2, arcsin/2, cos/2, cot/2, pi/2, sin/2, tan/2.

band/3 — Et

Définition : **band/3** définit la relation $\{(a, b, c) \in \mathbf{B}^3 \mid a = 1 \text{ ssi } (b, c) \in \pi_4(\text{and}/2)\}$.

Description : band/3 est une relation sur les booléens, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre trois booléens, dont le premier est la conjonction des deux autres.

Cette relation peut être utilisée pour construire des expressions booléennes complexes, ou encore comme connecteur dans la construction d'expressions numériques complexes.

Autre notation : **band**(a, b, c) s'écrit aussi $a = b$ **band** c.

Exemples :

```
>> band(A, B, C).
C ~ cc(0,1),
B ~ cc(0,1),
A ~ cc(0,1).
>> band(0, B, C).
C ~ cc(0,1),
B ~ cc(0,1).
>> band(1, B, C).
C = 1,
B = 1.
>> band(A, 0, C).
A = 0,
C ~ cc(0,1).
>> band(0, 1, C).
C = 0.
>> eq(B, C), band(A, B, C).
B = C,
A ~ cc(0,1),
C ~ cc(0,1).
```

Voir également : and/2, bequiv/3, bimpl/3, bnot/2, bor/3, bxor/3.

bcc/4 — Appartenance à un intervalle

Définition : **bcc/4** définit la relation $\{(a, b, c, d) \in \mathbf{B} \times \mathbf{R}^3 \mid a = 1 \text{ ssi } (b, c, d) \in \pi_4(\text{cc/3})\}$.

Description : **bcc/4** est une relation liant une variable booléenne à trois variables numériques, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur appartienne à un intervalle fermé des deux côtés de la forme $[c, d]$.

En associant une valeur booléenne à la valeur de vérité d'une contrainte d'appartenance à un intervalle, cette relation permet soit de construire des expressions numériques complexes, soit de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> bcc(A, B, C, D).
D ~ real,
C ~ real,
B ~ real,
A ~ cc(0,1).
>> bcc(A, B, 0, 1).
B ~ real,
A ~ cc(0,1).
>> bcc(1, B, 0, 2).
B ~ cc(0,2).
>> bcc(A, 0, 0, 2).
A = 1.
>> bcc(A, 1, 0, 2).
A = 1.
>> bcc(A, 2, 0, 2).
A = 1.
>> set_prolog_flag(interval_mode,union).
true.
>> bcc(0, B, 0, 2).
B ~ lt(0)u gt(2).
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : La relation **bcc/4** est une des pseudo-opérations produisant des booléens. Quelques conseils rapides sur l'utilisation de ces relations sont donnés ici. Ces conseils sont également valables pour les autres pseudo-opérations produisant des booléens.

En utilisant **bcc/4** dans une disjonction, aucune réduction de domaine des variables numériques n'est effectuée tant qu'il n'est pas possible de déterminer la valeur de la variable booléenne. Dans l'exemple suivant, la réduction attendue est effectuée, car les valeurs des variables booléennes sont connues :

```
>> or(bcc(A,1,2),bcc(A,6,7)), cc(A, 5,10).
A ~ cc(6,7).
```

Par contre, dans l'exemple ci-dessous, aucune réduction n'est effectuée, car il est impossible de donner des valeurs aux variables booléennes :

```
>> or(bcc(A,1,2),bcc(A,6,7)), A ~ cc(1,10).
A ~ cc(1,10).
```

De manière à obtenir une réduction dans un tel cas, il est préférable d'utiliser la relation **u/3**, dont la propagation est immédiate (sans passer par des variables booléennes). Dans ce cas, on utilise la relation **cc/3** au lieu de **bcc/4** :

```
>> A ~ cc(1,2) u cc(4,5), A ~ cc(1,10).
A ~ cc(1,5).
```

Dans les contraintes comme **bcc/4**, le premier argument est un booléen. Toutefois, ce booléen est également un réel, et il est donc possible de l'utiliser dans des contraintes numériques. L'une des applications immédiates est de construire un « opérateur de cardinalité ». Le but est ici de spécifier qu'un certain nombre de contraintes parmi une liste donnée doivent être vérifiées:

```
card([],0).
card([X|L], X .+. S) :-
  card(L, S).
```

Le programme ci-dessus se contente de calculer la somme des éléments d'une liste. Cette somme représente ici le nombre de contraintes vérifiées. Dans le premier exemple, on donne une liste de 4 contraintes, en demandant que 2 d'entre elles soient vérifiées :

```
>> L ~ [ bcc(X,1,2), bcc(X,2,3), bcc(X,2,4), bcc(X,3,4)],
       card(L,2).
L ~ [cc(0,1),cc(0,1),cc(0,1),cc(0,1)],
X ~ real.
```

Le résultat n'étant pas assez précis, il nous faut effectuer une énumération sur `L` pour savoir quelles contraintes sont vérifiées :

```
>> L ~ [ bcc(X,1,2), bcc(X,2,3), bcc(X,2,4), bcc(X,3,4)],
       card(L,2), boolsplit(L).
L = [0,0,1,1],
X ~ oc(3,4);
L = [0,1,1,0],
X ~ oo(2,3).
```

Il y a donc deux combinaisons dans lesquelles deux des quatre contraintes sont vérifiées. Pour terminer, on peut vouloir toutes les solutions, pour tous les nombres possible de contraintes vérifiées. Il nous faut alors ajouter une énumération sur les nombre de contraintes vérifiées :

```
>> L ~ [ bcc(X,1,2), bcc(X,2,3), bcc(X,2,4), bcc(X,3,4)],
       card(L,N), intsplit([N]), boolsplit(L).
N = 0,
L = [0,0,0,0],
X ~ real;
N = 1,
L = [1,0,0,0],
X ~ co(1,2);
N = 2,
L = [0,0,1,1],
X ~ oc(3,4);
N = 2,
L = [0,1,1,0],
X ~ oo(2,3);
N = 3,
X = 3,
L = [0,1,1,1];
N = 3,
X = 2,
L = [1,1,1,0].
```

Voir également : bco/4, boc/4, boo/4, boutcc/4, cc/3, outcc/3.

bco/4 — Appartenance à un intervalle

Définition : **bco/4** définit la relation $\{(a, b, c, d) \in \mathbf{B}\times\mathbf{R}^3 \mid a = 1 \text{ ssi } (b, c, d) \in \pi_4(\text{co/3})\}$.

Description : **bco/4** est une relation liant une variable booléenne à trois variables numériques, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur appartienne à un intervalle fermé à gauche et ouvert à droite de la forme $[c, d)$.

En associant une valeur booléenne à la valeur de vérité d'une contrainte d'appartenance à un intervalle, cette relation permet soit de construire des expressions numériques complexes, soit de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> bco(A, B, C, D).
D ~ real,
C ~ real,
B ~ real,
A ~ cc(0,1).
>> bco(A, B, 0, 1).
B ~ real,
A ~ cc(0,1).
>> bco(1, B, 0, 2).
B ~ co(0,2).
>> bco(A, 0, 0, 2).
A = 1.
>> bco(A, 1, 0, 2).
A = 1.
>> bco(A, 2, 0, 2).
A = 0.
>> set_prolog_flag(interval_mode,union).
true.
>> bco(0, B, 0, 2).
B ~ lt(0)u ge(2).
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : Voir **bcc/4** pour de plus amples informations concernant l'utilisation des prédicats de vérification d'appartenance à des intervalles.

Voir également : bcc/4, boc/4, boo/4, boutco/4, co/3, outco/3.

bdif/3 — Inégalité

Définition : **bdif/3** définit la relation $\{(a, b, c) \in \mathbf{B} \times \mathbf{A}^2 \mid a = 1 \text{ ssi } (b, c) \in \pi_4(\text{dif/2})\}$.

Description : **bdif/3** est une relation liant une variable booléenne à deux variables quelconques, qui est traitée par le solveur général. Elle établit une relation entre un booléen et le fait que deux valeurs soient distinctes.

En associant une valeur booléenne à la valeur de vérité d'une diséquation, cette relation permet soit de construire des contraintes complexes, soit de construire des contraintes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> bdif(A, B, C).
C ~ tree,
B ~ tree,
A ~ cc(0,1).
>> bdif(A, 1, 2).
A = 1.
>> bdif(A, 1, 1).
A = 0.
```

Dans le cas suivant, la réduction est impossible.

```
>> bdif(A, 1, A).
A ~ cc(0,1).
```

On peut énumérer les valeurs possibles de `A` pour se rendre compte que la contrainte n'a pas de solution.

```
>> bdif(A, 1, A), boolsplit([A]).
false.
```

Par contre, elle est ici possible, puisque `id` n'appartient pas au le domaine de la variable `A`.

```
>> bdif(A, id, A).
A = 1.
>> bdif(A, f(X), f(Y)).
Y ~ tree,
X ~ tree,
A ~ cc(0,1).
```

La réduction ne s'effectue pas dans la contrainte suivante, car les domaines de `X` et `Y` sont inchangés par la pose de la contrainte `dif`.

```
>> bdif(A, f(X), f(Y)), dif(X,Y).
Y ~ tree,
X ~ tree,
A ~ cc(0,1).
>> bdif(1, foo, X).
X ~ tree.
>> bdif(0, foo, X).
X = foo.
```

Note : D'un point de vue opérationnel, **bdif/3** est fortement lié à `beq/3`, et pourrait en fait être défini de la manière suivante :

```
bdif(A, X, Y) :-
  A ~ bnot(beq(X,Y)).
```

Voir beq/3 pour des commentaires sur l'utilisation de bdif/3.

Voir également : beq/3, dif/2, eq/2.

beq/3 — Egalité

Définition : **beq/3** définit la relation $\{(a, b, c) \in \mathbf{B} \times \mathbf{A}^2 \mid a = 1 \text{ ssi } (b, c) \in \pi_4(\text{eq/2})\}$.

Description : `beq/3` est une relation liant une variable booléenne à deux variables quelconques, qui est traitée par le solveur général. Elle établit une relation entre un booléen et le fait que deux valeurs soient égales.

En associant une valeur booléenne à la valeur de vérité d'une équation, cette relation permet soit de construire des contraintes complexes, soit de construire des contraintes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> beq(A, B, C).
C ~ tree,
B ~ tree,
A ~ cc(0,1).
>> beq(A, 1, 2).
A = 0.
>> beq(A, 1, 1).
A = 1.
>> beq(A, 1, A).
A ~ cc(0,1).
>> beq(A, 1, A), boolsplit([A]).
A = 0;
A = 1.
```

Ici, une conclusion est possible, puisque les domaines de `id` et de la variable `A` (qui est une variable booléenne) sont disjoints :

```
>> beq(A, id, A).
A = 0.
>> beq(A, f(X), f(Y)).
Y ~ tree,
X ~ tree,
A ~ cc(0,1).
```

La pose d'une diséquation ne permet souvent pas de réduire le domaine d'une variable, et ne déclenche donc pas de réduction au niveau de la variable booléenne :

```
>> beq(A, f(X), f(Y)), dif(X,Y).
Y ~ tree,
X ~ tree,
A ~ cc(0,1).
>> beq(A, f(X), f(Y)), X = Y .
X = Y,
A = 1,
Y ~ tree.
>> beq(1, foo, X).
X = foo.
>> beq(0, foo, X).
X ~ tree.
```

Note : Contrairement aux autres pseudo-opérations produisant des booléens, les relations **beq/3** et **bdif/3** peuvent réagir à l'égalité entre variables. Par exemple, dans l'exemple suivant, on obtient une réduction de la variable booléenne bien que les domaines des autres variables demeurent inchangés :

```
>> beq(A, f(X), f(Y)), X = Y .
X = Y,
A = 1,
Y ~ tree.
```

Par contre, comme nous l'avons montré dans les exemples, les relations **beq/3** et **bdif/3** ne détectent pas les non-égalités entre variables dans le cas général (c'est-à-dire sans que leurs domaines ne soient disjoints). Par exemple, nous avons :

```
>> beq(A, f(X), f(Y)), dif(X,Y).
Y ~ tree,
X ~ tree,
A ~ cc(0,1).
```

Toutefois, le solveur linéaire permet de déduire des non-égalités entre variables, et ces non-égalités peuvent être remontées à **beq/3** et **bdif/3**, comme on le voit dans les deux exemples suivants :

```
>> beq(A, X, X+1).
A = 0,
X ~ real.
>> bdif(A, X, X+1).
A = 1,
X ~ real.
```

Voir également : bdif/3, dif/2, eq/2.

bequiv/3 — Equivalence

Définition : **bequiv/3** définit la relation $\{(a, b, c) \in \mathbf{B}^3 \mid a = 1 \text{ ssi } (b, c) \in \pi_4(\text{equiv/2})\}$.

Description : **bequiv/3** est une relation sur les booléens, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre trois booléens, dont le premier est vrai (1) si les deux autres sont équivalents.

Cette relation peut être utilisée pour construire des expressions booléennes complexes, ou encore comme connecteur dans la construction d'expressions numériques complexes.

Autre notation : **bequiv**(a, b, c) s'écrit aussi $a = b$ **bequiv** c.

Exemples :

```
>> bequiv(A, B, C).
C ~ cc(0,1),
B ~ cc(0,1),
A ~ cc(0,1).
>> bequiv(A, 1, 1).
A = 1.
>> bequiv(1, B, C).
B = C,
C ~ cc(0,1).
```

Ici, on n'a pas de réduction, à cause de l'indépendance entre la relation **bequiv/3** et la contrainte d'égalité implicite :

```
>> bequiv(A, B, B).
B ~ cc(0,1),
A ~ cc(0,1).
>> bequiv(0, B, C).
C ~ cc(0,1),
B ~ cc(0,1).
>> bequiv(0, 1, C).
C = 0.
```

Note : Nous avons vu dans les exemples que la contrainte `bequiv(A,B,B)` ne permettait pas de réduire le domaine de `A`. Cela est dû au fait que la relation **bequiv/3** ne permet pas de détecter les égalités entre variables. Il est possible d'écrire une version plus puissante de **bequiv/3** en utilisant la relation **beq/3** qui, elle, détecte ces égalités :

```
my_bequiv(A, B, C) :-
  B ~ 0 u 1 ,
  C ~ 0 u 1 ,
  beq(A, B, C).
```

Les deux premières contraintes contraignent les variables `B` et `C` à prendre des valeurs booléennes, et la troisième contraint ces mêmes variables à être égales. On a alors des déductions plus puissantes :

```
>> my_bequiv(A,B,B).
A = 1,
B ~ cc(0,1).
```

Voir également : band/3, bimpl/3, bnot/2, bor/3, bxor/3,equiv/2.

bfinite/2 — Etre fini

Définition : **bfinite/2** définit la relation $\{(a,b) \in \mathbf{B} \times \mathbf{A} \mid a = 1 \text{ ssi } b \in \pi_4(\text{finite/1})\}$.

Description : **bfinite/2** est une relation liant une variable booléenne à une variable quelconque, qui est traitée par le solveur général et le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur représente ou non un arbre fini.

En associant une valeur booléenne à une information de typage d'une variable, cette relation permet par exemple de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> bfinite(A,B).
B ~ tree,
A ~ cc(0,1).
>> bfinite(0,B).
B ~ infinite.
>> bfinite(1,B).
B ~ finite.
>> bfinite(A,cc(1,3)).
A = 1.
>> X = f(X), bfinite(A,X).
A = 0,
X = f(X).
```

Voir également : binfinite/2, finite/1, infinite/1.

bge/3 ______ Supérieur ou égal

Définition : **bge/3** définit la relation $\{(a, b, c) \in \mathbf{B} \times \mathbf{R}^2 \mid a = 1 \text{ ssi } (b, c) \in \pi_4(\text{ge/2})\}$.

Description : **bge/3** est une relation liant une variable booléenne à deux variables numériques, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur soit ou non supérieure ou égale à une autre.

En associant une valeur booléenne à la valeur de vérité d'une comparaison numérique, cette relation permet de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors d'effectuer une partition sur le domaine de la variable.

Exemples :

```
>> bge(A,B,C).
C ~ real,
B ~ real,
A ~ cc(0,1).
>> bge(0, B, C).
C ~ real,
B ~ real.
>> bge(1, B, C).
C ~ real,
B ~ real.
>> bge(1, B, 2).
B ~ ge(2).
>> bge(A, cc(1,3), cc(2,4)).
A ~ cc(0,1).
>> bge(A, cc(1,2), cc(3,4)).
A = 0.
>> bge(A, cc(5,6), cc(6,7)).
A ~ cc(0,1).
>> bge(A, cc(6,7), cc(5,6)).
A = 1.
>> A ~ bge(X,2), boolsplit([A]).
A = 0,
X ~ lt(2);
A = 1,
X ~ ge(2).
```

Note : Voir **bcc/4** pour de plus amples informations concernant l'utilisation de ce type de contraintes dans des contraintes complexes.

Voir également : bgt/3, ble/3, blt/3, ge/2.

bgt/3 — Supérieur

Définition : **bgt/3** définit la relation $\{(a, b, c) \in \mathbf{B} \times \mathbf{R}^2 \mid a = 1 \text{ ssi } (b, c) \in \pi_4(\text{gt/2})\}$.

Description : **bgt/3** est une relation liant une variable booléenne à deux variables numériques, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur soit ou non supérieure strictement à une autre.

En associant une valeur booléenne à la valeur de vérité d'une comparaison numérique, cette relation permet de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors d'effectuer une partition sur le domaine de la variable.

Exemples :

```
>> bgt(A, B, C).
C ~ real,
B ~ real,
A ~ cc(0,1).
>> bgt(0, B, C).
C ~ real,
B ~ real.
>> bgt(1, B, C).
C ~ real,
B ~ real.
>> bgt(1, B, 2).
B ~ gt(2).
>> bgt(A, cc(1,3), cc(2,4)).
A ~ cc(0,1).
>> bgt(A, cc(1,2), cc(3,4)).
A = 0.
>> bgt(A, cc(5,6), cc(6,7)).
A = 0.
>> bgt(A, cc(6,7), cc(5,6)).
A ~ cc(0,1).
>> A ~ bgt(X,2), boolsplit([A]).
A = 0,
X ~ le(2);
A = 1,
X ~ gt(2).
```

Note : Voir **bcc/4** pour de plus amples informations concernant l'utilisation de ce type de contraintes dans des contraintes complexes.

Voir également : bge/3, ble/3, blt/3, gt/2.

bidentifier/2 — Etre un identificateur

Définition : **bidentifier/2** définit la relation $\{(a, b) \in \mathbf{B} \times \mathbf{F} \mid a = 1 \text{ ssi } b \in \pi_4(\text{identifier/1})\}$.

Description : **bidentifier/2** est une relation liant une variable booléenne à une variable quelconque, qui est traitée par le solveur général et le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur représente ou non un identificateur.

En associant une valeur booléenne à une information de typage d'une variable, cette relation permet par exemple de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> bidentifier(A,B).
B ~ tree,
A ~ cc(0,1).
>> bidentifier(0,B).
B ~ nidentifier.
>> bidentifier(1,B).
B ~ identifier.
>> bidentifier(A,list).
A ~ cc(0,1).
>> list(L), bidentifier(A,L), boolsplit([A]).
A = 0,
L ~ [tree|list];
A = 1,
L = [].
>> bidentifier(B, toto).
B = 1.
>> bidentifier(B, toto(1)).
B = 0.
```

Voir également : bnidentifier/2, identifier/1, nidentifier/1.

bimpl/3 — Implication

Définition : **bimpl/3** définit la relation $\{(a, b, c) \in \mathbf{B}^3 \mid a = 1 \text{ ssi } (b, c) \in \pi_4(\text{impl/2})\}$.

