2

Les Bases de Prolog IV

2.1 Introduction

LE PREMIER PRINCIPE d'un langage tel que Prolog IV est d'être bâti sur une structure mathématique expressive dans laquelle on résout des contraintes complexes portant sur des éléments inconnus.

Le deuxième principe est d'avoir la liberté d'enrichir cette structure de nouvelles relations et de continuer à résoudre des contraintes dans la nouvelle structure obtenue.

Ces principes ne pouvant être respectés qu'à des degrés divers, un troisième principe de ce langage est d'être suffisamment bien conçu pour que ses insuffisances soient clairement perçues et donc mieux maîtrisées par l'utilisateur.

Explicitons ces trois principes.

Premier principe : une structure de base expressive

A la base de notre langage se trouve la notion de contrainte et pour entrer dans le vif du sujet en voici une, peu commune, formulée avec des notations empruntées à la logique du premier ordre et plus généralement aux mathématiques :

$$\exists u \exists v \exists w \exists x \begin{pmatrix} \quad y \leq 5 \\ \wedge\ v_1 = \cos v_4 \\ \wedge\ \text{size}(u) = 3 \\ \wedge\ \text{size}(v) = 10 \\ \wedge\ u \bullet v = v \bullet w \\ \wedge\ y \geq 2 + (3 \times x) \\ \wedge\ x = (74 > \lfloor 100 \times v_1 \rfloor) \end{pmatrix} \tag{2.1}$$

La seule variable qui a des occurrences libres dans cette formule est y et donc cette contrainte exprime une propriété de sa valeur. Trouver cette valeur, c'est le rôle de Prolog IV.

Cet y doit donc être plus grand ou égal 5 et de plus il faut qu'il existe des vecteurs u, v, w et un réel x tels que les 1er et 4ème éléments du vecteur v soient égaux, les longueurs de u et v soient 3 et 10 le vecteur obtenu par concaténation de u avec v soit le même que celui obtenu par concaténation de v avec w, le réel y soit plus grand ou égal à $2 + 3x$ avec x contraint à être égal à 1 ou 0 suivant que 74 est ou n'est pas plus grand à $\lfloor 100 \times v_1 \rfloor$, c'est-à-dire la partie entière de 100 fois le premier élément du vecteur v. Ouf ! Il n'y a qu'un seul y qui satisfait à toute cette contrainte et c'est

$y = 5.$

Comment se présentent les choses en Prolog IV ? On lance le système qui signale qu'il attend une requête en affichant un double chevron fermant, on lui fournit la contrainte précédente dans une syntaxe adéquate, il la résout, fournit la réponse et affiche à nouveau le double chevron pour signaler qu'il est prêt à recommencer. Cela donne :

```
>> U ex V ex W ex X ex
   le(Y,5),
   V:1 = cos(V:4),
   size(U) = 3,
   size(V) = 10,
   U o V = V o W,
   ge(Y,2.+.(3.*.X)),
   X = bgt(74,floor(100.*.V:1)).

Y = 5.
```

Rapidement quelques mots sur la syntaxe utilisée. Prolog IV se voulant conforme syntaxiquement à la norme Prolog ISO [1], toute expression se doit d'être un « terme » au sens de cette norme [2]. Les termes au sens de la norme n'étant pas les termes comme on les connaît en logique, nous appellerons les premiers, termes-ISO. Ces termes-ISO sont essentiellement de l'une des trois formes

1. variable,
2. nombre ou identificateur,
3. identificateur(terme-ISO,...,terme-ISO),

avec des variations infixées, des notations spécifiques aux listes et la convention que les noms de variables commencent par des majuscules. Ces variations permettent d'écrire ici

`X:I`	pour	`index(X,I)`,
`X o Y`	pour	`conc(X,Y)`,
`X.+.Y`	pour	`plus(X,Y)`,
`X.*.Y`	pour	`times(X,Y)`,

les relations `index/3`, `conc/3`, `plus/3`, `times/3` étant utilisées avec des notations fonctionnelles.

Passons à la sémantique de la contrainte. Le connecteur logique $\wedge$, noté ici par une virgule et la quantification existentielle $\exists x p$ notée ici x `ex` p ont leurs sens habituels, nous y reviendrons plus tard. Les variables représentent des élément inconnus pris dans un certain domaine et les autres symboles des relations et des opérations bien précises dans ce même domaine.

Ces opérations et relations dans un domaine précis font partie d'une structure plus complète, choisie une bonne fois pour toute, la structure de base π_4. Ainsi que le laisse pressentir notre exemple de contrainte, il s'agit d'une structure

1. *ISO/IEC 13211-1*, Information Technology, Programming Languages, Prolog, Part 1: General Core, 1995.

2. A la page 5 du livre de Pierre Deransart, AbdelAli Ed-Dbali et Laurent Cervoni, *Prolog: The Standard*, Springer-Verlag, 1996, on peut lire : « In Standard Prolog a unique data structure is used: terms »

particulièrement riche. Nous avons voulu à la fois pouvoir manipuler (1) des symboles, (2) des nombres et (3) des objets complexes construits à partir de ceux-ci. En ce qui concerne la partie numérique, nous nous sommes montré particulièrement ambitieux en introduisant une centaine de relations portant sur l'ensemble **R** des nombres réels (au sens mathématique), sur l'ensemble **Q** des nombres rationnels (les fractions), sur l'ensemble **Z** des nombres entiers relatifs et sur l'ensemble **B** des valeurs booléennes 0 et 1, et ceci, tout en respectant les inclusions $\mathbf{B} \subset \mathbf{Z} \subset \mathbf{Q} \subset \mathbf{R}$.

Deuxième principe : Une structure enrichissable

En écrivant un programme Prolog IV on définit de nouvelles relations qui complètent la structure de base π_4. Cet enrichissement est plus d'ordre quantitatif que d'ordre qualitatif : essentiellement les nouvelles relations permettent d'exprimer en une seule contrainte la conjonction d'une multitude de contraintes atomiques exprimables dans π_4. Par exemple le programme

$$\forall l \forall x \left(\text{sigma}(x,l) \equiv \left(\begin{array}{c} l = \texttt{[]} \wedge x = 1 \\ \vee \\ \exists l' \exists y' \exists x' \left(\begin{array}{l} \quad l = \texttt{.}(y',l') \\ \wedge \ \text{plus}(y',x',x) \\ \wedge \ \text{sigma}(x',l') \end{array} \right) \end{array} \right) \right) \tag{2.2}$$

permet de définir le lien qui existe entre une liste et la somme de ses éléments par une conjonction de contraintes portant sur le lien qui existe entre deux nombres et leur somme. Des conventions syntaxiques permettent d'écrire le programme sous forme d'un paquet de clauses plus habituel dans le monde Prologien :

```
sigma(0,[]).
sigma(X,.(Yp,Lp)) :- plus(X,Yp,Xp), sigma(Xp,Lp).
```

Troisième principe : les insuffisances sont explicites

Prolog IV ne peut être parfait, il est hors de question de pouvoir résoudre parfaitement des contraintes dans une structure aussi riche que π_4 et il est encore moins question de laisser un usager enrichir π_4 sans aucune discipline et s'attendre à ce que miraculeusement la machine résolve toutes les contraintes exprimables dans cette nouvelle structure.