Description : **bimpl/3** est une relation sur les booléens, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre trois booléens, dont le premier est vrai si le troisième est impliqué par le second.

Cette relation peut être utilisée pour construire des expressions booléennes complexes, ou encore comme connecteur dans la construction d'expressions numériques complexes.

Autre notation : **bimpl**(a, b, c) s'écrit aussi $a = b$ **bimpl** c.

Exemples :

```
>> bimpl(A, B, C).
C ~ cc(0,1),
B ~ cc(0,1),
A ~ cc(0,1).
>> bimpl(0, B, C).
C = 0,
B = 1.
>> bimpl(1, B, C).
C ~ cc(0,1),
B ~ cc(0,1).
>> bimpl(A, 0, C).
A = 1,
C ~ cc(0,1).
>> bimpl(A, 1, C).
A = C,
C ~ cc(0,1).
>> bimpl(A, B, 0).
B ~ cc(0,1),
A ~ cc(0,1).
>> bimpl(A, B, 1).
A = 1,
B ~ cc(0,1).
>> eq(B, C), bimpl(A, B, C).
B = C,
A ~ cc(0,1),
C ~ cc(0,1).
```

Voir également : band/3, bequiv/3, bnot/2, bor/3, bxor/3, impl/2.

binfinite/2 — Etre infini

Définition : **binfinite/2** définit la relation $\{(a, b) \in \mathbf{B} \times \mathbf{A} \mid a = 1 \text{ ssi } b \in \pi_4(\text{infinite/1})\}$.

binfinite/2 est une relation liant une variable booléenne à une variable quelconque, qui est traitée par le solveur général et le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur représente ou non un terme infini.

En associant une valeur booléenne à une information de typage d'une variable, cette relation permet par exemple de construire des contraintes complexes.

Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> binfinite(A,B).
B ~ tree,
A ~ cc(0,1).
>> binfinite(0, A).
A ~ finite.
>> binfinite(1, B).
B ~ infinite.
>> binfinite(A, cc(1,3)).
A = 0.
>> X = f(X), binfinite(A, X).
A = 1,
X = f(X).
>> binfinite(0,[X]).
X ~ finite.
```

Voir également : bfinite/2, finite/1, infinite/1.

binlist/3 — Appartenance à une liste

Définition : **binlist/3** définit la relation $\{(a, b, c) \in \mathbf{B} \times \mathbf{A} \times \mathbf{L} \mid a = 1 \text{ ssi } (b, c) \in \pi_4(\text{inlist/2})\}$.

Description : **binlist/3** est une relation liant un booléen, un arbre quelconque et une liste, qui est traitée par le solveur général. Elle établit une relation entre un booléen et le fait qu'un élément appartienne ou non à une liste.

En associant une valeur booléenne à la valeur de vérité d'une contrainte d'appartenance à une liste, cette relation permet de construire des contraintes complexes, et d'utiliser les variables booléennes dans des énumérations.

Exemples :

```
>> binlist(A, B, C).
B ~ tree,
C ~ list,
A ~ cc(0,1).
>> binlist(0, B, C).
B ~ tree,
C ~ list.
>> binlist(1, B, C).
B ~ tree,
C ~ list.
>> binlist(A, 1, [1,2,3]).
A = 1.
>> binlist(A, B, []).
A = 0,
B ~ tree.
>> binlist(A, ge(4), [1,2,3]).
A = 0.
>> binlist(A, B, [1,2,3]).
B ~ real,
A ~ cc(0,1).
>> set_prolog_flag(interval_mode,union).
true.
>> binlist(A, B, [1,2,3]).
B ~ real,
A ~ 0 u 1.
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : Voir **inlist/2** pour des remarques importantes sur le fonctionnement des contraintes d'appartenance à une liste.

Voir également : boutlist/3, inlist/2, outlist/2.

bint/2 — Etre un entier

Définition : **bint/2** définit la relation $\{(a, b) \in \mathbf{B} \times \mathbf{R} \mid a = 1 \text{ ssi } b \in \pi_4(\text{int/1})\}$.

Description : **bint/2** est une relation liant une variable booléenne à une variable numérique, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur représente ou non un entier.

En associant une valeur booléenne à une information de typage d'une variable, cette relation permet par exemple de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> bint(A, B).
B ~ real,
A ~ cc(0,1).
>> bint(0, B).
B ~ real.
>> bint(1, B).
B ~ real.
>> bint(A, pi).
A = 0.
>> bint(A, oo(1,2)).
A = 0.
>> oc(X, 1, 2), bint(B,X), boolsplit([B]).
B = 0,
X ~ oo(1,2);
B = 1,
X = 2.
```

En mode intervalle simples, la contrainte «int» ne réduit pas suffisamment les domaines des variables pour permettre de conclure :

```
>> bint(A,I), I ~ int n cc(1,10).
I ~ cc(1,10),
A ~ cc(0,1).
```

Par contre, en mode union d'intervalles, une conclusion peut être obtenue :

```
>> set_prolog_flag(interval_mode,union).
true.
>> bint(A,I), I ~ int n cc(1,10).
A = 1,
I ~ 1 u 2 u 3 u 4 u 5 u 6 u 7 u 8 u 9 u 10.
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : **bint/2** est définie sur l'ensemble $\mathbf{B} \times \mathbf{R}$. Dans le cas où son deuxième argument prend une valeur non numérique, cette contrainte n'affecte pas la valeur 0 à son premier argument, mais échoue, comme on le voit dans l'exemple suivant.

```
>> bint(A,toto).
false.
```

Ceci est bien entendu dû au fait que l'ensemble des entiers ne constituent pas un sous-domaine de Prolog IV, mais seulement une contrainte numérique, qui est donc traitée en tant que telle.

Attention en utilisant la relation **bint/3** en mode union d'intervalles. Si le domaine d'une variable n'est pas restreint, des problèmes de mémoire peuvent survenir. Voir **int/1** pour plus de détails.

Voir également : bnint/2, int/1, nint/1.

ble/3 — Inférieur ou égal

Définition : **ble/3** définit la relation $\{(a, b, c) \in \mathbf{B} \times \mathbf{R}^2 \mid a = 1 \text{ ssi } (b, c) \in \pi_4(\text{le/2})\}$.

Description : **ble/3** est une relation liant une variable booléenne à deux variables numériques, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur soit ou non inférieure ou égale à une autre.

En associant une valeur booléenne à la valeur de vérité d'une comparaison numérique, cette relation permet de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors d'effectuer une partition sur le domaine de la variable.

Exemples :

```
>> ble(A, B, C).
C ~ real,
B ~ real,
A ~ cc(0,1).
>> ble(0, B, C).
C ~ real,
B ~ real.
>> ble(1, B, C).
C ~ real,
B ~ real.
>> ble(1, B, 2).
B ~ le(2).
>> ble(A, cc(1,3), cc(2,4)).
A ~ cc(0,1).
>> ble(A, cc(1,2), cc(3,4)).
A = 1.
>> ble(A, cc(5,6), cc(6,7)).
A = 1.
>> ble(A, cc(6,7), cc(5,6)).
A ~ cc(0,1).
>> A ~ ble(X,2), boolsplit([A]).
A = 0,
X ~ gt(2);
A = 1,
X ~ le(2).
```

Note : Voir **bcc/4** pour de plus amples informations concernant l'utilisation de ce type de contraintes dans des contraintes complexes.

Voir également : bge/3, bgt/3, blt/3, le/2.

bleaf/2 — Etre un arbre d'un nœud

Définition : **bleaf/2** définit la relation $\{(a, b) \in \mathbf{B} \times \mathbf{A} \mid a = 1 \text{ ssi } b \in \pi_4(\text{leaf/1})\}$.

Description : **bleaf/2** est une relation liant une variable booléenne à une variable quelconque, qui est traitée par le solveur général et le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur représente ou non un arbre réduit à un seul nœud.

En associant une valeur booléenne à une information de typage d'une variable, cette relation permet par exemple de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> bleaf(A,B).
B ~ tree,
A ~ cc(0,1).
>> bleaf(0,B).
B ~ nleaf.
>> bleaf(1, B).
B ~ leaf.
```

Bien entendu, aucun arbre infini ne peut être un arbre d'un nœud :

```
>> bleaf(A, infinite).
A = 0.
>> bleaf(A, cc(1,2)).
A = 1.
>> list(L), bleaf(A, L), boolsplit([A]).
A = 0,
L ~ [tree|list];
A = 1,
L = [].
```

Voir également : bnleaf/2, leaf/1, nleaf/1.

blist/2 — Etre une liste

Définition : **blist/2** définit la relation $\{(a, b) \in \mathbf{B} \times \mathbf{A} \mid a = 1 \text{ ssi } b \in \pi_4(\text{list/1})\}$.

Description : **blist/2** est une relation liant une variable booléenne à une variable quelconque, qui est traitée par le solveur général et le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur représente ou non une liste.

En associant une valeur booléenne à une information de typage d'une variable, cette relation permet par exemple de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> blist(A, B).
B ~ tree,
A ~ cc(0,1).
>> blist(0, B).
B ~ nlist.
>> blist(1, B).
B ~ list.
>> leaf(X), blist(A, X).
A ~ cc(0,1),
X ~ leaf.
>> leaf(X), blist(A, X), boolsplit([A]).
A = 0,
X ~ leaf,
X ~ nlist;
A = 1,
X = [].
>> blist(A, [X,Y]).
A = 1,
Y ~ tree,
X ~ tree.
```

Voir également : bnlist/2, list/1, nlist/1.

blt/3 — Inférieur

Définition : **blt/3** définit la relation $\{(a, b, c) \in \mathbf{B} \times \mathbf{R}^2 \mid a = 1 \text{ ssi } (b, c) \in \pi_4(\text{lt/2})\}$.

Description : **blt/3** est une relation liant une variable booléenne à deux variables numériques, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur soit ou non inférieure strictement à une autre.

En associant une valeur booléenne à la valeur de vérité d'une comparaison numérique, cette relation permet de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors d'effectuer une partition sur le domaine de la variable.

Exemples :

```
>> blt(A, B, C).
C ~ real,
B ~ real,
A ~ cc(0,1).
>> blt(0, B, C).
C ~ real,
B ~ real.
>> blt(1, B, C).
C ~ real,
B ~ real.
>> blt(1, B, 2).
B ~ lt(2).
>> blt(A, cc(1,3), cc(2,4)).
A ~ cc(0,1).
>> blt(A, cc(1,2), cc(3,4)).
A = 1.
>> blt(A, cc(5,6), cc(6,7)).
A ~ cc(0,1).
>> blt(A, cc(6,7), cc(5,6)).
A = 0.
>> A ~ blt(X,2), boolsplit([A]).
A = 0,
X ~ ge(2);
A = 1,
X ~ lt(2).
```

Note : Voir **bcc/4** pour de plus amples informations concernant l'utilisation de ce type de contraintes dans des contraintes complexes.

Voir également : bge/3, bgt/3, ble/3, lt/2.

bnidentifier/2 — Ne pas être un identificateur

Définition : **bnidentifier/2** définit la relation $\{(a,b) \in \mathbf{B} \times \mathbf{F} \mid a = 1 \text{ ssi } b \in \pi_4(\text{nidentifier/1})\}$.

Description : **bnidentifier/2** est une relation liant une variable booléenne à une variable quelconque, qui est traitée par le solveur général et le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur représente ou non un identificateur.

En associant une valeur booléenne à une information de typage d'une variable, cette relation permet par exemple de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> bnidentifier(A, B).
B ~ tree,
A ~ cc(0,1).
>> bnidentifier(0, B).
B ~ identifier.
>> bnidentifier(1, B).
B ~ nidentifier.
>> bnidentifier(A, list).
A ~ cc(0,1).
>> list(L), bnidentifier(A,L), boolsplit([A]).
A = 0,
L = [];
A = 1,
L ~ [tree|list].
>> bnidentifier(A, toto).
A = 0.
>> bnidentifier(A, toto(1)).
A = 1.
```

Voir également : bidentifier/2, identifier/1, nidentifier/1.

bnint/2 — Ne pas être un entier

Définition : **bnint/2** définit la relation $\{(a, b) \in \mathbf{B} \times \mathbf{R} \mid a = 1 \text{ ssi } b \in \pi_4(\text{nint/1})\}$.

Description : **bnint/2** est une relation liant une variable booléenne à une variable numérique, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur représente ou non un nombre non-entier.

Exemples :

```
>> bnint(A, B).
B ~ real,
A ~ cc(0,1).
>> bnint(0, B).
B ~ real.
>> bnint(1, B).
B ~ real.
>> bnint(A, pi).
A = 1.
>> bnint(A, oo(1,2)).
A = 1.
>> oc(X, 1, 2), bnint(B,X), boolsplit([B]).
B = 0,
X = 2;
B = 1,
X ~ oo(1,2).
```

Note : Voir **bint/2** pour des remarques importantes sur le fonctionnement des contraintes **bint/2** et **bnint/2**.

Voir également : bint/2, int/1, nint/1.

bnleaf/2 — Etre un arbre de plus d'un nœud

Définition : **bnleaf/2** définit la relation $\{(a, b) \in \mathbf{B} \times \mathbf{A} \mid a = 1 \text{ ssi } b \in \pi_4(\text{nleaf/1})\}$.

Description : **bnleaf/2** est une relation liant une variable booléenne à une variable quelconque, qui est traitée par le solveur général et le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur représente ou non un arbre réduit à un seul nœud.

En associant une valeur booléenne à une information de typage d'une variable, cette relation permet par exemple de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> bnleaf(A,B).
B ~ tree,
A ~ cc(0,1).
>> bnleaf(0, B).
B ~ leaf.
>> bnleaf(1, B).
B ~ nleaf.
>> bnleaf(A, infinite).
A = 1.
>> bnleaf(A, cc(1,2)).
A = 0.
>> list(L), bnleaf(A, L), boolsplit([A]).
A = 0,
L = [];
A = 1,
L ~ [tree|list].
```

Voir également : bleaf/2, leaf/1, nleaf/1.

bnlist/2 — Ne pas être une liste

Définition : **bnlist/2** définit la relation $\{(a, b) \in \mathbf{B} \times \mathbf{A} \mid a = 1 \text{ ssi } b \in \pi_4(\text{list/1})\}$.

Description : **bnlist/2** est une relation liant une variable booléenne à une variable quelconque, qui est traitée par le solveur général et le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur représente ou non une liste.

En associant une valeur booléenne à une information de typage d'une variable, cette relation permet par exemple de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> bnlist(A, B).
B ~ tree,
A ~ cc(0,1).
>> bnlist(0, B).
B ~ list.
>> bnlist(1, B).
B ~ nlist.
>> leaf(X), bnlist(A, X).
A ~ cc(0,1),
X ~ leaf.
>> leaf(X), bnlist(A, X), boolsplit([A]).
A = 0,
X = [];
A = 1,
X ~ leaf,
X ~ nlist.
>> bnlist(A, [X,Y]).
A = 0,
Y ~ tree,
X ~ tree.
```

Voir également : blist/2, list/1, nlist/1.

bnot/2 — Négation

Définition : **bnot/2** définit la relation $\{(a, b) \in \mathbf{B}^2 \mid a = 1 \text{ ssi } b \in \pi_4(\text{not/1})\}$.

bnot/2 est une relation sur les booléens, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre deux booléens, dont le premier est la négation du second.

Cette relation peut être utilisée pour construire des expressions booléennes complexes, ou encore comme connecteur dans la construction d'expressions numériques complexes.

Autre notation : $\mathbf{bnot}(a, b)$ s'écrit aussi $a = \mathbf{bnot}\ b$.

Exemples :

```
>> bnot(A, B).
B ~ cc(0,1),
A ~ cc(0,1).
>> bnot(0, B).
B = 1.
>> bnot(1, B).
B = 0.
>> bnot(A, 0).
A = 1.
>> bnot(A, 1).
A = 0.
```

Voir également : band/3, bequiv/3, bimpl/3, bor/3, bxor/3, not/1.

bnprime/2 — Ne pas être un entier premier

Définition : **bnprime/2** définit la relation $\{(a, b) \in \mathbf{B} \times \mathbf{R} \mid a = 1 \text{ ssi } b \in \pi_4(\text{nprime/1})\}$.

Description : La relation **bnprime/2** n'est pas implantée dans la version actuelle de Prolog IV. Toutefois, étant donné qu'elle fait partie de la définition du langage, le symbole a été réservé.

Exemples :

```
>> bnprime(A, B).
Relation not yet implemented: bnprime
false.
```

Voir également : bint/2, bnint/2, bprime/2, int/1, nint/1, nprime/2, prime/1.

bnreal/2 — Ne pas être un réel

Définition : **bnreal/2** définit la relation $\{(a, b) \in \mathbf{B} \times \mathbf{A} \mid a = 1 \text{ ssi } b \in \pi_4(\text{nreal/1})\}$.

Description : **bnreal/2** est une relation liant une variable booléenne à une variable quelconque, qui est traitée par le solveur général et le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur représente ou non un nombre réel.

En associant une valeur booléenne à une information de typage d'une variable, cette relation permet par exemple de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> bnreal(A,B).
B ~ tree,
A ~ cc(0,1).
>> bnreal(0, B).
B ~ real.
>> bnreal(1, B).
B ~ nreal.
>> bnreal(A, f(X)).
A = 1,
X ~ tree.
>> bnreal(A, cc(1,2)).
A = 0.
```

Voir également : breal/2, nreal/1, real/1.

boc/4 — Appartenance à un intervalle

Définition : **boc/4** définit la relation $\{(a, b, c, d) \in \mathbf{B}\times\mathbf{R}^3 \mid a = 1 \text{ ssi } (b, c, d) \in \pi_4(\text{oc/3})\}$.

Description : **boc/4** est une relation liant une variable booléenne à trois variables numériques, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur appartienne à un intervalle ouvert à gauche et fermé à droite de la forme $(c, d]$.

En associant une valeur booléenne à la valeur de vérité d'une contrainte d'appartenance à un intervalle, cette relation permet soit de construire des expressions numériques complexes, soit de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> boc(A, B, C, D).
D ~ real,
C ~ real,
B ~ real,
A ~ cc(0,1).
>> boc(A, B, 0, 1).
B ~ real,
A ~ cc(0,1).
>> boc(1, B, 0, 2).
B ~ oc(0,2).
>> boc(A, 0, 0, 2).
A = 0.
>> boc(A, 1, 0, 2).
A = 1.
>> boc(A, 2, 0, 2).
A = 1.
>> set_prolog_flag(interval_mode,union).
true.
>> boc(0, B, 0, 2).
B ~ le(0)u gt(2).
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : Voir **bcc/4** pour de plus amples informations concernant l'utilisation des prédicats de vérification d'appartenance à des intervalles.