Ce que nous avons voulu c'est que Prolog IV soit parfait dans son imperfection ou plus exactement que ses imperfections soient explicites. Ceci passe par un fondement logique bien précis que l'on peut résumer en trois points.

(1) Syntaxiquement le langage Prolog IV est vu comme un langage du premier ordre avec égalité et les contraintes comme des formules positives, quantifiées existentiellement (partiellement).

(2) Sémantiquement et en l'absence de tout programme, le langage Prolog IV est vu comme une théorie $\mathcal{T}(\pi_4)$ du premier ordre avec égalité c'est-à-dire un ensemble d'axiomes exprimant des propriétés de la structure π_4. C'est uniquement ces propriétés qui sont utilisées pour résoudre des contraintes, mais ces propriétés sont utilisées au mieux. Lorsque le programmeur donne une contrainte p à résoudre à Prolog IV il s'attend à ce que le système lui retourne une contrainte q équivalente à p dans la structure π_4 c'est-à-dire ayant les mêmes solutions que p dans la structure π_4 mais dont les solutions sont plus

« visibles ». Notamment il aimerait que Prolog IV retourne la contrainte *false* si p n'a pas de solutions dans π_4. En fait la contrainte q retournée sera non seulement équivalente à p dans la structure π_4 mais dans toutes les structures ϕ ayant les propriétés $\mathcal{T}(\pi_4)$, ce qui s'écrit

$$\mathcal{T}(\pi_4) \quad \models \quad p \equiv q$$

et ne sera *false* que si, non seulement elle n'a pas de solutions dans la structure π_4, mais si en plus elle n'a de solutions dans aucune des autres structures ϕ qui ont les propriétés $\mathcal{T}(\pi_4)$, c'est-à-dire si

$$\mathcal{T}(\pi_4) \quad \models \quad p \equiv \mathit{false}$$

(3) Sémantiquement et en présence d'un programme $\mathcal{P}$ le langage Prolog IV est vu comme la théorie $\mathcal{T}(\pi_4) \cup \mathcal{P}$. Le programme $\mathcal{P}$ est donc un ensemble d'axiomes supplémentaires que l'on ajoute à $\mathcal{T}(\pi_4)$. Chacun de ces axiomes définit la propriété essentielle d'une nouvelle relation nommée r et est de la forme

$$\forall x_1 \ldots \forall x_n \; (r \, x_1 \ldots x_n \equiv p).$$

où p est une formule n'ayant pas d'autres occurrences libres de variables que celles $x_1, \ldots, x_n$. Bien entendu tout est limpide si p ne fait que contenir des relations et des opération de π_4, mais dans le cas ou p fait intervenir d'autres nouvelles relations et surtout la relation r elle-même, on a à faire à des définitions récursives. La résolution d'une contrainte p doit alors être vue comme le calcul d'une contrainte équivalent à q dans la théorie $\mathcal{T}(\pi_4) \cup \mathcal{P}$ c'est-à-dire telle que

$$\mathcal{T}(\pi_4) \cup \mathcal{P} \quad \models \quad p \equiv q$$

Plan de l'exposé

Cette introduction constitue la partie 1 de notre exposé. La partie 2 est consacrée à des définitions purement logiques et notamment à ce que nous entendons par une contrainte, une structure, etc. C'est un petit cours de logique taillé sur mesure. La partie 3 décrit en détail la structure choisie π_4. La partie 4 est consacrée à l'axiomatisation de cette structure. La partie 5 décrit la notion de programme et la machinerie Prolog IV. Enfin la partie 6 donne des exemples divers de programmes Prolog IV.

2.2 Syntaxe et sémantique

2.2.1 Objets sémantiques

Les seuls objets sémantiques dont nous aurons besoin sont les relations et les opérations. Quelques définitions précises et quelques commentaires s'imposent ici. Soit D un ensemble et n un entier non négatif.

Une *relation* ρ est un sous-ensemble quelconque de D^n. On dit que ρ est une relation d'*arité* n *dans* D. Du fait qu'une relation est vue comme un ensemble de n-uplets, tous de même longueur, l'ensemble vide $\emptyset$ est une relation qui a pour arité n'importe quelle entier non négatif.

Une *opération* φ est une application de D^n dans D. On dit que φ est une opération d'*arité* n *dans* D. Il est commode de considérer les opérations comme des cas particuliers de relations ; pour ceci on confond l'opération n-aire φ avec la relation $n+1$-aire

$\{(a_0, a_1, \ldots, a_n) \in D^{n+1} \mid a_0 = \varphi(a_1, \ldots, a_n)\}$.

Une opération n-aire est donc une relation $n+1$-aire ρ qui a la propriété :

> quel que soit $(a_1, \ldots, a_n) \in D^n$,
> il existe un et un seul $a_0 \in D$ tel que $(a_0, a_1, \ldots, a_n) \in \rho$.

Une relation $n+1$-aire ρ qui a la propriété affaiblie :

> quel que soit $(a_1, \ldots, a_n) \in D^n$,
> il existe au plus un $a_0 \in D$ tel que $(a_0, a_1, \ldots, a_n) \in \rho$.

sera appelée *opération partielle* ou *pseudo-opération*.

2.2.2 Objets syntaxiques

Les seuls objets syntaxiques dont nous aurons besoin sont les termes et les formules. Les termes servent à représenter des éléments d'un domaine et les formules servent à représenter des propriétés de ces éléments.

On se donne une bonne fois pour toute un ensemble universel infini V de *symboles de variables*[3] pour représenter des individus pris dans un domaine. On se donne en plus des ensembles F, R de symboles d'opérations et de relations. A chacun de ces symboles est associé un entier positif ou nul, son *arité*.

Les *termes* sont des expressions de l'une des deux formes :

1	x	x
2	$f\, t_1 \ldots t_n$	$f\,(t_1, \ldots, t_n)$

avec $x \in V$, $f \in F$ et d'arité n et les t_i des termes plus courts que ceux que l'on est train de définir. Les notations sont celles qui sont le plus couramment utilisées. Dans la colonne de droite nous donnons un aperçu de la notation Prolog IV de chaque forme.

3. Rappelons que la grande trouvaille de la norme ISO, à laquelle nous nous conformons dans la syntaxe de Prolog IV, est de noter ces variables par des identificateurs commençant par une lettre majuscule !

Les *formules* sont des expressions de l'une des 11 formes

$$
\begin{array}{lll}
1 & \textit{true} & \texttt{true} \\
2 & \textit{false} & \texttt{false} \\
3 & t_1 \doteq t_2, & t_1 \;\texttt{=}\; t_2 \\
4 & r\, t_1 \ldots t_n & r\,\texttt{(}t_1\texttt{,}\ldots\texttt{,}t_n\texttt{)} \\
 & & \\
5 & (p \wedge q) & \texttt{(}p \;\texttt{,}\; q\texttt{)} \\
6 & (p \vee q) & \texttt{(}p \;\texttt{;}\; q\texttt{)} \\
7 & \exists x p & x \;\texttt{ex}\; p \\
 & & \\
8 & \neg p & \\
9 & (p \Rightarrow q) & \\
10 & (p \equiv q) & \\
11 & \forall x p &
\end{array}
\tag{2.3}
$$

où les t_i sont des termes, r un symbole de R d'arité n et p, q des formules plus courtes que celles que l'on est train de définir. Dans la colonne de droite nous donnons toujours, pour chaque forme, un aperçu de la notation Prolog IV, quand elle existe. Une occurrence de variable x dans une formule q est dite *liée* si elle se produit à l'intérieur d'une formule de la forme 7 ou 11 figurant dans q. Une occurrence qui n'est pas liée est dite *libre*. Une formule qui ne contient aucune occurrence libre de variable est une *proposition*.

Nous réservons le mot *contrainte* pour désigner des formules existentielles positives, c'est-à-dire ne faisant intervenir que les 7 première formes. Une contrainte ou formule *atomique* est une formule de la forme 1, 2, 3 ou 4.

2.2.3 Liens entre objets syntaxiques et objets sémantiques

Si l'on donne une signification à chaque symbole qui figure dans un terme ou dans une formule on pourra systématiquement étendre ces significations pour attribuer une signification à tout le terme où toute la formule. C'est cette extension de signification que nous allons préciser. On appelle *interprétation* I toute application qui fait correspondre

- à chaque variable $x \in V$, un élément $I(x)$ d'un ensemble D,
- à chaque symbole d'opération $f \in F$ d'arité n, une opération n-aire $I(f)$ dans le même ensemble D,
- à chaque symbole de relation $r \in F$ d'arité n, une relation n-aire $I(r)$ dans le même ensemble D.

L'ensemble commun D est appelé domaine de I et sera noté $\mathrm{dom}(I)$, l'ensemble F des symboles d'opérations interprétés par I est noté $\mathrm{op}(I)$ et l'ensemble R des symboles de relations interprété est noté $\mathrm{rel}(I)$

L'interprétation I est donc l'objet mathématique qui attribue une signification à chaque symbole de $V \cup F \cup R$. Cette interprétation ce prolonge en une application I^* qui attribue à chaque terme t construit sur $V \cup F$ un élément $I^*(t)$ de $\mathrm{dom}(I)$ et à chaque formule p construit sur $V \cup F \cup R$ une valeur de vérité $I^*(p)$ prise dans $\{\mathrm{vrai}, \mathrm{faux}\}$ comme suit.

L'application I^* est définie récursivement sur les deux formes de termes par :

$$
\begin{array}{llll}
1 & I^*(x) & = & I(x) \\
2 & I^*(f\, t_1 \ldots t_n) & = & I(f)(I^*(t_1), \ldots, I^*(t_n))
\end{array}
$$

et sur les 11 formes de formulas par :

1	$I^*(\mathit{true})$	$=$	vrai
2	$I^*(\mathit{false})$	$=$	faux
3	$I^*(t_1 \doteq t_2)$	$=$	$\begin{cases} \text{vrai} & \text{si } I^*(t_1) = I^*(t_2), \\ \text{faux} & \text{sinon} \end{cases}$
4	$I^*(r\, t_1 \ldots t_n)$	$=$	$\begin{cases} \text{vrai} & \text{si } (I^*(t_1), \ldots, I^*(t_n)) \in I(r) \\ \text{faux} & \text{sinon} \end{cases}$
5	$I^*(p \wedge q)$	$=$	$\begin{cases} \text{vrai} & \text{si } I^*(p) = \text{vrai et } I^*(q) = \text{vrai} \\ \text{faux} & \text{sinon} \end{cases}$
6	$I^*(p \vee q)$	$=$	$\begin{cases} \text{vrai} & \text{si } I^*(p) = \text{vrai ou } I^*(q) = \text{vrai} \\ \text{faux} & \text{sinon} \end{cases}$
7	$I^*(\exists x p)$	$=$	$\begin{cases} \text{vrai} & \text{s'il existe } J \in \text{next}(I, x) \text{ avec } J^*(p) = \text{vrai}, \\ \text{faux} & \text{sinon} \end{cases}$
8	$I^*(\neg p)$	$=$	$\begin{cases} \text{vrai} & \text{si } I^*(p) = \text{faux} \\ \text{faux} & \text{sinon} \end{cases}$
9	$I^*(p \Rightarrow q)$	$=$	$\begin{cases} \text{faux} & \text{si } I^*(p) = \text{vrai et } I^*(q) = \text{faux} \\ \text{vrai} & \text{sinon} \end{cases}$
10	$I^*(p \equiv q)$	$=$	$\begin{cases} \text{vrai} & \text{si } I^*(p) = I^*(q) \\ \text{faux} & \text{sinon} \end{cases}$
11	$I^*(\forall x p)$	$=$	$\begin{cases} \text{vrai} & \text{si pour tout } J \in \text{next}(I, x) \text{ on a } J^*(p) = \text{vrai}, \\ \text{faux} & \text{sinon} \end{cases}$

en convenant que $\text{next}(I, x)$ est l'ensemble d'interprétations composé de l'interprétation I et de toutes les interprétations J qui ne diffèrent de I que par le fait que $I(x) \neq J(x)$.

Une remarque importante s'impose : la valeur $I^\star(p)$ de la formule p ne dépend pas des valeurs $I(x)$ des variables x sans occurrences libres dans p. Donc, si p est une proposition, c'est-à-dire une contrainte sans occurrences libres de variables, $I^\star(p)$ ne dépend pas de la façon d'interpréter les variables.

Si $I^\star(p) = $ vrai on dit que p *est vraie dans* l'interprétation I mais aussi que I *satisfait* p. Si $\mathcal{T}$ est un ensemble de formules et p une formule on dira que p *est vrai dans* $\mathcal{T}$ et on écrira

$$\mathcal{T} \models p$$

si toute interprétation qui satisfait chaque formule de $\mathcal{T}$ satisfait aussi p.

Dans beaucoup de raisonnements on est amené à faire varier l'interprétation des symboles de variables tout en gardant la même interprétation pour les autres symboles. Il est alors intéressant de décomposer l'interprétation I en un couple (ϕ, σ) d'applications définies comme suit.

$$I(s) = \begin{cases} \phi(s), & \text{si } s \in F \cup R, \\ \sigma(s), & \text{si } s \in V. \end{cases}$$

L'application σ est une *affectation de variable* c'est-à-dire une application de V dans un ensemble que l'on notera $\text{dom}(V)$. L'application ϕ est une *structure* c'est-à-dire une famille d'opérations $\phi(f)$ et de relations $\phi(r)$ sur un domaine $\text{dom}(\phi)$ indicés par des symboles d'opérations f pris dans un ensemble $\text{op}(\phi)$ et par des symboles de relations pris dans un ensemble de symboles $\text{rel}(\phi)$.