Voir également : bcc/4, bco/4, boo/4, boutoc/4, oc/3, outco/3, outoc/3.

boo/4 — Appartenance à un intervalle

Définition : **boo/4** définit la relation $\{(a, b, c, d) \in \mathbf{B}\times\mathbf{R}^3 \mid a = 1 \text{ ssi } (b, c, d) \in \pi_4(\text{oo/3})\}$.

Description : **boo/4** est une relation liant une variable booléenne à trois variables numériques, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur appartienne à un intervalle ouvert des deux côtés de la forme (c, d).

En associant une valeur booléenne à la valeur de vérité d'une contrainte d'appartenance à un intervalle, cette relation permet soit de construire des expressions numériques complexes, soit de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> boo(A, B, C, D).
D ~ real,
C ~ real,
B ~ real,
A ~ cc(0,1).
>> boo(A, B, 0, 1).
B ~ real,
A ~ cc(0,1).
>> boo(1, B, 0, 2).
B ~ oo(0,2).
>> boo(A, 0, 0, 2).
A = 0.
>> boo(A, 1, 0, 2).
A = 1.
>> boo(A, 2, 0, 2).
A = 0.
>> set_prolog_flag(interval_mode,union).
true.
>> boo(0, B, 0, 2).
B ~ le(0)u ge(2).
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : Voir **bcc/4** pour de plus amples informations concernant l'utilisation des prédicats de vérification d'appartenance à des intervalles.

Voir également : bcc/4, bco/4, boc/4, boutoo/4, oo/3, outoo/3.

bor/3 — Ou

Définition : **bor/3** définit la relation $\{(a,b,c) \in \mathbf{B}^3 \mid a = 1 \text{ ssi } (b,c) \in \pi_4(\text{or/2})\}$.

Description : **bor/3** est une relation sur les booléens, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre trois booléens, dont le premier est la disjonction des deux autres.

Cette relation peut être utilisée pour construire des expressions booléennes complexes, ou encore comme connecteur dans la construction d'expressions numériques complexes.

Autre notation : **bor**(a,b,c) s'écrit aussi $a = b$ **bor** c.

Exemples :

```
>> bor(A, B, C).
C ~ cc(0,1),
B ~ cc(0,1),
A ~ cc(0,1).
>> bor(0, B, C).
C = 0,
B = 0.
>> bor(1, B, C).
C ~ cc(0,1),
B ~ cc(0,1).
>> bor(1, 0, C).
C = 1.
>> bor(1, 1, C).
C ~ cc(0,1).
>> bor(A, 1, C).
A = 1,
C ~ cc(0,1).
```

Voir également : band/3, bequiv/3, bimpl/3, bnot/2, bxor/3, or/2.

boutcc/4 — Non-appartenance à un intervalle

Définition : **boutcc/4** définit la relation $\{(a,b,c,d) \in \mathbf{B} \times \mathbf{R}^3 \mid a = 1 \text{ ssi } (b,c,d) \in \pi_4(\text{outcc/3})\}$.

Description : **boutcc/4** est une relation liant une variable booléenne à trois variables numériques, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur n'appartienne pas à un intervalle fermé des deux côtés de la forme $[c, d]$.

En associant une valeur booléenne à la valeur de vérité d'une contrainte de non-appartenance à un intervalle, cette relation permet soit de construire des expressions numériques complexes, soit de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> boutcc(A, B, C, D).
D ~ real,
C ~ real,
B ~ real,
A ~ cc(0,1).
>> boutcc(A, B, 0, 1).
B ~ real,
A ~ cc(0,1).
>> boutcc(1, B, 0, 2).
B ~ real.
>> boutcc(0, B, 0, 2).
B ~ cc(0,2).
>> boutcc(A, 0, 0, 2).
A = 0.
>> boutcc(A, 1, 0, 2).
A = 0.
>> boutcc(A, 2, 0, 2).
A = 0.
>> set_prolog_flag(interval_mode,union).
true.
>> boutcc(0, B, 0, 2).
B ~ cc(0,2).
>> boutcc(1, B, 0, 2).
B ~ lt(0)u gt(2).
>> set_prolog_flag(interval_mode,simple).
true.
```

Voir également : bcc/4, boutco/4, boutoc/4, boutoo/4, cc/3, outcc/3.

boutco/4 — Non-appartenance à un intervalle

Définition : **boutco/4** définit la relation $\{(a, b, c, d) \in \mathbf{B} \times \mathbf{R}^2 \mid a = 1 \text{ ssi } (b, c, d) \in \pi_4(\text{outco/3})\}$.

Description : **boutco/4** est une relation liant une variable booléenne à trois variables numériques, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur n'appartienne pas à un intervalle fermé à gauche et ouvert à droite de la forme $[c, d)$.

En associant une valeur booléenne à la valeur de vérité d'une contrainte de non-appartenance à un intervalle, cette relation permet soit de construire des expressions numériques complexes, soit de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> boutco(A, B, C, D).
D ~ real,
C ~ real,
B ~ real,
A ~ cc(0,1).
>> boutco(A, B, 0, 1).
B ~ real,
A ~ cc(0,1).
>> boutco(1, B, 0, 2).
B ~ real.
>> boutco(0, B, 0, 2).
B ~ co(0,2).
>> boutco(A, 0, 0, 2).
A = 0.
>> boutco(A, 1, 0, 2).
A = 0.
>> boutco(A, 2, 0, 2).
A = 1.
>> set_prolog_flag(interval_mode,union).
true.
>> boutco(0, B, 0, 2).
B ~ co(0,2).
>> boutco(1, B, 0, 2).
B ~ lt(0)u ge(2).
>> set_prolog_flag(interval_mode,simple).
true.
```

Voir également : bco/4, boutcc/4, boutoc/4, boutoo/4, co/3, outco/3.

boutlist/3 — Non-appartenance à une liste

Définition : **boutlist/3** définit la relation $\{(a, b, c) \in \mathbf{B} \times \mathbf{A} \times \mathbf{L} \mid a = 1 \text{ ssi } (b, c) \in \pi_4(\text{outlist/2})\}$.

Description : **boutlist/3** est une relation liant un booléen, un arbre quelconque et une liste, qui est traitée par le solveur général. Elle établit une relation entre un booléen et le fait qu'un élément appartienne ou non à une liste.

En associant une valeur booléenne à la valeur de vérité d'une contrainte de non-appartenance à une liste, cette relation permet de construire des contraintes complexes, et d'utiliser les variables booléennes dans des énumérations.

Exemples :

```
>> boutlist(A, B, C).
B ~ tree,
C ~ list,
A ~ cc(0,1).
>> boutlist(1, B, C).
B ~ tree,
C ~ list.
>> boutlist(1, B, [1,2,3]).
B ~ tree.
>> boutlist(0, B, [1,2,3]).
B ~ cc(1,3).
>> boutlist(A, 1, [1,2,3]).
A = 0.
>> boutlist(A, B, []).
A = 1,
B ~ tree.
```

Note : Voir **inlist/2** pour des remarques importantes sur le fonctionnement des contraintes d'appartenance à une liste.

Voir également : binlist/3, inlist/2, outlist/2.

boutoc/4 — Non-appartenance à un intervalle

Définition : **boutoc/4** définit la relation $\{(a, b, c, d) \in \mathbf{B} \times \mathbf{R}^2 \mid a = 1 \text{ ssi } (b, c, d) \in \pi_4(\text{outoc/3})\}$.

Description : **boutoc/4** est une relation liant une variable booléenne à trois variables numériques, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur n'appartienne pas à un intervalle ouvert à gauche et fermé à droite de la forme $(c, d]$.

En associant une valeur booléenne à la valeur de vérité d'une contrainte de non-appartenance à un intervalle, cette relation permet soit de construire des expressions numériques complexes, soit de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> boutoc(A, B, C, D).
D ~ real,
C ~ real,
B ~ real,
A ~ cc(0,1).
>> boutoc(A, B, 0, 1).
B ~ real,
A ~ cc(0,1).
>> boutoc(1, B, 0, 2).
B ~ real.
>> boutoc(0, B, 0, 2).
B ~ oc(0,2).
>> boutoc(A, 0, 0, 2).
A = 1.
>> boutoc(A, 1, 0, 2).
A = 0.
>> boutoc(A, 2, 0, 2).
A = 0.
>> set_prolog_flag(interval_mode,union).
true.
>> boutoc(0, B, 0, 2).
B ~ oc(0,2).
>> boutoc(1, B, 0, 2).
B ~ le(0)u gt(2).
>> set_prolog_flag(interval_mode,simple).
true.
```

Voir également : boc/4, boutcc/4, boutco/4, boutoo/4, oc/3, outoc/3.

boutoo/4 — Non-appartenance à un intervalle

Définition : **boutoo/4** définit la relation $\{(a, b, c, d) \in \mathbf{B} \times \mathbf{R}^2 \mid a = 1 \text{ ssi } (b, c, d) \in \pi_4(\text{outoo/3})\}$.

Description : **boutoo/4** est une relation liant une variable booléenne à trois variables numériques, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur n'appartienne pas à un intervalle ouvert des deux côtés de la forme (c, d).

En associant une valeur booléenne à la valeur de vérité d'une contrainte de non-appartenance à un intervalle, cette relation permet soit de construire des expressions numériques complexes, soit de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> boutoo(A, B, C, D).
D ~ real,
C ~ real,
B ~ real,
A ~ cc(0,1).
>> boutoo(A, B, 0, 1).
B ~ real,
A ~ cc(0,1).
>> boutoo(1, B, 0, 2).
B ~ real.
>> boutoo(0, B, 0, 2).
B ~ oo(0,2).
>> boutoo(A, 0, 0, 2).
A = 1.
>> boutoo(A, 1, 0, 2).
A = 0.
>> boutoo(A, 2, 0, 2).
A = 1.
>> set_prolog_flag(interval_mode,union).
true.
>> boutoo(0, B, 0, 2).
B ~ oo(0,2).
>> boutoo(1, B, 0, 2).
B ~ le(0)u ge(2).
>> set_prolog_flag(interval_mode,simple).
true.
```

Voir également : boo/4, boutcc/4, boutco/4, boutoc/4, oo/3, outoo/3.

bprime/2 — Etre un entier premier

Définition : **bprime/2** définit la relation $\{(a, b) \in \mathbf{B} \times \mathbf{R} \mid a = 1 \text{ ssi } b \in \pi_4(\text{prime/1})\}$.

Description : La relation **bnprime/2** n'est pas implantée dans la version actuelle de Prolog IV. Toutefois, étant donné qu'elle fait partie de la définition du langage, le symbole a été réservé.

Exemples :

```
>> bprime(A, B).
Relation not yet implemented: bprime
false.
```

Voir également : bint/2, bnint/2, bnprime/2, int/1, nint/1, nprime/2, prime/1.

breal/2 — Etre un réel

Définition : **breal/2** définit la relation $\{(a,b) \in \mathbf{B} \times \mathbf{A} \mid a = 1 \text{ ssi } b \in \pi_4(\text{real/1})\}$.

Description : **breal/2** est une relation liant une variable booléenne à une variable quelconque, qui est traitée par le solveur général et le solveur sur les intervalles. Elle établit une relation entre un booléen et le fait qu'une valeur représente ou non un nombre réel.

En associant une valeur booléenne à une information de typage d'une variable, cette relation permet par exemple de construire des contraintes complexes. Une énumération sur la valeur de la variable booléenne permet alors de choisir les contraintes à appliquer.

Exemples :

```
>> breal(A, B).
B ~ tree,
A ~ cc(0,1).
>> breal(0, B).
B ~ nreal.
>> breal(1, B).
B ~ real.
>> breal(A, toto).
A = 0.
>> breal(A, cc(1, 3)).
A = 1.
```

Voir également : bnreal/2, nreal/1, real/1.

bxor/3 — Ou exclusif

Définition : **bxor/3** définit la relation $\{(a,b,c) \in \mathbf{B}^3 \mid a = 1 \text{ ssi } (b,c) \in \pi_4(\text{xor/2})\}$.

Description : **bxor/3** est une relation sur les booléens, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre trois booléens, dont le premier est vrai si le second et le troisième soit différents (c'est-à-dire un ou exclusif).

Cette relation peut être utilisée pour construire des expressions booléennes complexes, ou encore comme connecteur dans la construction d'expressions numériques complexes.

Autre notation : **bxor**(a,b,c) s'écrit aussi $a = b$ **bxor** c.

Exemples :

```
>> bxor(A, B, C).
C ~ cc(0,1),
B ~ cc(0,1),
A ~ cc(0,1).
>> bxor(0, B, C).
B = C,
C ~ cc(0,1).
>> bxor(1, B, C).
C ~ cc(0,1),
B ~ cc(0,1).
>> bxor(A, 0, C).
A = C,
C ~ cc(0,1).
>> bxor(A, 1, C).
C ~ cc(0,1),
A ~ cc(0,1).
>> bxor(1, 0, C).
C = 1.
>> bxor(1, 1, C).
C = 0.
>> eq(B, C), bxor(A, B, C).
B = C,
A ~ cc(0,1),
C ~ cc(0,1).
>> dif(B, C), bxor(A, B, C).
A ~ cc(0,1),
C ~ cc(0,1),
B ~ cc(0,1).
```

Voir également : band/3, bequiv/3, bimpl/3, bnot/2, bor/3, xor/2.

cc/3 — Appartenance à un intervalle

Définition : **cc/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid a \in [b, c]\}$.

Description : **cc/3** est une relation sur les nombres réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un nombre et les bornes d'un intervalle fermé des deux côtés de la forme $[b, c]$ le contenant.

cc/3 est très couramment utilisée en Prolog IV, dans la mesure où cette relation est utilisée pour fixer le domaine d'une variable. Son utilisation la plus courante est donc sous la forme **cc**(A, v_1, v_2) où v_1 et v_2 sont des constantes numériques. Toutefois, cette relation peut également être utilisée pour effectuer des encadrements.

Exemples :

```
>> cc(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> cc(A, 1, 2).
A ~ cc(1,2).
>> cc(A, 0, 0).
A = 0.
```

La requête suivante indique que les intervalles fermés qui contiennent 0 ont une borne inférieure inférieure ou égale à 0 et une borne supérieure supérieure ou égale à 0.

```
>> cc(0, B, C).
C ~ ge(0),
B ~ le(0).
```

La borne supérieure doit être supérieure ou égale à la borne inférieure.

```
>> cc(A, 0, C).
C ~ ge(0),
A ~ ge(0).
>> A ~ cc(0,2), B ~ cc(1,3), C = A .+. B .
C ~ cc(1,5),
B ~ cc(1,3),
A ~ cc(0,2).
```

Voir également : bcc/4, boutcc/4, co/3, oc/3, oc/3, oo/3, outcc/3.

ceil/2 — Plus petit entier supérieur

Définition : **ceil/2** définit la relation $\{(a,b) \in \mathbf{Z} \times \mathbf{R} \mid a = \lceil b \rceil\}$.

Description : **ceil/2** est une relation liant deux variables numériques, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un nombre et le plus petit entier qui lui est supérieur ou égal.

Exemples :

```
>> ceil(A, B).
B ~ real,
A ~ real.
>> ceil(A, 1.5).
A = 2.
>> ceil(A, 1).
A = 1.
>> ceil(A, -1.5) .
A = -1.
>> ceil(2, B).
B ~ oc(1,2).
>> set_prolog_flag(interval_mode,union).
true.
>> ceil(A, cc(1,2)).
A ~ 1 u 2.
>> ceil(A, oo(1,2)).
A = 2.
>> set_prolog_flag(interval_mode,union).
true.
```

Note : Etant donné qu'un des arguments de **ceil/2** est défini comme étant un entier, il est important que le domaine des variables soit restreint à une valeur raisonnable avant d'utiliser cette contrainte en mode union d'intervalles. Sinon, des problèmes de mémoire peuvent se produire. Voir **int/1** pour de plus amples détails.

Voir également : abs/2, floor/2, int/1.

co/3 — Appartenance à un intervalle

Définition : **co/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid a \in [b, c)\}$.

Description : **co/3** est une relation sur les nombres réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un nombre et les bornes d'un intervalle fermé à gauche et ouvert à droite de la forme $[b, c)$ le contenant.

Exemples :

```
>> co(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> co(A, 1, 2).
A ~ co(1,2).
>> co(0, B, C).
C ~ gt(0),
B ~ le(0).
>> co(A, 0, 0).
false.
>> co(A, 0, C).
C ~ gt(0),
A ~ ge(0).
>> A ~ cc(0,2), B ~ co(1,3), C = A .+. B .
C ~ co(1,5),
B ~ co(1,3),
A ~ cc(0,2).
```

Note : Voir cc/3 pour une explication plus précise de certains exemples ci-dessus.

Voir également : bco/4, boutco/4, cc/3, oc/3, oo/3, outco/3.

conc/3
o — Concaténation de listes

Définition : **conc/3** définit la relation $\{(a, b, c) \in \mathbf{L}^3 \mid a = b \bullet c\}$.

Autre notation : **conc**(a, b, c) s'écrit aussi $a = b$ **o** c.

Description : **conc/3** est une relation sur les listes, qui est traitée par le solveur général. Elle établit une relation entre trois listes, dont la première est la concaténation des deux autres.

Exemples :

```
>> conc(A, B, C).
C ~ list,
B ~ list,
A ~ list.
>> A ~ conc([1,2,3],[4,5]).
A = [1,2,3,4,5].
>> [1,X,Y,4,5] ~ [Z,2] o [3|T] .
T = [4,5],
Z = 1,
Y = 3,
X = 2.
>> [1] o X = X o [1], size(X) = 10 .
X = [1,1,1,1,1,1,1,1,1,1].
```

Dans la requête suivante, il est impossible de conclure, car deux des listes impliquées dans la concaténation ont des longueurs inconnues.

```
>> [1|X] = [2,3] o Y .
Y ~ list,
X ~ list.
```

Dès que la longueur d'une deuxième liste est connue, l'échec est détecté.

```
>> [1|X] = [2,3] o Y , size(X) = 4 .
false.
```

Note : Comme les exemples ci-dessus le montrent, les contraintes **conc**(a, b, c) ne peuvent être résolues que quand les longueurs d'au moins deux des listes impliquées sont connues. Il convient donc de prendre des précautions en écrivant et en utilisant des programmes basés sur la concaténation, de manière à s'assurer que suffisamment d'informations sont connues.

Voir également : size/2, index/3.

cos/2 — Cosinus

Définition : **cos/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a = \cos b\}$.

Description : **cos/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique et son cosinus.

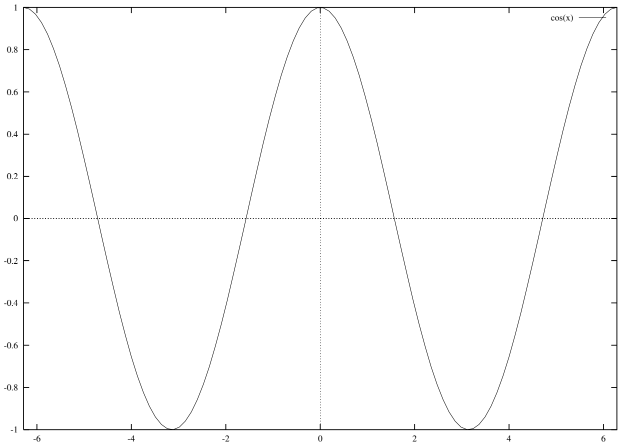
Exemples :

```
>> cos(A, B).
B ~ real,
A ~ cc(-1,1).
>> cos(A, 0).
A = 1.
>> A = 2.*.square(cos(pi./.4)).
A ~ oo(`<0.9999998`,`>1.0000002`).
>> A = 4.*.square(cos(pi./.3)).
A ~ oo(`<0.9999999`,`>1.0000005`).
>> cos(A, pi./.2).
A ~ oo(-`>4.371139e-8`,`>7.54979e-8`).
>> cos(A, A).
A ~ oo(`>0.739085`,`>0.7390852`).
```

Note : La fonction cosinus est périodique. Pour toute valeur possible $v \in [0, 1]$, il existe donc une infinité de $x \in \mathbf{R}$ tels que $\cos x = v$. Ceci pose un problème en mode unions d'intervalles, puisque certaines équations peuvent amener le système à créer une union d'intervalles trop importante pour tenir en mémoire. Il faut donc faire très attention en utilisant des relations représentant des fonctions périodiques, et il est nécessaire de borner les variables utilisées. Dans les exemples ci-dessous, on voit que l'ordre dans lequel les contraintes sont placées est très important dans les requêtes :

```
>> set_prolog_flag(interval_mode,union).
true.
>> cos(X) ~ 0.5, X ~ cc(0, 2 .*. pi).
error: heap_overflow
>> X ~ cc(0, 2 .*. pi), cos(X) ~ 0.5 .
X ~ cc('>1.0471974','>1.0471975')u oo('>5.235987','>5.235988').
>> set_prolog_flag(interval_mode,simple).
true.
```

Graphe :



Voir également : arccos/2, arcsin/2, arctan/2, cot/2, pi/2, sin/2, tan/2.

cosh/2 — Cosinus hyperbolique

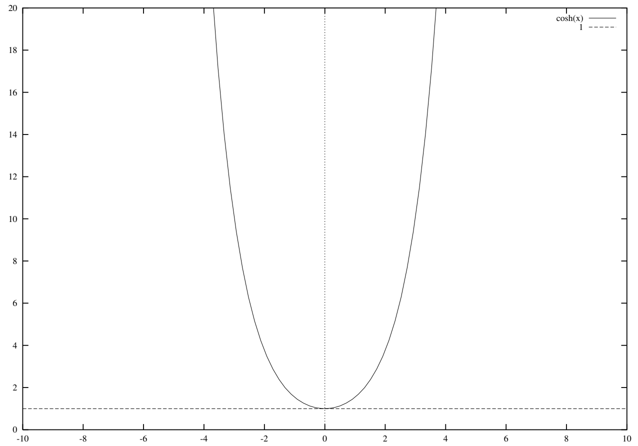
Définition : **cosh/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a = \cosh b\}$.

Description : **cosh/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique et son cosinus hyperbolique.

Exemples :

```
>> cosh(A, B).
B ~ real,
A ~ ge(1).
>> cosh(1, B).
B = 0.
>> cosh(A, 0).
A = 1.
>> cosh(A, 1).
A ~ cc('>1.5430805','>1.5430806').
>> cosh(A, 10).
A ~ cc('>11013.232','>11013.233').
>> cosh(A, 100).
A ~ ge('<4e38').
>> cosh(A, A).
A ~ ge('<4e38').
```

Graphe :



Voir également : coth/2, sinh/2, tanh/2.

cot/2 — Cotangente

Définition : **cot/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid b \bmod \pi \neq 0 \text{ et } a = \cot b\}$.

Description : cot/2 est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique et sa cotangente.

Exemples :

```
>> cot(A, B).
B ~ real,
A ~ real.
>> cot(0, B).
B ~ real.
>> cot(1, B).
B ~ real.
>> cot(A, 0).
false.
>> cot(A, pi./.4).
A ~ oo(`<0.9999995`,`>1.0000004`).
>> cot(A, pi./.2).
A ~ oo(-`>3.2584136e-7`,`<3.894144e-7`).
```

Il existe une infinité de solutions à l'équation $a = \cot a$:

```
>> cot(A, A).
A ~ real.
```

En restreignant l'espace de recherche entre 0 et 2π, on obtient une réduction, mais pas une solution (en effet, il reste deux solutions dans cet intervalle) :

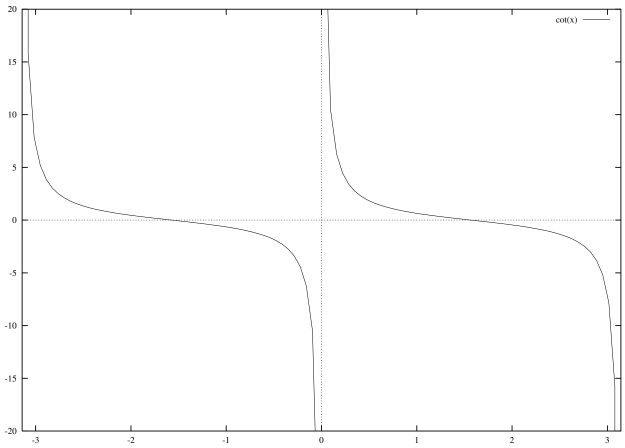
```
>> cot(A,A), A ~ oo(0,2.*.pi).
A ~ oo(`>0.21881127`,`>4.4969735`).
```

En restreignant l'espace de recherche entre 0 et π, on obtient une solution :

```
>> cot(A,A), A ~ oo(0, pi).
A ~ oo(`<0.8603334`,`>0.8603338`).
```

Note : Voir **cos/2** pour des remarques importantes concernant l'utilisation de relations associées à des fonctions périodiques en mode unions d'intervalles.

Graphe :



Voir également : arccos/2, arcsin/2, arctan/2, cos/2, pi/2, sin/2, tan/2.

coth/2 — Cotangente hyperbolique

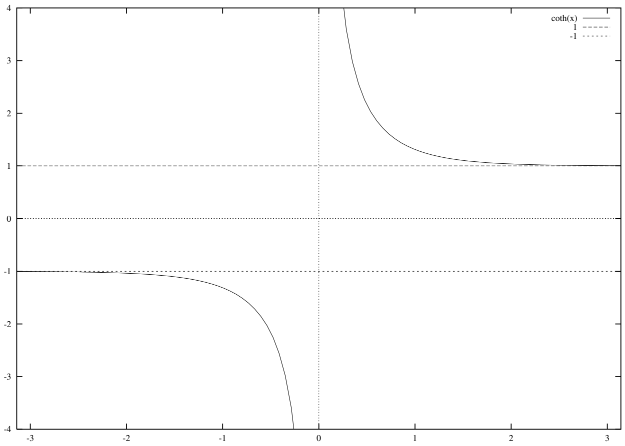
Définition : **coth/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid b \neq 0 \text{ et } a = \coth b\}$.

Description : **coth/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique et sa cotangente hyperbolique.

Exemples :

```
>> coth(A, B).
B ~ real,
A ~ real.
>> ge(A, 0), coth(A, B).
B ~ gt(0),
A ~ gt(1).
>> le(A, 0), coth(A, B).
B ~ lt(0),
A ~ lt(-1).
>> coth(A, 1).
A ~ cc(`>1.3130352`,`>1.3130353`).
>> coth(A, 10).
A ~ oc(1,`>1.0000001`).
>> coth(A, A).
A ~ real.
```

Graphe :



Voir également : cosh/2, sinh/2, tanh/2.

dif/2 — Inégalité

Définition : **dif/2** définit la relation $\{(a, b) \in \mathbf{A}^2 \mid a \neq b\}$.

Description : **dif/2** est une relation générale entre des arbres quelconques. Elle établit une relation entre deux valeurs différentes. Une contrainte construite avec **dif/2** est souvent appelée une diséquation.

Exemples :

```
>> dif(A, B).
B ~ tree,
A ~ tree.
>> dif(alain, michel).
true.
>> dif(alain, alain).
false.
>> dif(f(X), f(Y)).
Y ~ tree,
X ~ tree.
>> dif(f(X+Y), f(X-Y)).
Y ~ real,
X ~ real.
>> dif(f(X+Y), f(X-Y)), Y = 0 .
false.
```

En mode «intervalles simples», la seule réduction possible d'un intervalle est l'exclusion d'une borne :

```
>> X ~ cc(1,2), dif(X,2).
X ~ co(1,2).
>> set_prolog_flag(interval_mode,union).
true.
>> dif(X,2).
X ~ tree.
```

En mode «union d'intervalles», des «trous» sont créés dans un intervalle :

```
>> X ~ cc(1,3), dif(X, 2).
X ~ co(1,2)u oc(2,3).
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : La relation **dif/2** ayant un lien fort avec l'égalité, elle bénéficie en Prolog IV d'un traitement spécifique, qui permet de garantir son fonctionnement complet par rapport aux systèmes de contraintes constitués uniquement d'équations et de diséquations sur les arbres et de contraintes linéaires.

Voir également : eq/2, bdif/3, outlist/2.

div/3
./. Division

Définition : **div/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid c \neq 0 \text{ et } a = b/c\}$.

Description : **div/3** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre deux nombres et leur division.

Autre notation : **div**(a, b, c) s'écrit aussi $a = b$ **./.** c.

Exemples :

```
>> div(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> div(A, 1, 3) .
A = 1/3.
>> A = div(cc(1,2), cc(1,2)).
A ~ cc(0.5,2).
>> X ~ cc(0,1), A = cc(1,2) ./. X .
A ~ ge(1),
X ~ oc(0,1).
>> 1 ~ B ./. C .
B = C,
C ~ real.
```

Quand le domaine de la variable est l'ensemble des réels, la contrainte suivante ne réduit pas le domaine, et n'est donc pas résolue (nous avons en fait deux contraintes, dont une contrainte d'égalité) :

```
>> X = X ./. 2 .
X ~ real.
```

Par contre, si le domaine de la variable est plus petit, la réduction s'effectue correctement. On n'obtient toutefois pas un résultat exact, dans la mesure où l'équation est résolue par approximation et non par pur raisonnement :

```
>> X ~ cc(0,10), X = X ./. 2 .
X ~ cc(0,'>1e-45').
>> X ~ cc(-1,1), A = cc(1,2) ./. X .
A ~ real,
X ~ cc(-1,1).
```

En mode «unions d'intervalle», Les résultats sont parfois beaucoup plus précis ; on remarque entre autres l'exclusion de 0 pour le dénominateur :

```
>> set_prolog_flag(interval_mode, union).
true.
>> X ~ cc(-1,1), A = cc(1,2) ./. X .
A ~ le(-1)u ge(1),
X ~ co(-1,0)u oc(0,1).
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : Quand deux des arguments de **div/3** sont réduits à des singletons, le troisième est calculé de manière exacte. Nous avons donc :

```
>> div(A, 1, 3 ) .
A = 1/3.
```

Il y a bien entendu une exception notable à cette règle :

```
>> div(0,0,A).
A ~ real.
```

Voir également : divlin/3, minus/3, plus/3, times/3, uminus/2, uplus/2.

divlin/3

/ Division

Définition : **divlin/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid c \neq 0 \text{ et } a = b/c\}$.

Description : **divlin/3** est une relation sur les réels, qui est traitée par le solveur linéaire. Elle établit une relation entre deux nombres et leur division.

Autre notation : **divlin**(a, b, c) s'écrit aussi $a = b$ **/** c.

Exemples :

```
>> divlin(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> A ~ divlin(1, 3).
A = 1/3.
>> divlin(1, B, C).
B = C,
C ~ real.
>> divlin(A, B, 1).
A = B,
B ~ real.
>> X = X/2 .
X = 0.
```

Note : La division « linéaire » est retardée jusqu'à ce qu'elle puisse être transformée en une contrainte linéaire. La requête suivante ne fixe donc pas la valeur de la variable `A` :

```
>> A ~ B / B.
B ~ real,
A ~ real.
```

Pour plus de détails sur le retardement de contraintes dans le solveur linéaire, voir **timeslin/3**.

Voir également : div/3, minuslin/3, pluslin/3, timeslin/3, uminuslin/2, upluslin/2.

eq/2

= Egalité

Définition : **eq/2** définit la relation $\{(a, b) \in \mathbf{A}^2 \mid a = b\}$.

Description : **eq/2** est une relation générale sur les arbres, qui est traitée par le solveur général. Elle établit une relation entre deux nombres égaux.

La relation **eq/2** est à la base de Prolog IV, et est utilisée de manière implicite lors de l'unification de deux termes, et en particulier lors de l'exécution de programmes.

Autre notation : **eq**(a, b) s'écrit aussi $a =\ b$.

Exemples :

```
>> eq(A, B).
A = B,
B ~ tree.
>> eq(alain, michel).
false.
>> eq(alain, alain).
true.
>> eq(f(X),f(Y)).
X = Y,
Y ~ tree.
>> eq(f(X+Y),f(X-Y)).
Y = 0,
X ~ real.
>> X ~ cc(1,2), Y ~ cc(2,3), eq(X,Y).
Y = 2,
X = 2.
```

Note : La relation **eq/2** est implantée de manière complète dans le solveur linéaire. Elle peut donc être résolue beaucoup plus efficacement dans ce solveur que dans le solveur sur les intervalles, comme le montrent les exemples ci-dessous :

```
>> eq(1+X, X).
false.
>> eq(1.+. X, X).
X ~ real.
```

Bien entendu, l'égalité sur les arbres est également traitée de manière complète, comme dans tout système Prolog.

```
>> [X,Y,Z,T] = [T,Z,X,Y].
Z = T,
Y = T,
X = T,
T ~ tree.
```

Voir également : beq/3, dif/2.

equiv/2 — Equivalence

Définition : **equiv/2** définit la relation $\{(a, b) \in \mathbf{B}^2 \mid a = b\}$.

Description : **equiv/2** est une relation sur les booléens, qui est traitée par le solveur sur les intervalles. Une contrainte **equiv**(a, b) établit une relation entre deux booléens identiques a et b.

Cette contrainte est normalement utilisée pour construire une combinaison d'autres contraintes exprimées par des pseudo-opérations produisant des booléens.

Autre notation : **equiv**(a, b) s'écrit aussi a **equiv** b.

Exemples :

```
>> equiv(A, B).
A = B,
B ~ cc(0,1).
>> equiv(1, B).
B = 1.
>> equiv(A, 0).
A = 0.
```

Voir également : and/2, bequiv/3, impl/2, not/1, or/2, xor/2.

exp/2 — Exponentielle

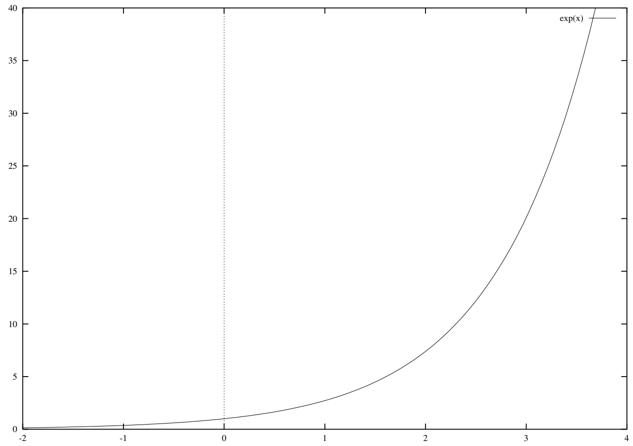
Définition : **exp/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a = \mathrm{e}^b\}$.

Description : **exp/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique x et la valeur e^x (où $e = 2.718281828459\ldots$).

Exemples :

```
>> exp(A, B).
B ~ real,
A ~ gt(0).
>> exp(1, B).
B = 0.
>> A ~ exp(-1).
A ~ cc('>0.3678794','>0.36787945').
>> exp(A, 0).
A = 1.
>> A ~ exp(1).
A ~ cc('>2.7182817','<2.718282').
>> exp(A, 2).
A ~ cc('<7.389056','>7.389056').
>> exp(A, 10).
A ~ cc('>22026.464','>22026.466').
>> A ~ exp(100).
A ~ ge('<4e38').
>> exp(A, A).
A ~ gt('<4e38').
>> 100 ~ exp(C).
C ~ cc('<4.60517','>4.60517').
```

Graphe :



Voir également : ln/2, log/2, power/3, root/3, square/2, sqrt/2.

finite/1 — Etre fini

Définition : **finite/1** définit la relation $\{a \in \mathbf{A} \mid a \text{ est fini}\}$.

Description : **finite/1** est une relation générale sur les arbres, qui est traitée par le solveur général. Une contrainte **finite**(a) est vérifiée quand son argument a représente un arbre fini.

finite/1 est une relation, et pas une primitive de vérification. Si une variable `X` est contrainte par `finite(X)`, elle ne peut plus s'unifier avec un arbre infini.

Exemples :

```
>> finite(A).
A ~ finite.
>> X = f(X), finite(X).
false.
>> finite([X|L]).
L ~ finite,
X ~ finite.
```

Voir également : bfinite/2, binfinite/2, infinite/1.

floor/2 — Plus grand entier inférieur

Définition : **floor/2** définit la relation $\{(a, b) \in \mathbf{Z} \times \mathbf{R} \mid a = \lfloor b \rfloor\}$.

Description : **floor/2** est une relation liant deux variables numériques, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un nombre et le plus grand entier qui lui est inférieur ou égal.

Exemples :

```
>> floor(A, B).
B ~ real,
A ~ real.
>> floor(A, 1.5).
A = 1.
>> floor(A, 1).
A = 1.
>> floor(A, -1.5).
A = -2.
>> floor(2, B).
B ~ co(2,3).
>> set_prolog_flag(interval_mode,union).
true.
>> floor(A, cc(1,2)).
A ~ 1 u 2.
>> floor(A, oo(1,2)).
A = 1.
>> set_prolog_flag(interval_mode,union).
true.
```

Note : Etant donné qu'un des arguments de **floor/2** est défini comme étant un entier, il est important que le domaine de cet argument soit restreint à une valeur raisonnable avant d'utiliser cette contrainte en mode union d'intervalles. Sinon, des problèmes de mémoire peuvent se produire. Voir **int/1** pour de plus amples détails.

Voir également : abs/2, ceil/2, int/1.

gcd/3 — Plus grand diviseur commun

Définition : **gcd/3** définit la relation $\{(a, b, c) \in \mathbf{Z}^3 \mid a \geq 0, b \geq 0, c \geq 0 \text{ et } a = \text{pgcd}(b, c)\}$.

Description : La relation **gcd/3** n'est pas implantée dans la version actuelle de Prolog IV. Toutefois, étant donné qu'elle fait partie de la définition du langage, le symbole a été réservé.

Exemples :

```
>> gcd(A, B, C).
Relation not yet implemented: gcd
false.
```

Voir également : intdiv/3, lcm/3.

ge/2 — Supérieur ou égal

Définition : **ge/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a \geq b\}$.

Description : **ge/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre deux valeurs numériques telles que la première est supérieure ou égale à la seconde.