Lorsque l'interprétation I satisfait la formule p et que I est décomposé dans le couple (ϕ, σ) on peut utiliser un jargon plus mathématique que logique et dire que l'affectation σ *est solution* de la formule p *dans* la structure ϕ.

2.2.4 Notation fonctionnelle de relations

Afin d'utiliser des notations fonctionnelles pour représenter des individus qui sont dans certaines relations, on assimile la relation $(n+1)$-aire ρ sur l'ensemble D à l'application suivante de D^n dans l'ensemble $\mathcal{P}(D)$ des parties de D

$$(a_1, \ldots, a_n) \mapsto \{a_0 \mid (a_0, a_1, \ldots, a_n) \in \rho\}.$$

On construit alors des expressions qui, contrairement aux termes, ne représentent pas des éléments de D mais des ensembles d'éléments de D. Ces nouvelles expressions que nous appellerons *pseudo-termes* sont de l'une des deux formes :

1	$r\, s_1 \ldots s_n$	`r(s1, ..., sn)`
2	$f\, s_1 \ldots s_n$	`f(s1, ..., sn)`

avec $x \in V$, $f \in F$ et d'arité n, $r \in R$ et d'arité $n+1$, les s_i des termes ou pseudo-termes plus courts que ceux que l'on est train de définir et l'expression de la forme 2 contenant au moins un pseudo-terme. Ici aussi la colonne de droite donne un aperçu des notations Prolog IV. On donne alors une définition généralisée des formules logiques en complétant leurs 11 formes, de la page 66, par les deux formes

3bis	$s_1 \sim s_2$	`s1 ~ s2`
4bis	$r\, s_1 \ldots s_n$	`r(s1, ..., sn)`

où les s_i sont des termes ou des pseudo-termes et où r appartient à R et est d'arité n. Bien entendu les formules atomiques sont maintenant les formules de la forme 1, 2, 3, 3bis, 4 ou 4bis et les notions d'occurrences liées ou libres de variables restent inchangées.

Le signe $\sim$ se lit *est compatible avec* et, si a et b sont des éléments ou des sous-ensembles de D, a la signification suivante :

$$a \sim b \overset{def}{\equiv} \begin{cases} a = b, & \text{si } a \text{ et } b \text{ sont des éléments de } D, \\ a \in b & \text{si } a \text{ est un élément et } b \text{ un sous-ensemble de } D, \\ a \ni b & \text{si } a \text{ est un sous-ensemble et } b \text{ un élément de } D, \\ a \cap b \neq \emptyset & \text{si } a \text{ et } b \text{ sont des sous-ensembles de } D. \end{cases}$$

La contrainte atomique $r\, s_1 \ldots s_n$ exprime qu'il existe des individus $a_1, \ldots, a_n$ qui sont dans la relation r et qui sont compatibles avec $s_1, \ldots, s_n$.

Plus précisément si on se donne une interprétation I, la valeur de vérité $I^\star(p)$ d'une formule généralisée de la forme 3bis ou 4bis est définie par

$$3bis \quad I^*(s_1 \sim s_2) \quad = \quad \begin{cases} \textit{true}, & \text{si } I^*(s_1) \sim I^*(s_2) \\ \textit{false}, & \text{sinon} \end{cases}$$

$$4bis \quad I^*(r\, s_1 \ldots s_n) \quad = \quad \begin{cases} \textit{true} & \text{s'il existe des } a_i \text{ avec} \\ & a_1 \sim I^*(s_1), \ldots, a_n \sim I^*(s_n) \\ & \text{et } (a_1, \ldots, a_n) \in I(r) \\ false, & \text{sinon} \end{cases}$$

où la valeur $I^\star(s)$ d'un pseudo-terme est un ensemble défini sur les deux formes du pseudo-terme par

1	$I^*(r\,s_1 \dots s_n)$	$=$	$\begin{cases} \text{l'ensemble des } b \text{ tels qu'il existe des } a_i \text{ avec} \\ a_1 \sim I^*(s_1), \dots, a_n \sim I^*(s_n) \text{ et } (b, a_1, \dots, a_n) \in I(r) \end{cases}$
2	$I^*(f\,s_1 \dots s_n)$	$=$	$\begin{cases} \text{l'ensemble des } b \text{ tels qu'il existe des } a_i \text{ avec} \\ a_1 \sim I^*(s_1), \dots, a_n \sim I^*(s_n) \text{ et } b = I(f)(a_1, \dots, a_n). \end{cases}$

Toutes les autres notions liées aux interprétations restent inchangées.

Une formule généralisée peut toujours être transformée en une formule équivalente ne faisant intervenir ni pseudo-terme ni le signe $\sim$. Il suffit de répéter les transformations suivantes :

$$\begin{array}{lll} 1 \quad t_1 \sim t_2 & \text{devient} & t_1 = t_2 \\ 2 \quad p(r\,s_1 \dots s_n) & \text{devient} & \exists x \;\; r\,x s_1 \dots s_n \wedge p(x) \end{array} \tag{2.4}$$

où t_1 et t_2 sont des termes, où $p(r\,s_1 \dots s_n)$ est une formule atomique dans laquelle on a choisi une occurrence d'un pseudo-terme $r\,s_1 \dots s_n$ de la forme 1 et où $p(x)$ est la même formule atomique dans laquelle on a substitué une nouvelle variable x à l'occurrence choisie du pseudo-terme.

La transformation 2 permet d'éliminer tous les symboles de relations imbriqués dans des contraintes atomiques et donc d'éliminer tous les pseudo-termes. La transformation 1 permet ensuite d'éliminer tous les signes $\sim$.

Si on applique ces transformations sur la contrainte

```
X ~ [cc(1,2) u co(4,8), gt(9), real]
```

du tableau 2.3 à la page 73 on obtient, en tenant compte de ce que `cc(1,2) u co(4,8)` n'est que la variante infixée de `u(cc(1,2),co(4,8))`,

```
A ex B ex C ex D ex E ex
X = [A,D,E], u(A,B,C), cc(B,1,2), co(C,4,8),
gt(D,9), real(E)
```

Terminons cette partie par une remarque d'ordre syntaxique. L'introduction du signe $\sim$ n'était pas absolument nécessaire, on aurait très bien pu utiliser le signe « égal » en étendant son sens à celui de « compatible ». Il aurait cependant été choquant de lier par l'égalité des ensembles pouvant être distincts. Cependant, dans une contrainte de la forme $x \sim s$, où s est un pseudo-terme représentant un ensemble de un ou de zéro élément, le signe égal est plus esthétique. Pour toutes ces raisons, le compilateur Prolog IV acceptera que l'on remplace en entrée toute occurrence du signe compatible par celle du signe égal. C'est aussi ce que nous avions fait dans notre tout premier exemple de contrainte à la page 61.