Exemples :

```
>> ge(A, B).
B ~ real,
A ~ real.
>> ge(A, 0).
A ~ ge(0).
>> A ~ cc(0,4), B ~ cc(1,3), ge(A, B).
B ~ cc(1,3),
A ~ cc(1,4).
```

Note : La relation **ge/2** apparaît souvent dans les réponses de Prolog IV, dans la mesure où un pseudo-terme **ge**(v) désigne en fait l'intervalle $[v, +\infty)$.

Voir également : bge/3, gelin/2, gt/2, le/2, lt/2.

gelin/2 — Supérieur ou égal

Définition : **gelin/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a \geq b\}$.

Description : **gelin/2** est une relation sur les réels, qui est traitée par le solveur linéaire. Elle établit une relation entre deux valeurs numériques telles que la première est supérieure ou égale à la seconde.

Exemples :

```
>> gelin(A, B).
B ~ real,
A ~ real.
>> gelin(A,B), gelin(B,A).
B ~ real,
A ~ real.
>> gelin(A, B), gelin(B, A), dif(A, B).
false.
```

Voir également : ge/2, gtlin/2, lelin/2, ltlin/2.

gt/2 — Supérieur

Définition : **gt/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a > b\}$.

Description : **gt/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre deux valeurs numériques telles que la première est strictement supérieure à la seconde.

Exemples :

```
>> gt(A, B).
B ~ real,
A ~ real.
>> gt(A, 0).
A ~ gt(0).
>> A ~ cc(0,2), B ~ cc(1,3), gt(A,B).
B ~ co(1,2),
A ~ oc(1,2).
>> gt(A,B), gt(B,A).
B ~ real,
A ~ real.
```

Note : La relation **gt/2** apparaît souvent dans les réponses de Prolog IV, dans la mesure où un pseudo-terme **gt**(v) désigne en fait l'intervalle $(v, +\infty)$.

Voir également : bgt/3, ge/2, gtlin/2, le/2, lt/2.

gtlin/2 — Supérieur

Définition : **gtlin/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a > b\}$.

Description : **gtlin/2** est une relation sur les réels, qui est traitée par le solveur linéaire. Elle établit une relation entre deux valeurs numériques telles que la première est strictement supérieure à la seconde.

Exemples :

```
>> gtlin(A, B).
B ~ real,
A ~ real.
>> gtlin(A, B), gtlin(B, A).
false.
```

Voir également : gelin/2, gt/2, lelin/2, ltlin/2.

identifier/1 — Etre un identificateur

Définition : **identifier/1** définit la relation $\{a \in \mathbf{A} \mid a \text{ est un identificateur}\}$.

Description : **identifier/1** est une relation générale sur les arbres, qui est traitée par le solveur général. Une contrainte **identifier**(a) est vérifiée quand son argument a représente un identificateur.

identifier/1 est une relation, et pas une primitive de vérification, comme **atom/1**. Une contrainte `identifier(X)` peut donc être appliquée à une variable `X` dont la valeur est inconnue ; elle ne pourra alors plus s'unifier avec un arbre qui n'est pas un identificateur.

Exemples :

```
>> identifier(A).
A ~ identifier.
>> identifier(toto).
true.
>> identifier(cc(1,3)).
false.
>> X ~ list, identifier(X).
X = [].
```

Voir également : bidentifier/2, bidentifier/2, nidentifier/1.

if/4 — Si alors sinon

Définition : **if/4** définit la relation $\{(a, b, c, d) \in \mathbf{A} \times \mathbf{B} \times \mathbf{A}^2 \mid \text{si } b = 1 \text{ alors } a = c \text{ sinon } a = d\}$.

Description : if/4 est une relation liant un booléen et trois arbres quelconques. Quand le booléen est vrai (1), le premier arbre est égal au second ; sinon il est égal au troisième.

Cette contrainte est normalement utilisée pour construire des contraintes complexes dans lesquelles la vérification d'une contrainte conditionne la pose d'autres contraintes.

Exemples :

```
>> if(A, B, C, D).
D ~ tree,
C ~ tree,
A ~ tree,
B ~ cc(0,1).
>> if(A, B, 1, 2).
B ~ cc(0,1),
A ~ cc(1,2).
>> A = if(B, 1, 2), boolsplit([B]).
B = 0,
A = 2;
B = 1,
A = 1.
>> A ~ cc(1,3), A = if(B, cc(1,2), cc(4,5)).
B = 1,
A ~ cc(1,2).
```

Note : Attention, `if` est une relation et non une structure de contrôle. En particulier, dans la requête suivante, le pseudo-terme `ntree` est transformé en contrainte et évalué, ce qui provoque l'échec de la requête.

```
>> A ~ if(1,tree,ntree).
false.
```

Cette requête est en fait équivalente à la requête suivante :

```
>> if(A,1,C,D), tree(C), ntree(D).
false.
```

On voit bien qu'une contrainte `ntree(D)` apparaît explicitement, ce qui provoque l'échec. En pratique, ce cas de figure se rencontre dans les requêtes du type :

```
>> A ~ if(bge(X,0), ln(X), ln(uminus(X))).
false.
```

Ici, le problème vient du fait que la contrainte implicite `ln(V,X)` contraint `X` à être strictement positif, alors que l'autre contrainte implicite le contraint à être négatif, ce qui déclenche l'échec.

impl/2 — Implication

Définition : **impl/2** définit la relation $\{(a, b) \in \mathbf{B}^2 \mid \text{ si } a = 1 \text{ alors } b = 1\}$.

Description : **impl/2** est une relation sur les booléens, qui est traitée par le solveur sur les intervalles. Une contrainte **impl**(a, b) réussit si son deuxième argument b est impliqué par son premier a.

Cette contrainte est normalement utilisée pour construire une combinaison d'autres contraintes exprimées par des pseudo-opérations produisant des booléens.

Autre notation : **impl**(a, b) s'écrit aussi a **impl** b.

Exemples :

```
>> impl(A, B).
B ~ cc(0,1),
A ~ cc(0,1).
>> impl(0, B).
B ~ cc(0,1).
>> impl(1, B).
B = 1.
>> impl(A, 0).
A = 0.
>> impl(A, 1).
A ~ cc(0,1).
```

Voir également : and/2, bimpl/3, equiv/2, not/1, or/2, xor/2.

index/3

: Nième élément d'une liste

Définition : **index/3** définit la relation $\{(a, b, c) \in \mathbf{A} \times \mathbf{L} \times \mathbf{Z} \mid 1 \leq c \leq |b|$ et a est le $c^{\text{ième}}$ élément de $b\}$.

Description : **index/3** est une relation d'accès à une liste, qui est traitée par le solveur général. Une contrainte **index**(a, b, c) réussit si son troisième argument c est l'index de son premier argument a dans le deuxième b.

Autre notation : **index**(a, b, c) s'écrit aussi $a = b\,\mathbf{:}\,c$.

Exemples :

```
>> index(A, B, C).
A ~ tree,
B ~ list,
C ~ ge(1).
>> index(A, [3,7,5,2], cc(2,3)).
A ~ cc(5,7).
>> index(A, [3,7,5,2], C).
C ~ cc(1,4),
A ~ cc(2,7).
>> A ~ [3,7,5,2]:2.
A = 7.
>> index(2, [3,7,5,2], X).
X = 4.
```

Dans la requête suivante, la pauvreté du résultat peut surprendre. Cela provient du fait que la variable `X` n'est pas typée comme variable numérique, et donc que la liste n'est pas exclusivement numérique. Or, en raison de la définition des sous-domaines d'approximation de Prolog IV, les algorithmes numériques ne sont déclenchés que sur les listes purement numériques :

```
>> index(2, [3,7,5,X], Y).
X ~ tree,
Y ~ cc(1,4).
```

Ici, on a simplement ajouté un type à la variable `X` :

```
>> index(2, [3,7,5,X], Y), X ~ real .
Y = 4,
X = 2.
>> set_prolog_flag(interval_mode,union).
true.
>> index(A, [3,7,5,2], C).
C ~ 1 u 2 u 3 u 4,
A ~ 2 u 3 u 5 u 7.
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : Dans les exemples ci-dessus, nous avons constaté que les réductions effectuées par le traitement des contraintes construites avec `index/3` ne sont pas toujours simples à comprendre. Voici donc quelques remarques qui peuvent aider (on suppose ici qu'on a une contrainte `index(X,L,I)` :

– Pour que des réductions soient effectuées, il faut au moins que `I` soit connu ou que la longueur de `L` soit connue.

– Pour que des réductions numériques soient effectuées sur `X` et sur les éléments de `L`, il faut que tous les éléments de `L` soient contraints à être des nombres réels.

– Pour trouver l'index d'un élément non-numérique d'une liste, il faut que tous les éléments de la liste soient connus.

En fait, toutes ces remarques proviennent de la définition des sous-domaines privilégiés de Prolog IV.

Note : Etant donné qu'un des arguments de **index/3** est défini comme étant un entier, il est important que le domaine de cet argument soit restreint à une valeur raisonnable avant d'utiliser cette contrainte en mode union d'intervalles. Sinon, des problèmes de mémoire peuvent se produire. Voir **int/1** pour de plus amples détails.

Voir également : conc/3, size/2.

infinite/1 — Etre infini

Définition : **infinite/1** définit la relation $\{a \in \mathbf{A} \mid a \text{ est infini}\}$.

Description : **infinite/1** est une relation générale sur les arbres. Une contrainte **infinite**(a) réussit si son argument a est un arbre infini.

infinite/1 est une relation, et pas une primitive de vérification. Si une variable `X` est contrainte par `infinite(X)`, elle ne peut plus s'unifier avec un arbre fini.

Exemples :

```
>> infinite(A).
A ~ infinite.
>> eq(X, f(X)), infinite(X).
X = f(X).
>> X ~ leaf n infinite.
false.
```

Voir également : bfinite/2, binfinite/2, finite/1.

inlist/2 — Appartenance à une liste

Définition : **inlist/2** définit la relation $\{(a, b) \in \mathbf{A} \times \mathbf{L} \mid a \text{ figure dans } b\}$.

Description : **inlist/2** est une relation sur les listes, qui est traitée par le solveur général. Une contrainte **inlist**(a, b) réussit si son premier argument a est un élément de son second argument b.

Exemples :

```
>> inlist(A, B).
A ~ tree,
B ~ list.
>> inlist(B, [1,5,3]).
B ~ cc(1,5).
>> X ~ cc(1,5), Y ~ cc(2,7), X ~ inlist([3,Y]).
Y ~ cc(2,7),
X ~ cc(2,5).
```

La requête suivante diffère de la précédente par le fait que 13 est disjoint du le domaine de `X`. Le système déduit alors que `X` doit être égale à l'unique autre élément de la liste, soit `Y` :

```
>> X ~ cc(1,5), Y ~ cc(2,7), X ~ inlist([13,Y]).
X = Y,
Y ~ cc(2,5).
>> inlist(B, []).
false.
>> set_prolog_flag(interval_mode,union).
true.
>> inlist(B, [1,5,3]).
B ~ 1 u 3 u 5.
>> set_prolog_flag(interval_mode,simple).
```

Note : Les réductions effectuées par **inlist/2** sont parfois assez complexes. Voici quelques remarques à ce sujet (en considérant que nous avons la contrainte `inlist(X,L)`) :

- Tant que la longueur de `L` n'est pas connue, aucune réduction n'est effectuée sur `X`, puisque tout arbre peut apparaître dans la partie inconnue de la liste.
- Pour que des réductions soient effectuées sur `X`, il faut également que tous les éléments de la liste soient contraints à être des nombres réels, soit que tous les éléments de la liste soient connus.
- Pour que des réductions soient effectuées sur un élément de `L`, il faut que tous les éléments de `L` soient contraints à être des nombres réels et que les domaines de tous les éléments de `L` sauf un (celui qui va être réduit) soient disjoints du domaine de `X`.

Les raisonnements sont similaires quand on travaille sur outlist/2, binlist/2 ou boutlist/2. Les mêmes conditions que ci-dessus doivent être respectées pour pouvoir faire des déductions, ce qui peut paraître plus gênant, comme le montrent les exemples ci-dessous :

```
>> A ~ binlist(1,[1,X]).
X ~ tree,
A ~ cc(0,1).
>> A ~ binlist(1,[1,real]).
A = 1.
>> outlist(toto,[toto,identifier]).
true.
>> outlist(toto,[toto,titi]).
false.
```

Voir également : binlist/3, boutlist/3, outlist/2.

int/1 — Etre un entier

Définition : **int/1** définit la relation $\{a \in \mathbf{R} \mid a \text{ est entier}\}$.

Description : **int/1** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. **int/1** réussit quand son argument est un entier.

int/1 est une relation, et pas une primitive de vérification comme **integer/1**. Si une variable `X` est contrainte par `int(X)`, elle ne peut plus s'unifier avec un nombre non entier. Cette information est bien entendu par le solveur sur les intervalles pour effectuer des réductions de domaines.

Exemples :

```
>> int(A).
A ~ real.
>> int(1).
true.
>> A ~ cc(1,2), int(A).
A ~ cc(1,2).
>> A ~ cc(1/2,2), int(A).
A ~ cc(1,2).
>> A ~ oo(1,3), int(A).
A = 2.
>> set_prolog_flag(interval_mode,union).
true.
>> A ~ cc(1,2), int(A).
A ~ 1 u 2.
>> A ~ cc(1,10), int(A ./. 2).
A ~ 2 u 4 u 6 u 8 u 10.
```

La requête ci-dessous est équivalente à la précédente ; seule son expression change. La seconde contrainte exprime le fait qu'il doit exister un entier doit `A` soit le double :

```
>> A ~ cc(1,10), A ~ 2 .*. int .
A ~ 2 u 4 u 6 u 8 u 10.
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : Si la relation **int/1** est appliquée à une variable dont le domaine n'a pas été réduit, une erreur en résultera probablement, à moins que votre machine ne dispose d'une très importante quantité de mémoire. Cela vient du fait que le système essaie de représenter en mémoire la liste de tous les entiers représentables par des rationnels IEEE :

```
>> set_prolog_flag(interval_mode,union).
true.
>> int(A).
error: heap_overflow
>> set_prolog_flag(interval_mode,simple).
true.
```

Quand on combine plusieurs contraintes, il faut donc faire attention à leur ordre, comme on le voit ci-dessous :

```
>> int(A), A ~ cc(1,10).
error: heap_overflow
>> A ~ cc(1,10), int(A).
A ~ 1 u 2 u 3 u 4 u 5 u 6 u 7 u 8 u 9 u 10.
```

Toutefois, en utilisant des pseudo-expressions, Prolog IV se charge de réordonner les contraintes de manière à éviter que des problèmes se produisent, si c'est possible.

```
>> A ~ cc(1,10) n int .
A ~ 1 u 2 u 3 u 4 u 5 u 6 u 7 u 8 u 9 u 10.
>> A ~ int n cc(1,10).
A ~ 1 u 2 u 3 u 4 u 5 u 6 u 7 u 8 u 9 u 10.
```

Des problèmes similaires peuvent se poser avec toutes les relations qui ont à voir avec des entiers, à savoir **bint/2**, **bnint/2**, **ceil/2**, **floor/2**, **index/3** et **nint/2**.

Note : L'interprétation de **int/1**, soit l'ensemble des entiers, n'est pas un sous-domaine privilégié de Prolog IV. En conséquence, les contraintes construites avec **int/1** ne sont pas reportées lors de l'écriture des solutions.

Voir également : bint/2, bnint/2, nint/1.

intdiv/3 — Division entière

Définition : **intdiv/3** définit la relation $\{(a, b, c) \in \mathbf{Z}^3 \mid$ il existe $d \in \mathbf{Z}$ tel que $b = ac + d$ et $c > d \geq 0\}$.

Description : La relation **intdiv/3** n'est pas implantée dans la version actuelle de Prolog IV. Toutefois, étant donné qu'elle fait partie de la définition du langage, le symbole a été réservé.

Exemples :

```
>> intdiv(A, B, C).
Relation not yet implemented: intdiv
false.
```

Voir également : gcd/3, lcm/3.

lcm/3 — Plus petit commun multiple

Définition : **lcm/3** définit la relation $\{(a, b, c) \in \mathbf{Z}^3 \mid a \geq 0, b \geq 0, c \geq 0$ et $a = \text{ppcm}(b, c)\}$.

Description : La relation **lcm/3** n'est pas implantée dans la version actuelle de Prolog IV. Toutefois, étant donné qu'elle fait partie de la définition du langage, le symbole a été réservé.

Exemples :

```
>> lcm(A, B, C).
Relation not yet implemented: lcm
false.
```

Voir également : gcd/3, intdiv/3.

le/2 — Inférieur ou égal

Définition : **le/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a \leq b\}$.

Description : **le/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre deux valeurs numériques telles que la première est inférieure ou égale à la seconde.

Exemples :

```
>> le(A, B).
B ~ real,
A ~ real.
>> le(A, 0).
A ~ le(0).
>> A ~ cc(0,4), B ~ cc(1,3), le(A,B).
B ~ cc(1,3),
A ~ cc(0,3).
>> A ~ cc(0,4), B ~ cc(1,3), le(B,A).
B ~ cc(1,3),
A ~ cc(1,4).
```

Note : La relation **le/2** apparaît souvent dans les réponses de Prolog IV, dans la mesure où un pseudo-terme **le**(v) désigne en fait l'intervalle $(-\infty, v]$.

Voir également : ble/3, ge/2, gt/2, lelin/2, lt/2.

leaf/1 — Etre un arbre d'un nœud

Définition : **leaf/1** définit la relation $\{a \in \mathbf{A} \mid a$ est un arbre d'un nœud$\}$.

Description : **leaf/1** est une relation générale sur les arbres, qui est traitée par le solveur général. Une contrainte **leaf**(a) est vérifiée quand son argument a représente un arbre d'un seul nœud.

leaf/1 est une relation, et pas une primitive de vérification comme **atomic/1**. Si une variable `X` est contrainte par `leaf(X)`, elle ne peut plus s'unifier avec un arbre de plus d'un nœud.

Exemples :

```
>> leaf(A).
A ~ leaf.
>> leaf(1).
true.
>> leaf(cc(1,2)).
true.
>> leaf(toto(X)).
false.
>> list(L), leaf(L).
L = [].
```

Voir également : bleaf/2, bnleaf/2, nleaf/1.

lelin/2 — Inférieur ou égal

Définition : **lelin/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a \leq b\}$.

Description : **lelin/2** est une relation sur les réels, qui est traitée par le solveur linéaire. Elle établit une relation entre deux valeurs numériques telles que la première est inférieure ou égale à la seconde.

Exemples :

```
>> lelin(A, B).
B ~ real,
A ~ real.
>> lelin(A, B), lelin(B, A).
B = A,
A ~ real.
>> lelin(A, B), lelin(B, A), dif(A, B).
false.
```

Voir également : gelin/2, gtlin/2, le/2, ltlin/2.

list/1 — Etre une liste

Définition : **list/1** définit la relation $\{a \in \mathbf{A} \mid a \text{ est une liste}\}$.

Description : **list/1** est une relation générale sur les arbres, qui est traitée par le solveur général. Une contrainte **list**(a) est vérifiée quand son argument a représente une liste.

list/1 est une relation, et pas une primitive de vérification. Si une variable `X` est contrainte par `list(X)`, elle ne peut plus s'unifier avec un arbre qui n'est pas une liste.