2.3 La structure de base π_4

2.3.1 Domaine de la structure choisie π_4

Le domaine dom(π_4) de la structure choisie est l'ensemble des arbres dont les nœuds, qui forment un ensemble éventuellement infini, sont étiquetés

1. soit par des identificateurs Prolog-ISO (complétés d'une arité),
2. soit par des nombres réels (considérés comme des étiquettes d'arité nulle),

On ne distingue pas les arbres ayant un seul nœud de l'étiquette portée par ce nœud. Les identificateurs et les réels sont donc assimilés à des arbres d'un nœud étiquetés par des identificateurs et par de réels. Plusieurs sous-ensembles de notre domaine jouent un rôle particulier, passons les en revue.

Ensemble des nombres rationnels Ce sont les nombres réels de la forme $\pm\frac{p}{q}$ où q est un entier positif et p un entier positif ou nul. On les écrira tout simplement sous la forme `p/q` avec éventuellement un signe devant.

Ensemble des nombres rationnels décimaux Ce sont des nombres rationnels de la forme $\pm\, p \times 10^{\pm q}$, où p et q sont des entiers positifs ou nuls. On les note d'une façon classique avec éventuellement un point décimal et une partie exposant introduite par la lettre `E`. Remarquons que les nombres entiers sont un cas particulier des nombres décimaux.

Ensemble des nombres rationnels IEEE Ce sont des nombres binaires, à virgule flottante, conformes à la norme IEEE simple précision[4]. Ils sont de la forme

$$\pm\, p \times 2^q \quad \text{avec} \quad 0 \leq p \leq 2^{24}–1 \text{ et } -149 \leq q \leq 104,$$

où p et q sont des entiers. Il y en a $2n + 1$ avec $n = 2^{31} - 2^{23} - 1$, c'est-à-dire en tout et pour tout $2\,139\,095\,039$. Si on les numérote de $-n$ à n et que l'on désigne par q_i le nombre numéro i alors l'entier $|i|$ écrit en binaire sur 31 bits et précédé d'un bit pour indiquer le signe de i constitue le codage IEEE de q_i. Ce codage est sans redondances sauf le nombre 0 qui est codé avec le signe plus ou le signe moins.

Les rationnels IEEE sont des cas particuliers des rationnels décimaux[5] et peuvent donc être notés sous forme décimale avec beaucoup de chiffres. Cependant afin de disposer d'une notation plus compacte on écrit `'<`x`'` et `'>`x`'` pour désigner les rationnels IEEE qui précèdent et suivent immédiatement le nombre décimal sans signe x.

4. IEEE Standard for Binary Floating-Point Arithmetic, IEEE Std 754-1985, Approved March 21, 1985, Reaffirmed December 6,1990, IEEE Standards Board, approved July 26, 1985 American National Standards Institute

5. En effet si q est non négatif ce sont des entiers et donc des rationnels décimaux et si q est négatif ils sont de la forme $\pm\,(5^{|q|} \times p) \times 10^q$.

Intervalle C'est un ensemble de nombre réels de l'une des 10 formes :

1	$(-\infty, +\infty)$:	**R**	`real`
2	$[a, +\infty)$:	$\{x \in \mathbf{R} \,\vert\, a \leq x\}$,	`ge(`a`)`
3	$(a, +\infty)$:	$\{x \in \mathbf{R} \,\vert\, a < x\}$,	`gt(`a`)`
4	$(-\infty, b]$:	$\{x \in \mathbf{R} \,\vert\, x \leq b\}$,	`le(`b`)`
5	$(-\infty, b)$:	$\{x \in \mathbf{R} \,\vert\, x < b\}$,	`lt(`b`)`
6	$[a, b]$:	$\{x \in \mathbf{R} \,\vert\, a \leq x \leq b\}$,	`cc(`a`,`b`)`
7	$(a, b]$:	$\{x \in \mathbf{R} \,\vert\, a < x \leq b\}$,	`oc(`a`,`b`)`
8	$[a, b)$:	$\{x \in \mathbf{R} \,\vert\, a \leq x < b\}$,	`co(`a`,`b`)`
9	(a, b) :	$\{x \in \mathbf{R} \,\vert\, a < x < b\}$	`oo(`a`,`b`)`
10	$\emptyset$:	l'ensemble vide.	`ntree`

où a et b, les bornes inférieure et supérieure, sont des nombres réels et où la dernière colonne donne les notations Prolog IV.

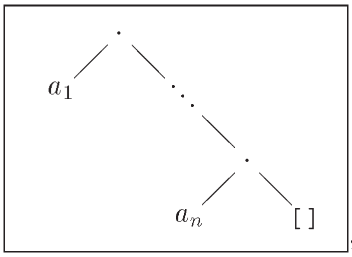
Intervalle IEEE C'est un intervalle dont les bornes inférieure a et supérieure b, si elles existent, sont des nombres rationnels IEEE. On remarquera que l'ensemble des intervalles IEEE a pour éléments l'ensemble vide et l'ensemble des réels et est fermé pour toute intersection finie ou infinie de ses éléments. Le tout dernier point découle du fait que l'ensemble des nombres rationnels IEEE est fini.

Union d'intervalles IEEE C'est un ensemble de réels de la forma $E_1 \cup \cdots \cup E_n$, où les E_i sont des intervalles IEEE.

Ensemble des arbres rationnels Ce sont les arbres qui ont un ensemble fini de sous-arbres. Rappelons qu'il existe des arbres ayant un ensemble infini de nœuds mais un nombre fini de sous-arbres (non isomorphes).

Ensemble des arbres doublement rationnels Ce sont les arbres rationnels qui ne font pas intervenir d'autres nombres que des nombres rationnels.

Ensemble des listes Ce sont des arbres a de la forme



,

où les a_i sont des arbres quelconques. Une telle liste a est notée $[a_1, \ldots, a_n]$ et on désigne par $|a|$ sa *taille* éventuellement nulle n. La concaténation de deux listes, de tailles éventuellement nulles, est définie et notée par $[a_1, \ldots, a_m] \circ [a_{m+1}, \ldots, a_n] = [a_1, \ldots, a_n]$.

2.3.2 Sous-domaines privilégiés

Tout le mécanisme de résolution approché de contraintes de Prolog IV repose sur le fait qu'à chaque variable est attribué un sous-ensemble de l'ensemble

$\mathrm{dom}(\pi_4)$ d'arbres et que constamment on essaye

1. de «réduire» ces sous-ensembles en tenant compte de leurs interactions avec chaque contrainte prise séparément et
2. de «globaliser» toute contrainte locale c'est-à-dire de la remplacer par une contrainte plus basique que l'on sait résoudre globalement.

Ces sous-ensembles ne sont pas quelconques, ce sont les *sous-domaines privilégiés*. Ils sont obtenus par intersections multiples des 15 formes de bases du tableau 2.1 qui représentent des sous-ensembles pas toujours disjoints. Une

TAB. 2.1 – *Sous-domaines privilégiés*

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

variante possible pour les formes 7 et 12 est donnée dans le tableau 2.2. Les modalités pour utiliser cette variante sont décrites dans la partie 9 de ce document : environnement général.

TAB. 2.2 – *Variante des sous-domaines privilégiés*

	Formes de base
7	pour chaque n-uplet $A_1, \ldots, A_n$ d'unions A_i d'intervalles IEEE, l'ensemble des listes de la forme $[a_1, \ldots, a_n]$
12	pour chaque union A d'intervalle IEEE, l'ensemble A

Chaque sous-domaine privilégié est de catégorie (a), de catégorie (b) ou à la fois de catégorie (a) et (b). Les sous-domaines de catégorie (a) interviennent dans le processus de réduction des sous-domaines et ceux de catégorie (b) dans le processus de globalisation des contraintes.

– Sont de catégorie (a) tous les sous-domaines privilégiés, à l'exception de ceux qui sont de la forme 14, lorsque a est un nombre, qui n'est pas un rationnel IEEE.
– Sont de catégorie (b) tous les sous-domaines privilégiés, à l'exception de ceux qui sont de la forme 7 ou 12, lorsque un A_i ou A est autre que

l'ensemble vide ou l'ensemble **R** de tous les réels.

On remarquera que l'ensemble des sous-domaines privilégiés de catégorie (a) est fermé par intersections finis ou infinies et a pour élément l'ensemble de tous les arbres. De ceci il s'ensuit que, pour tout sous-ensemble A du domaine dom(π_4), le plus petit (au sens de l'inclusion) sous-domaine privilégié de catégorie (a) qui contiennent A existe toujours.

En effet cet ensemble est égal à l'intersection de tous les sous-domaines privilégiés de catégorie (a) qui contiennent A, sous-domaines parmi lesquels figure au moins dom(π_4).

Dans le tableau 2.3 on trouvera des exemples de notation Prolog IV pour exprimer que X appartient à l'une des 15 formes possible de sous-domaines privilégiés avec les deux variantes. Tous ces exemples sont présentés sous forme d'une disjonction multiple formant une requête.

TAB. 2.3 – *Exemples d'appartenances à des sous-domaines privilégiés de base*

```
>> X ~ tree;                                  % 1
   X ~ finite;                                % 2
   X ~ infinite;                              % 3
   X ~ list;                                  % 4
   X ~ nlist;                                 % 5
   X ~ [tree,tree,tree];                      % 6
   X ~ [cc(1,2),co(4,8.1),gt(`>9.1`),real];   % 7
   X ~ [cc(1,2) u co(4,8), gt(9), real];      % 7 variante
   X ~ leaf;                                  % 8
   X ~ nleaf;                                 % 9
   X ~ ident;                                 % 10
   X ~ nident;                                % 11
   X ~ oc(`<2.1`,9);                          % 12
   X ~ oc(2.1,3) u cc(3,5) u gt(5);           % 12 variante
   X ~ nreal;                                 % 13
   Y ex X = trio(X,Y,6), Y = duo(X,Y);        % 14
   X ~ ntree.                                 % 15
```

2.3.3 Opérations de la structure π_4

L'ensemble op(π_4) des symboles d'opérations f est constitué de

- tout identificateur avec arité, id/n, qui n'est pas un identificateur réservé du tableau 2.4,
- toute constante représentant un nombre rationnel.

TAB. 2.4 – *Identificateurs réservés pour les symboles de relation de π_4*

abs/2	bnleaf/1	div/3	ln/2	outoo/3
and/2	bnlist/1	divlin/3	log/2	pi/1
arccos/2	bnot/2	eq/2	lt/2	plus/3
arcsin/2	• bnprime/2	equiv/2	ltlin/2	pluslin/3
arctan/2	bnreal/2	exp/2	max/3	power/3
band/3	boc/4	finite/1	min/3	• prime/1
bcc/4	boo/4	floor/2	minus/3	real/1
bco/4	bor/3	• gcd/3	minuslin/3	root/3
bdif/3	boutcc/4	ge/2	• mod/3	sin/2
beq/3	boutco/4	gelin/2	n/3	sinh/2
bequiv/3	boutlist/3	gt/2	nidentifier/1	size/2
bfinite/2	boutoc/4	gtlin/2	nint/1	sqrt/2
bge/3	boutoo/4	identifier/1	nlist/1	square/2
bgt/3	• bprime/2	if/4	nleaf/1	tan/2
bidentifier/2	breal/2	impl/2	not/1	tanh/2
bimpl/3	bxor/3	index/3	• nprime/1	times/3
binfinite/2	cc/3	infinite/1	ntree/1	timeslin/3
binlist/3	ceil/2	inlist/2	nreal/2	tree/1
bint/2	co/3	• intdiv/3	oc/3	u/3
ble/3	conc/3	int/1	oo/3	uminus/2
bleaf/1	cos/2	• lcm/3	or/2	uminuslin/2
blist/1	cosh/2	le/2	outcc/3	uplus/2
blt/3	cot/2	leaf/1	outco/3	upluslin/2
bnidentifier/2	coth/2	lelin/2	outlist/2	xor/2
bnint/2	dif/2	list/1	outoc/3	

L'opération $\pi_4(f)$ que la structure π_4 associe au symbole n-aire f consiste à partir d'une suite $a_1, \ldots, a_n$ d'arbres à construire un arbre dont la racine est étiquetée, à peu de chose près par f, et dont la suite de fils immédiats est $a_1, \ldots, a_n$. Plus précisément l'opération $\pi_4(f)$ est l'application

$$\boxed{\pi_4(f): \quad (a_1, \ldots, a_n) \quad \mapsto \quad \begin{array}{ccc} & \ddot{f} & \\ \diagup & \ldots & \diagdown \\ a_1 & & a_n \end{array}}$$

où $\ddot{f}$ est, soit un identificateur égal[6] à f, soit le nombre rationnel représenté par la constante f. Bien entendu, les a_i sont des éléments du domaine dom(π_4) et l'entier n peut être nul.

6. Si l'identificateur f commence par le caractère ASCII accent circonflexe, on enlève ce caractère dans $\ddot{f}$. Ceci permet de créer des arbres étiquetés par des identificateurs réservés.

2.3.4 Relations de la structure π_4

L'ensemble $\text{rel}(\pi_4)$ des symboles de relation r est constitué de

- tout identificateur avec arité, id/n, du tableau 2.4 à la page 74,
- le symbole $\text{dif}'/2$,
- tout symbole de la forme in$A/1$, où A est un sous-domaine privilégié.