Exemples :

```
>> list(A).
A ~ list.
>> list([1,2]).
true.
>> list(1).
false.
>> list([X|L]).
X ~ tree,
L ~ list.
>> leaf(X), list(X).
X = [].
```

Voir également : blist/2, bnlist/2, nlist/1.

ln/2 — Logarithme népérien

Définition : **ln/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid b > 0 \text{ et } a = \ln b\}$.

Description : ln/2 est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique et son logarithme naturel (de base e, où $e = 2.718281828459\ldots$).

Exemples :

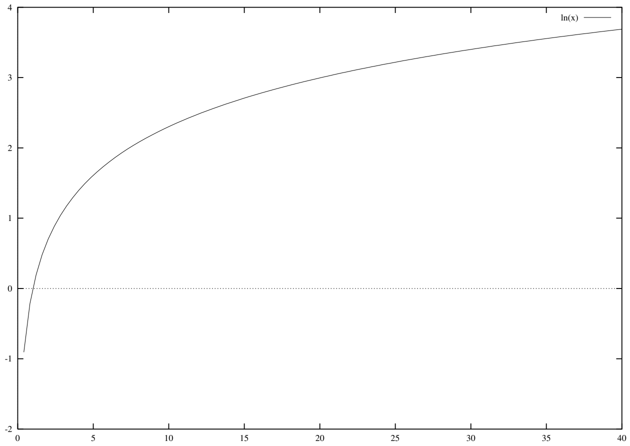
```
>> ln(A, B).
A ~ real,
B ~ gt(0).
>> ln(-1, B).
B ~ cc(`>0.3678794`,`>0.36787945`).
>> ln(0, B).
B = 1.
>> ln(1, B).
B ~ cc(`>2.7182817`,`<2.718282`).
>> ln(A, 1).
A = 0.
>> ln(A, 10).
A ~ cc(`<2.302585`,`>2.302585`).
>> ln(A, A).
A ~ gt(`<4e38`).
```

La contrainte $b > 0$ est appliquée au deuxième argument b de la contrainte **ln**(a, b). Ceci peut donc résulter en des échecs ou en des réductions de domaine :

```
>> ln(A, -2).
false.
>> B ~ cc(-2,2), A ~ ln(B).
A ~ le(`<0.6931472`),
B ~ oc(0,2).
```

Note : **ln/2** est ici défini comme une relation. Il n'y a donc pas de problème pour le traitement des logarithmes de nombres négatifs (voir exemples ci-dessus).

Graphe :



Voir également : exp/2, log/2, power/3, root/3, square/2, sqrt/2.

log/2 — Logarithme Décimal

Définition : **log/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid b > 0 \text{ et } a = \log b\}$.

Description : log/2 est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique et son logarithme de base 10.

Exemples :

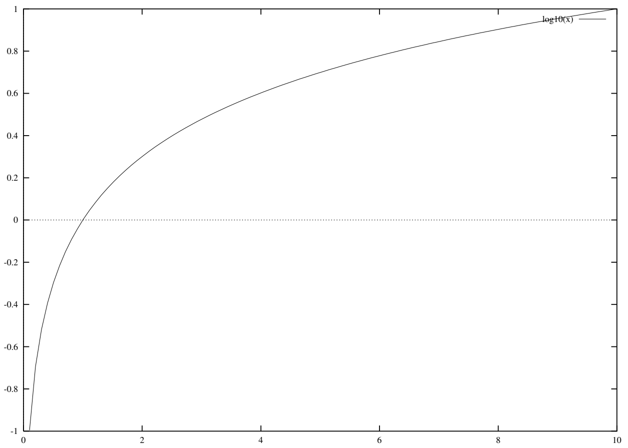
```
>> log(A, B).
A ~ real,
B ~ gt(0).
>> log(-1, B).
B = 1/10.
>> log(0, B).
B = 1.
>> log(1, B).
B = 10.
>> log(2, B).
B = 100.
>> log(A, 1).
A = 0.
>> log(A, 10).
A = 1.
>> log(A, 100).
A = 2.
>> log(A, A).
A ~ gt(`<4e38`).
```

La contrainte $b > 0$ est appliquée au deuxième argument b de la contrainte **log**(a, b). Ceci peut donc résulter en des échecs ou en des réductions de domaine :

```
>> log(A, -2).
false.
>> B ~ cc(-2,2), A ~ log(B).
A ~ le(`>0.30103003`),
B ~ oc(0,2).
```

Note : **log/2** est ici défini comme une relation. Il n'y a donc pas de problème pour le traitement des logarithmes de nombres négatifs (voir exemples ci-dessus).

Graphe :



Voir également : exp/2, ln/2, power/3, root/3, square/2, sqrt/2.

lt/2 — Inférieur

Définition : **lt/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a < b\}$.

Description : **lt/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre deux valeurs numériques telles que la première est strictement inférieure à la seconde.

Exemples :

```
>> lt(A, B).
B ~ real,
A ~ real.
>> lt(A, 0).
A ~ lt(0).
>> A ~ cc(0,4), B ~ cc(1,3), lt(A, B).
B ~ cc(1,3),
A ~ co(0,3).
>> A ~ cc(0,4), B ~ cc(1,3), lt(B,A).
B ~ cc(1,3),
A ~ oc(1,4).
```

La relation **lt/2** apparaît souvent dans les réponses de Prolog IV, dans la mesure où un pseudo-terme **lt**(v) désigne en fait l'intervalle $(-\infty, v)$.

Voir également : blt/3, ge/2, gt/2, le/2, ltlin/2.

ltlin/2 — Inférieur

Définition : **ltlin/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a < b\}$.

Description : **ltlin/2** est une relation sur les réels, qui est traitée par le solveur linéaire. Elle établit une relation entre deux valeurs numériques telles que la première est strictement inférieure à la seconde.

Exemples :

```
>> ltlin(A, B).
B ~ real,
A ~ real.
>> ltlin(A, B), ltlin(B, A).
false.
```

Voir également : gelin/2, gtlin/2, lelin/2, lt/2.

max/3 Maximum

Définition : **max/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid a = \max(b, c)\}$.

Description : **max/3** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre deux valeurs et leur maximum.

Exemples :

```
>> max(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> A ~ max(cc(1,3), cc(2,4)).
A ~ cc(2,4).
>> A ~ max(0, C).
C ~ real,
A ~ ge(0).
>> max(0, B, C).
C ~ le(0),
B ~ le(0).
```

Voir également : min/3.

min/3 Minimum

Définition : **min/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid a = \min(b, c)\}$.

Description : **min/3** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre deux valeurs et leur minimum.

Exemples :

```
>> min(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> min(0, B, C).
C ~ ge(0),
B ~ ge(0).
>> A ~ min(cc(1,3), cc(2,4)).
A ~ cc(1,3).
>> A ~ min(0, C).
C ~ real,
A ~ le(0).
```

Voir également : max/3.

minus/3

.-. Soustraction

Définition : **minus/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid a = b - c\}$.

Description : **minus/3** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre deux nombres et leur différence.

Autre notation : **minus**(a, b, c) s'écrit aussi $a = b$ **.-.** c.

Exemples :

```
>> minus(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> A ~ minus(3, 1).
A = 2.
>> A ~ cc(1,4) .-. cc(1,2).
A ~ cc(-1,3).
>> A ~ A .-. 1 .
A ~ real.
>> minus(0, B, C).
B = C,
C ~ real.
```

Note : Quand deux des arguments de **minus/3** sont réduits à des singletons, le troisième est calculé de manière exacte. Nous avons donc :

```
>> A ~ 2/3 .-. 1/3 .
A ~ 1/3.
```

Voir également : div/3, minuslin/3, plus/3, times/3, uminus/2, uplus/2.

minuslin/3

- Soustraction

Définition : **minuslin/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid a = b - c\}$.

Description : **minuslin/2** est une relation sur les réels, qui est traitée par le solveur linéaire. Elle établit une relation entre deux nombres et leur différence.

Autre notation : **minuslin**(a, b, c) s'écrit aussi $a = b$ **-** c.

Exemples :

```
>> minuslin(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> A ~ minuslin(3, 1).
A = 2.
>> A ~ A - 1 .
false.
>> minuslin(0, B, C).
B = C,
C ~ real.
```

Voir également : divlin/3, minus/3, pluslin/3, timeslin/3, uminuslin/2, upluslin/2.

mod/3 — Modulo

Définition : **mod/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid 0 \leq a < c \text{ et } a = b \bmod c\}$.

Description : La relation **mod/3** n'est pas implantée dans la version actuelle de Prolog IV. Toutefois, étant donné qu'elle fait partie de la définition du langage, le symbole a été réservé.

Exemples :

```
>> mod(A, B, C).
Relation not yet implemented: mod
false.
```

Voir également : intdiv/3

n/3 — Intersection

Définition : **n/3** définit la relation $\{(a, b, c) \in \mathbf{A}^3 \mid a = b \text{ et } a = c\}$.

Description : **n/3** est une relation générale sur les arbres. Une contrainte $\mathbf{n}(a, b, c)$ est vérifiée si ses trois arguments sont égaux.

Le nom de **n/3** a été choisi car il ressemble au symbole de l'intersection. C'est en effet l'usage le plus courant de cette relation, au niveau des pseudo-termes. Se reporter à la note après les exemples pour plus d'explications à ce sujet.

Autre notation : $\mathbf{n}(a, b, c)$ s'écrit aussi $a = b \ \mathbf{n} \ c$.

Exemples :

```
>> n(A, B, C).
B = C,
A = C,
C ~ tree.
>> n(A, cc(1,3), cc(2,4)).
A ~ cc(2,3).
>> A ~ int n cc(1,5) .
A ~ cc(1,5).
>> set_prolog_flag(interval_mode,union).
true.
>> A ~ int n cc(1,5) .
A ~ 1 u 2 u 3 u 4 u 5.
>> A ~ int n cc(1,10) n 2 .*. nint .
A ~ 1 u 3 u 5 u 7 u 9.
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : Attention, «n» n'est pas l'intersection, puisque Prolog IV ne raisonne que sur les nombres, et non sur les ensembles. La requête ci-dessous donne le résultat prévu par la définition de la relation, mais n'est bien sûr pas compatible avec ce qu'on attendrait d'une intersection :

```
>> cc(1,5) ~ cc(1,5) n A .
A ~ cc(1,5).
```

Considérons un autre exemple. Quand on utilise «n» en tant que pseudo-terme, on pense immédiatement à l'intersection, comme par exemple dans la requête suivante :

```
>> set_prolog_flag(interval_mode,union).
true.
>> X ~ cc(1,10) n int n dif(5).
X ~ 1 u 2 u 3 u 4 u 6 u 7 u 8 u 9 u 10.
```

En décomposant cette pseudo-expression, on obtient :

```
>> cc(A,1,10), n(C,A,B), int(B), dif(D,5), n(X,C,D).
D = X,
C = X,
B = X,
A = X,
X ~ cc(1,10).
```

Dans ce résultat, on voit bien les variables toutes égales.

Voir également : u/3.

nidentifier/1 — Ne pas être un identificateur

Définition : **nidentifier/1** définit la relation $\{a \in \mathbf{A} \mid a$ n'est pas un identificateur$\}$.

Description : **nidentifier/1** est une relation générale sur les arbres, qui est traitée par le solveur général. Une contrainte **nidentifier**(a) est vérifiée quand son argument a représente un arbre qui n'est pas un identificateur.

nidentifier/1 est une relation, et pas une primitive de vérification. Si une variable `X` est contrainte par `nidentifier(X)`, elle ne peut plus s'unifier avec un identificateur.

Exemples :

```
>> nidentifier(A).
A ~ nidentifier.
>> nidentifier(toto).
false.
>> nidentifier(cc(1,2)).
true.
>> list(X), nidentifier(X).
X ~ [tree|list].
```

Voir également : bidentifier/2, bnidentifier/2, identifier/1.

nint/1 — Ne pas être un entier

Définition : **nint/1** définit la relation $\{a \in \mathbf{R} \mid a$ n'est pas entier$\}$.

Description : **nint/1** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. **nint/1** réussit quand son argument est un nombre non entier.

L'interprétation de **nint/1** est définie sur l'ensemble des réels. Cette relation échoue si son argument prend une valeur non numérique.

nint/1 est une relation, et pas une primitive de vérification. Si une variable `X` est contrainte par `nint(X)`, elle ne peut plus s'unifier avec un nombre entier.

Exemples :

```
>> nint(A).
A ~ real.
>> nint(1.5).
true.
>> A ~ cc(1,3) n nint .
A ~ oo(1,3).
>> set_prolog_flag(interval_mode,union).
true.
>> A ~ cc(1,3) n nint .
A ~ oo(1,2)u oo(2,3).
>> A ~ cc(1,10) n 3 .*. nint .
A ~ co(1,3)u oo(3,6)u oo(6,9)u oc(9,10).
>> A ~ cc(1,10) n int n 3 .*. nint .
A ~ 1 u 2 u 4 u 5 u 7 u 8 u 10.
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : L'interprétation de **nint/1**, soit l'ensemble des réels non entiers, n'est pas un sous-domaine privilégié de Prolog IV. En conséquence, les contraintes **nint/1** ne sont pas reportées lors de l'écriture des solutions.

Voir également : bint/2, bnint/2, bnprime/2, bprime/2, int/1, nprime/1, prime/1.

nleaf/1 — Etre un arbre de plus d'un nœud

Définition : **nleaf/1** définit la relation $\{a \in \mathbf{A} \mid a$ est un arbre de plus d'un nœud$\}$.

Description : **nleaf/1** est une relation générale sur les arbres, qui est traitée par le solveur général. Une contrainte **nleaf**(a) est vérifiée quand son argument a représente un arbre de plus d'un nœud.

nleaf/1 est une relation, et pas une primitive de vérification comme **compound/1**. Si une variable `X` est contrainte par `nleaf(X)`, elle ne peut plus s'unifier avec un arbre qui n'est pas un identificateur.

Exemples :

```
>> nleaf(A).
A ~ nleaf.
>> nleaf(2).
false.
>> nleaf(f(X)).
X ~ tree.
>> list(L), nleaf(X).
L ~ list,
X ~ nleaf.
```

Voir également : bleaf/2, bnleaf/2, leaf/1

nlist/1 — Ne pas être une liste

Définition : **nlist/1** définit la relation $\{a \in \mathbf{A} \mid a$ n'est pas une liste$\}$.

Description : **nlist/1** est une relation générale sur les arbres, qui est traitée par le solveur général. Une contrainte **nlist**(a) est vérifiée quand son argument a ne représente pas une liste.

nlist/1 est une relation, et pas une primitive de vérification. Si une variable `X` est contrainte par `nlist(X)`, elle ne peut plus s'unifier avec une liste.

Exemples :

```
>> nlist(X).
X ~ nlist.
>> nlist([X|Y]).
X ~ tree,
Y ~ nlist.
>> nlist(cc(1,2)).
true.
>> nlist([1,2,3]).
false.
```

Voir également : blist/2, bnlist/2, list/1.

not/1 — Négation

Définition : **not/1** définit la relation $\{a \in \mathbf{B} \mid a = 0\}$.

Description : **not/1** est une relation sur les booléens, qui est traitée par le solveur sur les intervalles. Une contrainte **not**(a) réussit si son argument a représente la valeur faux (0).

Une telle contrainte est normalement utilisée pour construire une négation d'autres contraintes exprimées par des pseudo-opérations produisant des booléens.

Attention, **not/1** est ici une relation, et n'a pas de relation avec le prédicat **not/1** (ou $\backslash +$ fourni par certains systèmes Prolog (dont Prolog II+ et Prolog III), qui implémente généralement la négation par échec.

Exemples :

```
>> not(A).
A = 0.
>> not(bor(A,B)).
B = 0,
A = 0.
>> not(bnot(A)).
A = 1.
>> not(band(A,B)).
B ~ cc(0,1),
A ~ cc(0,1).
```

Voir également : and/2, bnot/2, equiv/2, impl/2, or/2, xor/2.

nprime/1 — Ne pas être un entier premier

Définition : **nprime/1** définit la relation $\{a \in \mathbf{R} \mid a \text{ n'est pas un entier premier}\}$.

Description : La relation **nprime/1** n'est pas implantée dans la version actuelle de Prolog IV. Toutefois, étant donné qu'elle fait partie de la définition du langage, le symbole a été réservé.

Exemples :

```
>> nprime(A).
Relation not yet implemented: nprime
false.
```

Voir également : bint/2, bnint/2, bnprime/2, bprime/2, int/1, nint/1, prime/1.

nreal/1 — Ne pas être un nombre réel

Définition : **nreal/1** définit la relation $\{a \in \mathbf{A} \mid a \text{ n'est pas un nombre réel}\}$.

Description : **nreal/1** est une relation générale sur les arbres, qui est traitée par le solveur général. Une contrainte **nreal**(a) est vérifiée quand son argument a ne représente pas un nombre réel.

nreal/1 est une relation, et pas une primitive de vérification. Si une variable `X` est contrainte par `nreal(X)`, elle ne peut plus s'unifier avec un arbre qui n'est pas un nombre réel.

Exemples :

```
>> nreal(A).
A ~ nreal.
>> nreal(toto).
true.
>> nreal(1).
false.
>> nreal(cc(1,2)).
false.
```

Voir également : breal/2, bnreal/2, real/1.

ntree/1 — Ne pas être un arbre

Définition : **ntree/1** définit la relation $\{a \in \mathbf{A} \mid a \text{ ne peut exister}\}$.

Description : **ntree/1** est une relation générale dont l'interprétation est l'ensemble vide. Elle ne réussit donc jamais, quelque soit l'argument fourni.

Exemples :

```
>> ntree(A).
false.
>> ntree(toto).
false.
>> ntree(1).
false.
```

Voir également : tree/1.

oc/3 — Appartenance à un intervalle

Définition : **oc/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid a \in (b, c]\}$.

Description : **oc/3** est une relation sur les nombres réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un nombre et les bornes d'un intervalle ouvert à gauche et fermé à droite de la forme $(b, c]$ le contenant.

Exemples :

```
>> oc(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> oc(A, 1, 2).
A ~ oc(1,2).
>> oc(0, B, C).
C ~ ge(0),
B ~ lt(0).
>> oc(A, 0, 0).
false.
>> oc(A, 0, C).
C ~ gt(0),
A ~ gt(0).
>> A ~ cc(0,2), B ~ oc(1,3), C = A .+. B .
C ~ oc(1,5),
B ~ oc(1,3),
A ~ cc(0,2).
```

Note : Voir **cc/3** pour de plus amples explications sur les exemples ci-dessus.

Voir également : boc/4, boutoc/4, cc/3, co/3, oo/3, outoc/3.

oo/3 — Appartenance à un intervalle

Définition : **oo/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid a \in (b, c)\}$.

Description : **oo/3** est une relation sur les nombres réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un nombre et les bornes d'un intervalle ouvert des deux côtés de la forme (b, c) le contenant.