Le symbole $\text{dif}'/2$ et les symboles de la forme in$A/1$ ne sont pas accessibles au programmeur. Ils ont été introduits pour énoncer les propriétés utilisés par les algorithmes de résolution des contraintes.

Les tableaux 2.5, 2.6 et 2.7 donnent l'association symbole-relation

$r \mapsto \pi_4(r)$

pour chaque symbole r de $\text{rel}(\pi_4)$. Attention, les relations dont les noms sont précédés d'un point dans le tableau 2.4 ne sont pas implantées.

TAB. 2.5 – *Relations de la structure de base π_4, première partie*

Arbres et listes, relations

1 tree/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ n'est assujetti à aucune condition$\}$
2 ntree/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ ne peut exister$\}$
3 finite/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ est fini$\}$
4 infinite/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ est infini$\}$
5 list/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ est une liste$\}$
6 nlist/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ n'est pas une liste$\}$
7 leaf/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ est un arbre d'un nœud$\}$
8 nleaf/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ est un arbre de plus d'un nœud$\}$
9 real/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ est un nombre réel$\}$
10 nreal/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ n'est pas un nombre réel$\}$
11 identifier/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ est un identificateur$\}$
12 nidentifier/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ n'est pas un identificateur$\}$
13 eq/2	$\mapsto$	$\{(a,b) \in \mathbf{A}^2 \mid a = b\}$
14 dif/2	$\mapsto$	$\{(a,b) \in \mathbf{A}^2 \mid a \neq b\}$
15 inlist/2	$\mapsto$	$\{(a,b) \in \mathbf{A} \times \mathbf{L} \mid a$ figure dans $b\}$
16 outlist/2	$\mapsto$	$\{(a,b) \in \mathbf{A} \times \mathbf{L} \mid a$ ne figure pas dans $b\}$

Arbres et listes, pseudo-opérations produisant des booléens

17 bfinite/2	$\mapsto$	$\{(a,b) \in \mathbf{B} \times \mathbf{A} \mid a = 1$ ssi $b \in \pi_4($finite$/1)\}$
18 binfinite/2	$\mapsto$	$\{(a,b) \in \mathbf{B} \times \mathbf{A} \mid a = 1$ ssi $b \in \pi_4($infinite$/1)\}$
19 blist/2	$\mapsto$	$\{(a,b) \in \mathbf{B} \times \mathbf{A} \mid a = 1$ ssi $b \in \pi_4($list$/1)\}$
20 bnlist/2	$\mapsto$	$\{(a,b) \in \mathbf{B} \times \mathbf{A} \mid a = 1$ ssi $b \in \pi_4($nlist$/1)\}$
21 bleaf/2	$\mapsto$	$\{(a,b) \in \mathbf{B} \times \mathbf{A} \mid a = 1$ ssi $b \in \pi_4($leaf$/1)\}$
22 bnleaf/2	$\mapsto$	$\{(a,b) \in \mathbf{B} \times \mathbf{A} \mid a = 1$ ssi $b \in \pi_4($nleaf$/1)\}$
23 breal/2	$\mapsto$	$\{(a,b) \in \mathbf{B} \times \mathbf{A} \mid a = 1$ ssi $b \in \pi_4($real$/1)\}$
24 bnreal/2	$\mapsto$	$\{(a,b) \in \mathbf{B} \times \mathbf{A} \mid a = 1$ ssi $b \in \pi_4($nreal$/1)\}$
25 bidentifier/2	$\mapsto$	$\{(a,b) \in \mathbf{B} \times \mathbf{A} \mid a = 1$ ssi $b \in \pi_4($identifier$/1)\}$
26 bnidentifier/2	$\mapsto$	$\{(a,b) \in \mathbf{B} \times \mathbf{A} \mid a = 1$ ssi $b \in \pi_4($nidentifier$/1)\}$
27 beq/3	$\mapsto$	$\{(a,b,c) \in \mathbf{B} \times \mathbf{A}^2 \mid a = 1$ ssi $(b,c) \in \pi_4($eq$/2)\}$
28 bdif/3	$\mapsto$	$\{(a,b,c) \in \mathbf{B} \times \mathbf{A}^2 \mid a = 1$ ssi $(b,c) \in \pi_4($dif$/2)\}$
29 binlist/3	$\mapsto$	$\{(a,b,c) \in \mathbf{B} \times \mathbf{A} \times \mathbf{L} \mid a = 1$ ssi $(b,c) \in \pi_4($inlist$/2)\}$
30 boutlist/3	$\mapsto$	$\{(a,b,c) \in \mathbf{B} \times \mathbf{A} \times \mathbf{L} \mid a = 1$ ssi $(b,c) \in \pi_4($outlist$/2)\}$

Arbres et listes, pseudo-opérations courantes

31 size/2	$\mapsto$	$\{(a,b) \in \mathbf{Z} \times \mathbf{L} \mid a = \lvert b \rvert\}$
32 conc/3	$\mapsto$	$\{(a,b,c) \in \mathbf{L}^3 \mid a = b \bullet c\}$
33 index/3	$\mapsto$	$\{(a,b,c) \in \mathbf{A} \times \mathbf{L} \times \mathbf{Z} \mid 1 \leq c \leq \lvert b \rvert$ et a est le $c^{\text{ième}}$ élément de $b\}$
34 u/3	$\mapsto$	$\{(a,b,c) \in \mathbf{A}^3 \mid a = b$ ou $a = c\}$
35 n/3	$\mapsto$	$\{(a,b,c) \in \mathbf{A}^3 \mid a = b$ et $a = c\}$
36 if/4	$\mapsto$	$\{(a,b,c,d) \in \mathbf{A} \times \mathbf{B} \times \mathbf{A}^2 \mid$ si $b = 1$ alors $a = b$ sinon $a = c\}$

Booléens, relations

37 not/1	$\mapsto$	$\{a \in \mathbf{B} \mid a = 0)\}$
38 xor/2	$\mapsto$	$\{(a,b) \in \mathbf{B}^2 \mid a \neq b\}$
39 or/2	$\mapsto$	$\{(a,b) \in \mathbf{B}^2 \mid a = 1$ ou $b = 1\}$
40 and/2	$\mapsto$	$\{(a,b) \in \mathbf{B}^2 \mid a = 1$ et $b = 1\}$
41 impl/2	$\mapsto$	$\{(a,b) \in \mathbf{B}^2 \mid$ si $a = 1$ alors $b = 1\}$
42 equiv/2	$\mapsto$	$\{(a,b) \in \mathbf{B}^2 \mid a = b\}$

A : ensemble des arbres, **B** : ensemble $\{0, 1\}$,
L : ensemble des listes, **Z** : ensemble des entiers, **R** : ensemble des nombres réels.