Exemples :

```
>> oo(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> oo(A, 1, 2).
A ~ oo(1,2).
>> oo(0, B, C).
C ~ gt(0),
B ~ lt(0).
>> oo(A, 0, 0).
false.
>> oo(A, 0, C).
C ~ gt(0),
A ~ gt(0).
>> A ~ cc(0,2), B ~ oo(1,3), C = A .+. B .
C ~ oo(1,5),
B ~ oo(1,3),
A ~ cc(0,2).
```

Note : Voir **cc/3** pour de plus amples explications sur les exemples ci-dessus.

Voir également : boo/4, boutoo/4, cc/3, co/3, oc/3, outoo/3.

or/2 — Ou

Définition : **or/2** définit la relation $\{(a, b) \in \mathbf{B}^2 \mid a = 1 \text{ ou } b = 1\}$.

Description : **or/2** est une relation sur les booléens, qui est traitée par le solveur sur les intervalles. Une contrainte **or**(a, b) réussit si un de ses deux arguments vaut 1.

Cette contrainte est normalement utilisée pour construire une combinaison d'autres contraintes exprimées par des pseudo-opérations produisant des booléens.

Autre notation : **or**(a, b) s'écrit aussi a **or** b.

Exemples :

```
>> or(A, B).
B ~ cc(0,1),
A ~ cc(0,1).
>> or(0, B).
B = 1.
>> or(1, B).
B ~ cc(0,1).
```

Voir également : and/2, band/3, bequiv/3, bimpl/3, bnot/2, bor/3, bxor/3, equiv/2, impl/2, not/1, xor/2.

outcc/3 — Non-appartenance à un intervalle

Définition : **outcc/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid a \notin [b, c]\}$.

Description : **outcc/3** est une relation sur les nombres réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un nombre et les bornes d'un intervalle fermé des deux côtés de la forme $[b, c]$ ne le contenant pas.

Exemples :

```
>> outcc(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> outcc(A, 1, 2).
A ~ real.
>> A ~ outcc(1,3) n cc(1,4).
A ~ oc(3,4).
>> set_prolog_flag(interval_mode,union).
true.
>> outcc(A, 1, 2).
A ~ lt(1)u gt(2).
>> A ~ outcc(1,3) n cc(1,4).
A ~ oc(3,4).
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : Comme on l'a vu dans les exemples, il est courant en mode intervalles simples que la pose d'une contrainte **outcc**(a, b, c) n'effectue pas de réduction immédiate. Toutefois, ces contraintes, comme celles construites avec **dif/2** sont avant tout destinées à être dormantes jusqu'à ce que les domaines des variables soient suffisamment réduits.

Voir également : bcc/4, boutcc/4, cc/3, outco/3, outoc/3, outoo/3.

outco/3 ______________ Non-appartenance à un intervalle

Définition : **outco/3** définit la relation $\{(a,b,c) \in \mathbf{R}^3 \mid a \notin [b,c)\}$.

Description : **outco/3** est une relation sur les nombres réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un nombre et les bornes d'un intervalle fermé à gauche et ouvert à droite de la forme $[b,c)$ ne le contenant pas.

Exemples :

```
>> outco(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> outco(A, 1, 2).
A ~ real.
>> A ~ outco(1,3) n cc(1,4) .
A ~ cc(3,4).
>> set_prolog_flag(interval_mode,union).
true.
>> outco(A, 1, 2).
A ~ lt(1)u ge(2).
>> A ~ outco(1,3) n cc(1,4) .
A ~ cc(3,4).
>> set_prolog_flag(interval_mode,simple).
true.
```

Voir également : bco/4, boutco/4, co/3, outcc/3, outoc/3, outoo/3.

outlist/2 ______________ Non-appartenance à une liste

Définition : **outlist/2** définit la relation $\{(a,b) \in \mathbf{A} \times \mathbf{L} \mid a$ ne figure pas dans $b\}$.

Description : **outlist/2** est une relation sur les listes, qui est traitée par le solveur général. Une contrainte **outlist**(a,b) réussit si son premier argument a n'est pas un élément de son second argument b.

Exemples :

```
>> outlist(A, B).
A ~ tree,
B ~ list.
>> outlist(A, []).
A ~ tree.
>> outlist(A,[1,2,3]).
A ~ real.
```

Attention, ces contraintes sont évaluées par approximation. Avec des valeurs non numériques, il est donc indispensable que la liste soit entièrement connue pour avoir des résultats satisfaisants :

```
>> outlist(toto, [X,toto]).
X ~ tree.
```

Il en est de même si certaines des variables ne sont pas typées numériques :

```
>> outlist(1,[_,_,1,_]).
true.
```

Par contre on obtient le résultat attendu dès que les variables sont typées, même si leurs valeurs sont inconnues (ceci est dû au fait que les produits cartésiens d'intervalles sont des sous-domaines utilisés dans l'approximation de Prolog IV) :

```
>> outlist(1,[real,real,1,real]).
false.
>> set_prolog_flag(interval_mode,union).
true.
>> outlist(A,[1,2,3]).
A ~ lt(1)u oo(1,2)u oo(2,3)u gt(3).
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : Voir **inlist/2** pour des remarques importantes concernant le fonctionnement opérationnel de cette contrainte.

Voir également : binlist/3, boutlist/3, intlist/2.

outoc/3 — Non-appartenance à un intervalle

Définition : **outoc/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid a \notin (b, c]\}$.

Description : **outoc/3** est une relation sur les nombres réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un nombre et les bornes d'un intervalle ouvert à gauche et fermé à droite de la forme $(b, c]$ ne le contenant pas.

Exemples :

```
>> outoc(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> outoc(A, 1, 2).
A ~ real.
>> A ~ outoc(1,3) n cc(2,4) .
A ~ oc(3,4).
>> A ~ outoc(1,3) n cc(1,4) .
A ~ cc(1,4).
>> set_prolog_flag(interval_mode,union).
true.
>> outoc(A, 1, 2).
A ~ le(1)u gt(2).
>> A ~ outoc(1,3) n cc(1,4) .
A ~ 1 u oc(3,4).
>> set_prolog_flag(interval_mode,simple).
true.
```

Voir également : boc/4, boutoc/4, oc/3, outcc/3, outco/3, outoo/3.

outoo/3 — Non-appartenance à un intervalle

Définition : **outoo/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid a \notin (b, c)\}$.

Description : **outoo/3** est une relation sur les nombres réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un nombre et les bornes d'un intervalle ouvert des deux côtés de la forme (b, c) ne le contenant pas.

Exemples :

```
>> outoo(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> outoo(A, 1, 2).
A ~ real.
>> A ~ outoo(1,3) n cc(2,4) .
A ~ cc(3,4).
>> A ~ outoo(1,3) n cc(1,4) .
A ~ cc(1,4).
>> set_prolog_flag(interval_mode,union).
true.
>> outoo(A, 1, 2).
A ~ le(1)u ge(2).
>> A ~ outoo(1,3) n cc(1,4) .
A ~ 1 u cc(3,4).
```

Voir également : boo/4, boutoo/4, oo/3, outcc/3, outco/3, outoc/3.

pi/1 — π

Définition : **pi/1** définit la relation $\{a \in \mathbf{R} \mid a = \pi\}$.

Description : **pi/1** est une relation sur les réels qui est vraie si son argument prend la valeur π.

Etant donné que π n'est pas représentable avec les rationnels IEEE disponibles en Prolog IV, la meilleure approximation possible est utilisée, c'est-à-dire l'intervalle ouvert des deux côtés dont les bornes inférieures et supérieures sont respectivement le plus grand rationnel IEEE inférieur à π et le plus petit rationnel IEEE supérieur à π.

Exemples :

```
>> pi(A).
A ~ oo('>3.1415925','>3.1415927').
>> A ~ sin(pi).
A ~ oo(-'>1.6292068e-7','<1.947072e-7').
```

Il n'est pas possible d'unifier π avec une constante en Prolog IV, car le langage sait qu'il n'existe pas de représentation rationnelle de π ; or toutes les constantes du langage sont rationnelles :

```
>> pi(3.141592653589793238).
false.
```

Une erreur classique est d'utiliser `-pi`. Ceci ne fonctionne pas, car ici, `pi` est un pseudo-terme, utilisé pour poser une contrainte dans le solveur des intervalles. Etant donné que « – » est le raccourci pour la négation unaire du solveur linéaire, la communication ne s'effectue pas entre les solveurs et nous obtenons la requête suivante (au lieu du résultat correct donné en-dessous) :

```
>> A = -pi.
A ~ real.
>> A ~ .-. pi .
A ~ oo(-`>3.1415927`,-`>3.1415925`).
```

Voir également : arccos/2, arcsin/2, arctan/2, cos/2, cot/2, sin/2, tan/2.

plus/3

.+. — Addition

Définition : **plus/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid a = b + c\}$.

Description : **plus/3** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre deux nombres et leur somme.

Autre notation : **plus**(a, b, c) s'écrit aussi $a = b$ **.+.** c.

Exemples :

```
>> plus(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> A ~ plus(3, 1).
A = 4.
>> A ~ cc(1,4) .+. cc(1,2).
A ~ cc(2,6).
>> A ~ A .+. 1 .
A ~ real.
>> plus(A, B, 0).
A = B,
B ~ real.
```

Note : Quand deux des arguments de **plus/3** sont réduits à des singletons, le troisième est calculé de manière exacte. Nous avons donc :

```
>> A ~ 1/3 .+. 1/3 .
A ~ 2/3.
```

Voir également : div/3 minus/3, pluslin/3, times/3, uminus/2, uplus/2.

pluslin/3

+ — Addition

Définition : **pluslin/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid a = b + c\}$.

Description : **pluslin/2** est une relation sur les réels, qui est traitée par le solveur linéaire. Elle établit une relation entre deux nombres et leur somme.

Autre notation : **pluslin**(a, b, c) s'écrit aussi $a = b$ **+** c.

Exemples :

```
>> pluslin(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> A ~ pluslin(3, 1).
A = 4.
>> A ~ A + 1 .
false.
>> pluslin(A, B, 0).
A = B,
B ~ real.
```

Voir également : divlin/3, minuslin/3, plus/3, timeslin/3, uminuslin/2, upluslin/2.

power/3

.^. Puissance nième

Définition : **power/3** définit la relation $\{(a, b, c) \in \mathbf{R}^2 \times \mathbf{Z} \mid c \geq 0 \text{ et } a = b^c\}$.

Description : L'implantation de **power/3** n'est pas totalement conforme à la définition ci-dessus. En fait, **power/3** est implantée comme une suite infinie de relations binaires **power**$_1$**/2**, **power**$_2$**/2**, ..., où l'interprétation de chacun des **power**$_i$**/2** est donnée par :

$\{(a, b) \in \mathbf{R}^2 \mid a = b^i\}$.

Dans la pratique, cela se ressent dans l'implantation de **power/3** par le fait qu'aucune réduction ne peut être effectuée sur le domaine du troisième argument. L'exploitation des contraintes **power**(a, b, c) est donc retardée jusqu'à ce que la valeur de ce troisième argument c soit connue.

Autre notation : **power/3**(a, b, c) s'écrit aussi $a = b$ **.^.** c.

Exemples :

```
>> power(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> A ~ power(2,3).
A = 8.
>> A ~ 1.01 .^. 36.
A ~ oo('>1.4307682','>1.4307744').
>> A ~ power(cc(1,3),2).
A ~ cc(1,9).
>> set_prolog_flag(interval_mode,union).
true.
>> A ~ power(cc(-2,-1) u cc(1,2),2).
A ~ cc(1,4).
>> A ~ power(cc(-2,-1) u cc(1,2),3).
A ~ cc(-8,-1)u cc(1,8).
```

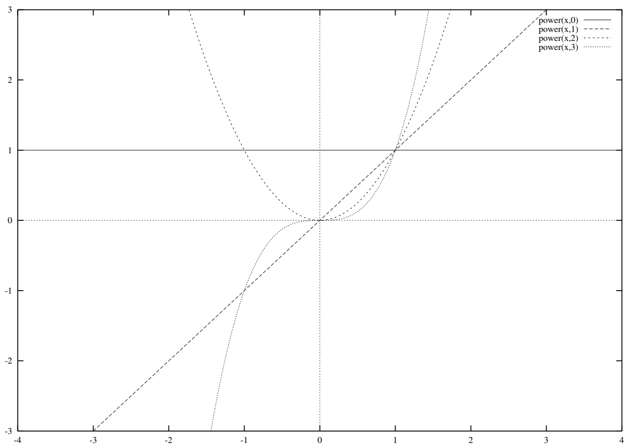
Les contraintes **power**(a, b, c) sont donc retardées jusqu'à ce que la valeur du dernier argument c soit connue :

```
>> A ~ power(cc(1,3),cc(2,3)).
A ~ real.
```

Par convention, $\forall x, x^0 = 1$, et donc en particulier, $0^0 = 1$.

```
>> power(A, B, 0).
A = 1,
B ~ real.
>> A ~ 0 .^. 0.
A = 1.
```

Graphe :



Voir également : exp/2, ln/2, log/2, root/3, square/2, sqrt/2.

prime/1 — Etre un entier premier

Définition : **prime/1** définit la relation $\{a \in \mathbf{R} \mid a \text{ est un entier premier}\}$.

Description : La relation **prime/1** n'est pas implantée dans la version actuelle de Prolog IV. Toutefois, étant donné qu'elle fait partie de la définition du langage, le symbole a été réservé.

Exemples :

```
>> prime(A).
Relation not yet implemented: prime
false.
```

Voir également : int/1, nint/1, nprime/1.

real/1 — Etre un nombre réel

Définition : **real/1** définit la relation $\{a \in \mathbf{A} \mid a \text{ est un nombre réel}\}$.

Description : **real/1** est une relation générale sur les arbres, qui est traitée par le solveur général. Une contrainte **real**(a) est vérifiée quand son argument représente un nombre réel.

real/1 est une relation, et pas une primitive de vérification comme **number/1**. Si une variable `X` est contrainte par `real(X)`, elle ne peut plus s'unifier avec un arbre qui n'est pas un nombre réel.

Attention, comme toutes les relations numériques de Prolog IV, la relation **real/1** n'est pas compatible avec les nombres flottants de la norme ISO.

Exemples :

```
>> real(A).
A ~ real.
>> real(a).
false.
>> real(1).
true.
>> real(f(X)).
false.
>> real(1 + X + Y).
Y ~ real,
X ~ real.
```

Voir également : breal/2, bnreal/2, nreal/1.

root/3 — Racine nième

Définition : **root/3** définit la relation $\{(a, b, c) \in \mathbf{R}^2 \times \mathbf{Z} \mid c > 0 \text{ et } a = \sqrt[c]{b}\}$.

Description : L'implantation de **root/3** n'est pas totalement conforme à la définition ci-dessus. En fait, **root/3** est implantée comme une suite infinie de relations binaires **root**$_1$**/2**, **root**$_2$**/2**, $\ldots$, où l'interprétation de chacun des **root**$_i$**/2** est donnée par :

$\{(a, b) \in \mathbf{R}^2 \mid a = \sqrt[i]{b}\}$.

Dans la pratique, cela se ressent dans l'implantation de **root/3** par le fait qu'aucune réduction ne peut être effectuée sur le domaine du troisième argument. L'exploitation des contraintes **root**(a, b, c) est donc retardée jusqu'à ce que la valeur de ce troisième argument soit connue.

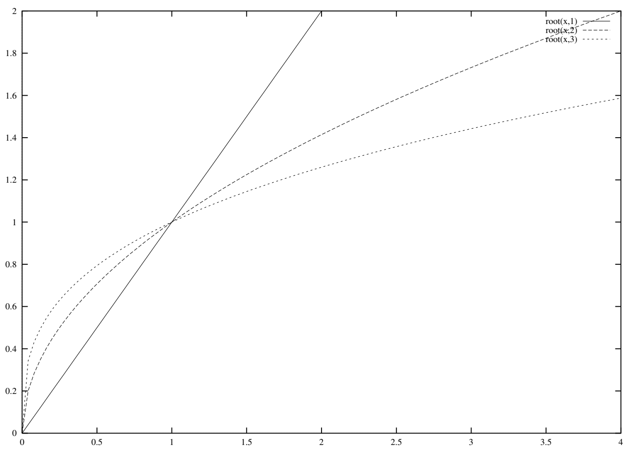
Exemples :

```
>> root(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> A ~ root(2,3).
A ~ cc('<1.259921','>1.259921').
>> A ~ root(0.12, 36).
A ~ oo('<0.9428046','>0.9428046').
>> A ~ root(cc(1,3),2).
A ~ cc(1,'<1.732051').
```

Par convention, $\forall x, x^0 = 1$. En transposant aux racines, nous obtenons donc :

```
>> root(A, B, 0).
B = 1,
A ~ real.
>> A ~ root(1,0).
A ~ real.
```

Graphe :



Voir également : exp/2, ln/2, log/2, power/3, square/2, sqrt/2.

sin/2 — Sinus

Définition : **sin/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a = \sin b\}$.

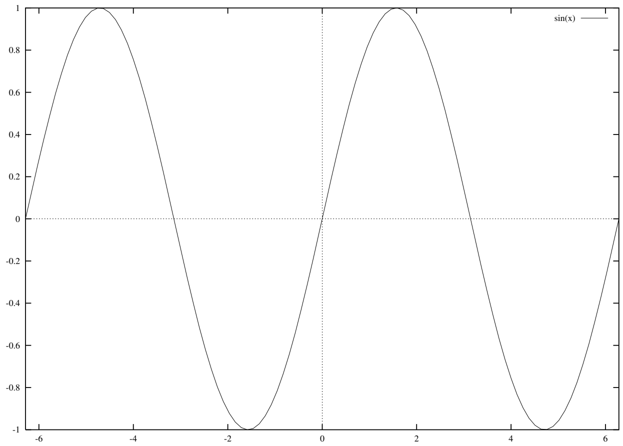
Description : **sin/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique et son sinus.

Exemples :

```
>> sin(A, B).
B ~ real,
A ~ cc(-1,1).
>> sin(A, 0).
A = 0.
>> A = 2.*.square(sin(pi./.4)).
A ~ oo(`<0.9999984`,`>1.000001`).
>> A = 4.*.square(sin(pi./.3)).
A ~ oo(`<2.999997`,`>3.0000016`).
>> sin(A, pi./.2).
A ~ oc(`>0.9999999`,1).
>> sin(A, A).
A ~ oo(-`>0.017607333`,`<0.01740003`).
```

Note : Voir **cos/2** pour des remarques importantes concernant l'utilisations de relations associées à des fonctions périodiques en mode unions d'intervalles.

Graphe :



Voir également : arccos/2, arcsin/2, arctan/2, cos/2, cot/2, pi/2, tan/2.

sinh/2 — Sinus hyperbolique

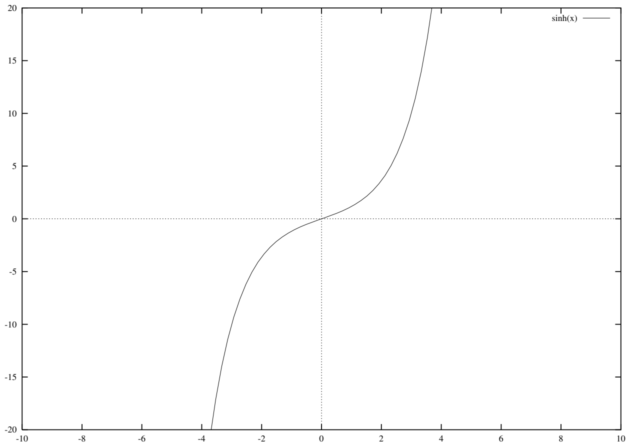
Définition : **sinh/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a = \sinh b\}$.

Description : **sinh/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique et son sinus hyperbolique.

Exemples :

```
>> sinh(A, B).
B ~ real,
A ~ real.
>> sinh(0, B).
B = 0.
>> sinh(1, B).
B ~ cc('<0.8813736','>0.8813736').
>> sinh(A, 0).
A = 0.
>> sinh(A, 1).
A ~ cc('>1.1752011','>1.1752012').
>> sinh(A, 10).
A ~ cc('>11013.232','>11013.233').
>> sinh(A, 100).
A ~ ge('<4e38').
>> sinh(A, A).
A ~ real.
```

Graphe :



Voir également : cosh/2, coth/2, tanh/2.

size/2 — Taille d'une liste

Définition : **size/2** définit la relation $\{(a, b) \in \mathbf{Z} \times \mathbf{L} \mid a = |b|\}$.

Description : **size/2** est une relation sur les listes, qui est traitée par le solveur général. Elle établit une relation entre une liste et une valeur numérique représentant sa longueur.

Exemples :

```
>> size(A, B).
B ~ list,
A ~ ge(0).
>> size(0, B).
B = [].
>> size(B) ~ 4 .
B ~ [tree,tree,tree,tree].
```

La contrainte **size**(a, b) ne fait des déductions sur la longueur a que quand celle-ci devient connue. Nous avons donc :

```
>> size(A, [1, 2, 3, 4 | B]).
B ~ list,
A ~ ge(0).
```

La variable `A` pourrait ici être contrainte à appartenir à `ge(4)`.

Voir également : conc/3, index/3.

sqrt/2 Racine carré

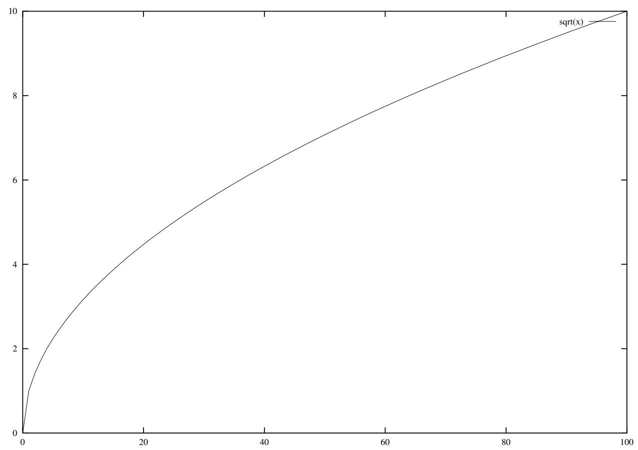
Définition : **sqrt/2** définit la relation $\{(a,b) \in \mathbf{R}^2 \mid b \geq 0 \text{ et } a = \sqrt{b}\}$.

Description : **sqrt/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique x et sa racine carrée $\sqrt{x}$.

Exemples :

```
>> sqrt(A, B).
B ~ ge(0),
A ~ ge(0).
>> A ~ sqrt(2).
A ~ cc('>1.4142135','>1.4142136').
>> A ~ cc(-2,2), B ~ sqrt(A).
B ~ cc(0,'>1.4142136'),
A ~ cc(0,2).
>> cc(2,3) ~ sqrt(B).
B ~ cc(4,9).
```

Graphe :



Voir également : exp/2, ln/2, log/2, power/3, root/3, square/2.

square/2 Carré

Définition : **square/2** définit la relation $\{(a,b) \in \mathbf{R}^2 \mid a = b^2\}$.

Description : **square/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique x et son carré x^2.

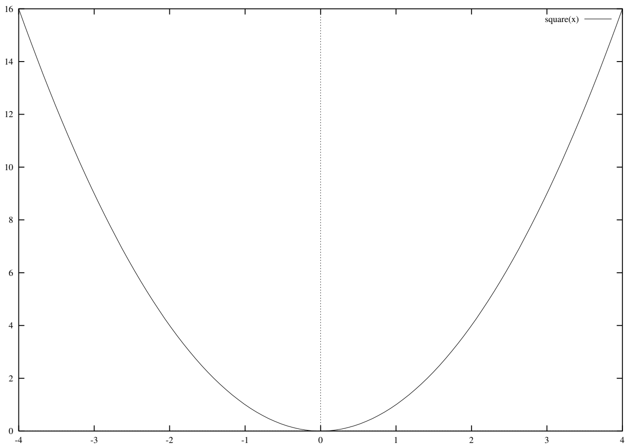
Exemples :

```
>> square(A,B).
B ~ real,
A ~ ge(0).
>> A = square(2).
A = 4.
>> A ~ square(cc(2,3)).
A ~ cc(4,9).
```

On note ici que la contrainte `sqrt(A,B)` n'est pas strictement équivalente à la contrainte `square(B,A)`, car la relation square est également définie pour les nombres négatifs.

```
>> cc(2,3) ~ square(B).
B ~ cc(-`<1.732051`,`<1.732051`).
>> set_prolog_flag(interval_mode,union).
true.
>> cc(2,3) ~ square(B).
B ~ cc(-`<1.732051`,-`>1.4142135`)u cc(`>1.4142135`,`<1.732051`).
>> A ~ square(cc(2,3)) n int .
A ~ 4 u 5 u 6 u 7 u 8 u 9.
>> A ~ square(cc(2,3) n int).
A ~ 4 u 9.
>> set_prolog_flag(interval_mode,simple).
true.
```

Graphe :



Voir également : exp/2, ln/2, log/2, power/3, root/3, sqrt/2.

tan/2 — Tangente

Définition : **tan/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid b \bmod \pi \neq \frac{\pi}{2} \text{ et } a = \tan b\}$.

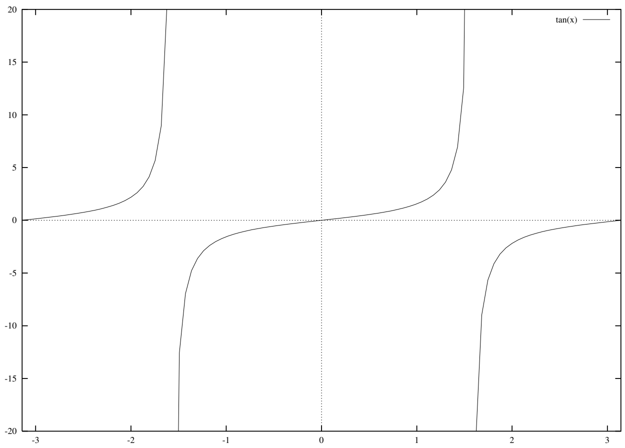
Description : **tan/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique et sa tangente.

Exemples :

```
>> tan(A, B).
B ~ real,
A ~ real.
>> tan(0, B).
B ~ real.
>> tan(1, B).
B ~ real.
>> tan(A, 0).
A = 0.
>> tan(A, pi./.4).
A ~ oo('<0.9999999','>1.0000001').
>> tan(A, pi./.2).
A ~ real.
>> tan(A, A).
A ~ real.
```

Note : Voir **cos/2** pour des remarques importantes concernant l'utilisations de relations associées à des fonctions périodiques en mode unions d'intervalles.

Graphe :



Voir également : arccos/2, arcsin/2, arctan/2, cos/2, cot/2, pi/2, sin/2.

tanh/2 — Tangente hyperbolique

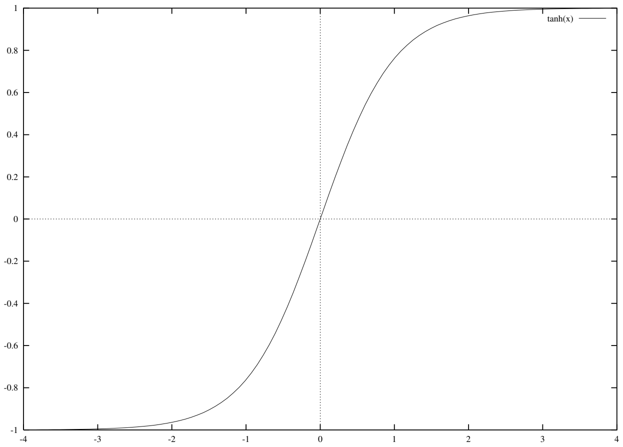
Définition : **tanh/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a = \tanh b\}$.

Description : **tanh/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre une valeur numérique et sa tangente hyperbolique.

Exemples :

```
>> tanh(A, B).
B ~ real,
A ~ oo(-1,1).
>> tanh(0, B).
B = 0.
>> tanh(A, 0).
A = 0.
>> tanh(A, 1).
A ~ cc(`>0.7615941`,`<0.7615942`).
>> tanh(A, 2).
A ~ cc(`>0.9640275`,`<0.9640276`).
>> tanh(A, 10).
A ~ cc(`>0.9999999`,1).
```

Graphe :



Voir également : cosh/2, coth/2, sinh/2.

times/3
.*. — Multiplication

Définition : **times/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid a = b \times c\}$.

Description : **times/3** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre deux nombres et leur produit.

Autre notation : **times**(a, b, c) s'écrit aussi $a = b$ **.*.** c.

Exemples :

```
>> times(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> A ~ times(2, 3).
A = 6.
>> A ~ cc(1,2) .*. cc(1,3).
A ~ cc(1,6).
>> A ~ B .*. 0 .
A = 0,
B ~ real.
>> A ~ B .*. 1 .
A = B,
B ~ real.
>> times(cc(1,2), cc(-1,1), A) .
A ~ real.
>> set_prolog_flag(interval_mode,union).
true.
>> times(cc(1,2), cc(-1,1), A) .
A ~ le(-1)u ge(1).
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : Quand deux des arguments de **times/3** sont réduits à des singletons, le troisième est calculé de manière exacte. Nous avons donc :

```
>> 1 ~ 3 .*. A .
A ~ 1/3.
```

Il existe deux exceptions notables à cette règle :

```
>> times(0,0,A).
A ~ real.
>> times(0,A,0).
A ~ real.
```

Voir également : div/3, minus/3, plus/3, timeslin/3, uminus/2, uplus/2.

timeslin/3

$*$ Multiplication

Définition : **timeslin/3** définit la relation $\{(a, b, c) \in \mathbf{R}^3 \mid a = b \times c\}$.

Description : **timeslin/2** est une relation sur les réels, qui est traitée par le solveur linéaire. Elle établit une relation entre deux nombres et leur produit.

Autre notation : **timeslin**(a, b, c) s'écrit aussi $a = b * c$.

Exemples :

```
>> timeslin(A, B, C).
C ~ real,
B ~ real,
A ~ real.
>> A ~ timeslin(2, 3).
A = 6.
>> A ~ B * 0 .
A = 0,
B ~ real.
>> A ~ B * 1 .
A = B,
B ~ real.
```

Note : **timeslin/3** est une relation du solveur linéaire. Or, il est possible de l'utiliser pour poser des contraintes non linéaires, comme par exemple dans la requête suivante :

```
>> X * X = -1 .
X ~ real.
```

Comme on le voit, la contrainte n'est pas prise en compte. En fait, elle est retardée jusqu'à ce qu'elle devienne linéaire, c'est-à-dire jusqu'à ce qu'au moins un des arguments de la multiplication soit connu. Dans l'exemple suivant, on obtient un échec dans la deuxième requête dès que la valeur de `T` est connue :

```
>> X = Y*T, X = Y*T + 1 .
T ~ real,
Y ~ real,
X ~ real.
>> X = Y*T, X = Y*T + 1, T = 2 .
false.
```

Les mêmes remarques s'appliquent à la relation **divlin/2**.

Voir également : divlin/3, minuslin/3, pluslin/3, times/3, uminuslin/2, uplusin/2.

tree/1 — Etre un arbre

Définition : **tree/1** définit la relation $\{a \in \mathbf{A} \mid a$ n'est assujetti à aucune condition$\}$.

Description : **tree/1** est une relation générale sur les arbres, qui est vérifiée pour tout arbre. Cette contrainte n'échoue jamais.

tree/1 est également le pseudo-terme le plus général, qui est très souvent utilisé dans les réponses de Prolog IV pour indiquer qu'aucune restriction n'a été effectuée sur le domaine d'une variable.

Exemples :

```
>> tree(A).
A ~ tree.
>> tree(1).
true.
>> tree(f(X)).
X ~ tree.
```

Voir également : ntree/1.

u/3 — Union

Définition : **u/3** définit la relation $\{(a, b, c) \in \mathbf{A}^3 \mid a = b$ ou $a = c\}$.

Description : **u/3** est une relation générale sur les arbres. Une contrainte $\mathbf{u}(a, b, c)$ est vérifiée si son premier argument a est égal soit à son deuxième b, soit à son troisième c.

Le nom de **u/3** a été choisi en référence au symbole de l'union des ensembles. C'est en effet l'usage le plus courant de cette relation, au niveau des pseudo-termes. Se reporter à la note après les exemples pour plus d'explications à ce sujet.

Autre notation : $\mathbf{u}(a, b, c)$ s'écrit aussi $a = b$ **u** c.

Exemples :

```
>> u(A, B, C).
C ~ tree,
B ~ tree,
A ~ tree.
>> A ~ u(1,2).
A ~ cc(1,2).
>> u(A, cc(1,2), cc(3,4)).
A ~ cc(1,4).
>> X ~ identifier u real .
X ~ leaf.
>> set_prolog_flag(interval_mode,union).
true.
>> A ~ u(1,2).
A ~ 1 u 2.
>> u(A, cc(1,2), cc(3,4)).
A ~ cc(1,2)u cc(3,4).
>> set_prolog_flag(interval_mode,simple).
true.
```

Note : Attention, la relation «u» n'est pas une relation ensembliste, et n'effectue donc pas toutes les propagations qu'on attendrait d'une union. C'est simplement un moyen élégant de noter des disjonctions :

```
>> A ~ cc(1,2), A ~ B u C .
A ~ cc(1,2),
C ~ tree,
B ~ tree.
```

La requête ci-dessus est équivalente au système d'équations suivant :

$$\begin{cases} 1 \leq a \leq 2 \\ (a = b) \vee (a = c) \end{cases}$$

On voit bien que ce système ne contraint pas les valeurs de b et c, à cause de la disjonction. A l'opposé, une telle contrainte peut faire des déductions qui n'ont rien à voir avec l'union des ensembles :

```
>> A ~ cc(1,5), B ~ cc(-10,10), A ~ B u cc(7,10).
A = B,
B ~ cc(1,5).
```

Voir également : n/3.

uminus/2

.−. Moins unaire

Définition : **uminus/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a = -b\}$.

Description : **uminus/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre un nombre et son opposé.

Autre notation : **uminus**(a, b) s'écrit aussi $a =$ **.−.** b.

Exemples :

```
>> uminus(A, B).
B ~ real,
A ~ real.
>> A ~ uminus(-1) .
A = 1.
>> A ~ .-. 2.
A = -2.
>> A ~ .-. cc(1,2).
A ~ cc(-2,-1).
>> A ~ le(-3) n ( .-. cc(1,5)).
A ~ cc(-5,-3).
>> A ~ .-. A .
A ~ real.
```

Note : Quand un des arguments de **uminus/2** est réduit à un singleton, le deuxième est calculé de manière exacte. Nous avons donc :

```
>> A ~ .-. 1/3 .
A ~ -1/3.
```

Voir également : div/3, minus/3, plus/3, times/3, uminuslin/2, uplus/2.

uminuslin/2

− ——— Moins unaire

Définition : **uminuslin/2** définit la relation $\{(a,b) \in \mathbf{R}^2 \mid a = \text{-}b\}$.

Description : **uminuslin/2** est une relation sur les réels, qui est traitée par le solveur linéaire. Elle établit une relation entre un nombre et son opposé.

Autre notation : **uminuslin**(a,b) s'écrit aussi $a = -\ b$.

Exemples :

```
>> uminuslin(A, B).
B ~ real,
A ~ real.
>> A ~ uminuslin(2).
A = -2.
>> A ~ -A .
A = 0.
```

Voir également : divlin/3, minuslin/3, pluslin/3, timeslin/3, uminus/2, upluslin/2.

uplus/2

.+. ——— Plus unaire

Définition : **uplus/2** définit la relation $\{(a,b) \in \mathbf{R}^2 \mid a = +b\}$.

Description : **uplus/2** est une relation sur les réels, qui est traitée par le solveur sur les intervalles. Elle établit une relation entre deux nombres réels égaux. C'est donc l'identité sur les nombres réels.

Autre notation : **uplus**(a,b,c) s'écrit aussi $a = b$ **.+.** c.

Exemples :

```
>> uplus(A, B).
A = B,
B ~ real.
>> A ~ uplus(1).
A = 1.
>> A ~ cc(1,4), B ~ cc(3,5), A ~ .+. B .
A = B,
B ~ cc(3,4).
```

Note : Quand un des arguments de **uplus/2** est réduit à un singleton, le deuxième est calculé de manière exacte. Nous avons donc :

```
>> A ~ .+. 1/3 .
A ~ 1/3.
```

Voir également : div/3, minus/3, plus/3, times/3, uminus/2, upluslin/2.

upluslin/2

+ — Plus unaire

Définition : **upluslin/2** définit la relation $\{(a, b) \in \mathbf{R}^2 \mid a = +b\}$.

Description : **upluslin/2** est une relation sur les réels, qui est traitée par le solveur linéaire. Elle établit une relation entre deux nombres réels égaux. C'est donc l'identité sur les nombres réels.

Autre notation : **upluslin**(a, b, c) s'écrit aussi $a = b$ **+** c.

Exemples :

```
>> upluslin(A, B).
A = B,
B ~ real.
>> A ~ upluslin(1).
A = 1.
>> +A ~ -A .
A = 0.
```

Voir également : divlin/3, minuslin/3, pluslin/3, timeslin/3, uminuslin/2, uplus/2.

xor/2 — Ou exclusif

Définition : **xor/2** définit la relation $\{(a, b) \in \mathbf{B}^2 \mid a \neq b\}$.

Description : **xor/2** est une relation sur les booléens, qui est traitée par le solveur sur les intervalles. Une contrainte **xor**(a, b) réussit si ses deux arguments sont des booléens distincts (ce qui revient à la définition du ou exclusif).

Cette contrainte est normalement utilisée pour construire une combinaison d'autres contraintes exprimées par des pseudo-opérations produisant des booléens.

Autre notation : **xor**(a, b) s'écrit aussi a **xor** b.

Exemples :

```
>> xor(A, B).
B ~ cc(0,1),
A ~ cc(0,1).
>> xor(0, B).
B = 1.
>> xor(1, B).
B = 0.
>> xor(A, 0).
A = 1.
>> xor(A, 1).
A = 0.
>> xor(A, A).
A ~ cc(0,1).
>> xor(A, A), boolsplit([A]).
false.
```

Voir également : and/2, bxor/3, equiv/2, impl/2, not/1, or/2.