TAB. 2.6 – *Relations de la structure de base π_4, deuxième partie*

Booléens, pseudo-opérations

43 bnot/2	$\mapsto$	$\{(a,b) \in \mathbf{B}^2 \mid a = 1$ ssi $b \in \pi_4(\mathrm{not}/1)\}$
44 bxor/3	$\mapsto$	$\{(a,b,c) \in \mathbf{B}^3 \mid a = 1$ ssi $(b,c) \in \pi_4(\mathrm{xor}/2)\}$
45 bor/3	$\mapsto$	$\{(a,b,c) \in \mathbf{B}^3 \mid a = 1$ ssi $(b,c) \in \pi_4(\mathrm{or}/2)\}$
46 band/3	$\mapsto$	$\{(a,b,c) \in \mathbf{B}^3 \mid a = 1$ ssi $(b,c) \in \pi_4(\mathrm{and}/2)\}$
47 bimpl/3	$\mapsto$	$\{(a,b,c) \in \mathbf{B}^3 \mid a = 1$ ssi $(b,c) \in \pi_4(\mathrm{impl}/2)\}$
48 bequiv/3	$\mapsto$	$\{(a,b,c) \in \mathbf{B}^3 \mid a = 1$ ssi $(b,c) \in \pi_4(\mathrm{equiv}/2)\}$

Réels, relations

49 int/1	$\mapsto$	$\{a \in \mathbf{R} \mid a$ est entier$\}$
50 nint/1	$\mapsto$	$\{a \in \mathbf{R} \mid a$ n'est pas entier$\}$
51 prime/1	$\mapsto$	$\{a \in \mathbf{R} \mid a$ est un entier premier$\}$
52 nprime/1	$\mapsto$	$\{a \in \mathbf{R} \mid a$ est un entier non premier$\}$
53 gt/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a > b\}$
54 gtlin/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a > b\}$
55 lt/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a < b\}$
56 ltlin/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a < b\}$
57 ge/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a \geq b\}$
58 gelin/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a \geq b\}$
59 le/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a \leq b\}$
60 lelin/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a \leq b\}$
61 cc/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a \in [b,c]\}$
62 co/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a \in [b,c)\}$
63 oc/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a \in (b,c]\}$
64 oo/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a \in (b,c)\}$
65 outcc/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a \notin [b,c]\}$
66 outco/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a \notin [b,c)\}$
67 outoc/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a \notin (b,c]\}$
68 outoo/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a \notin (b,c)\}$

Réels, pseudo-opérations produisant des booléens

69 bint/2	$\mapsto$	$\{(a,b) \in \mathbf{B}\times\mathbf{R} \mid a = 1$ ssi $b \in \pi_4(\mathrm{int}/1)\}$
70 bnint/2	$\mapsto$	$\{(a,b) \in \mathbf{B}\times\mathbf{R} \mid a = 1$ ssi $b \in \pi_4(\mathrm{nint}/1)\}$
71 bprime/2	$\mapsto$	$\{(a,b) \in \mathbf{B}\times\mathbf{R} \mid a = 1$ ssi $b \in \pi_4(\mathrm{prime}/1)\}$
72 bnprime/2	$\mapsto$	$\{(a,b) \in \mathbf{B}\times\mathbf{R} \mid a = 1$ ssi $b \in \pi_4(\mathrm{nprime}/1)\}$
73 bgt/3	$\mapsto$	$\{(a,b,c) \in \mathbf{B}\times\mathbf{R}^2 \mid a = 1$ ssi $(b,c) \in \pi_4(\mathrm{gt}/2)\}$
74 blt/3	$\mapsto$	$\{(a,b,c) \in \mathbf{B}\times\mathbf{R}^2 \mid a = 1$ ssi $(b,c) \in \pi_4(\mathrm{lt}/2)\}$
75 bge/3	$\mapsto$	$\{(a,b,c) \in \mathbf{B}\times\mathbf{R}^2 \mid a = 1$ ssi $(b,c) \in \pi_4(\mathrm{ge}/2)\}$
76 ble/3	$\mapsto$	$\{(a,b,c) \in \mathbf{B}\times\mathbf{R}^2 \mid a = 1$ ssi $(b,c) \in \pi_4(\mathrm{le}/2)\}$
77 bcc/4	$\mapsto$	$\{(a,b,c,d) \in \mathbf{B}\times\mathbf{R}^3 \mid a = 1$ ssi $(b,c,d) \in \pi_4(\mathrm{cc}/3)\}$
78 bco/4	$\mapsto$	$\{(a,b,c,d) \in \mathbf{B}\times\mathbf{R}^3 \mid a = 1$ ssi $(b,c,d) \in \pi_4(\mathrm{co}/2)\}$
79 boc/4	$\mapsto$	$\{(a,b,c,d) \in \mathbf{B}\times\mathbf{R}^3 \mid a = 1$ ssi $(b,c,d) \in \pi_4(\mathrm{oc}/2)\}$
80 boo/4	$\mapsto$	$\{(a,b,c,d) \in \mathbf{B}\times\mathbf{R}^3 \mid a = 1$ ssi $(b,c,d) \in \pi_4(\mathrm{oo}/2)\}$
81 boutcc/4	$\mapsto$	$\{(a,b,c,d) \in \mathbf{B}\times\mathbf{R}^3 \mid a = 1$ ssi $(b,c,d) \in \pi_4(\mathrm{outcc}/2)\}$
82 boutco/4	$\mapsto$	$\{(a,b,c,d) \in \mathbf{B}\times\mathbf{R}^3 \mid a = 1$ ssi $(b,c,d) \in \pi_4(\mathrm{outco}/2)\}$
83 boutoc/4	$\mapsto$	$\{(a,b,c,d) \in \mathbf{B}\times\mathbf{R}^3 \mid a = 1$ ssi $(b,c,d) \in \pi_4(\mathrm{outoc}/2)\}$
84 boutoo/4	$\mapsto$	$\{(a,b,c,d) \in \mathbf{B}\times\mathbf{R}^3 \mid a = 1$ ssi $(b,c,d) \in \pi_4(\mathrm{outoo}/2)\}$

A : ensemble des arbres, **B** : ensemble $\{0,1\}$,
L : ensemble des listes, **Z** : ensemble des entiers, **R** : ensemble des nombres réels.

TAB. 2.7 – *Relations de la structure de base π_4, troisième partie*

Réels et entiers, pseudo-opérations courantes

85 floor/2	$\mapsto$	$\{(a,b) \in \mathbf{Z}\times\mathbf{R} \mid a = \lfloor b \rfloor\}$
86 ceil/2	$\mapsto$	$\{(a,b) \in \mathbf{Z}\times\mathbf{R} \mid a = \lceil b \rceil\}$
87 gcd/3	$\mapsto$	$\{(a,b,c) \in \mathbf{Z}^3 \mid a \geq 0, b \geq 0, c \geq 0$ et $a = \text{pgcd}(b,c)\}$
88 lcm/3	$\mapsto$	$\{(a,b,c) \in \mathbf{Z}^3 \mid a \geq 0, b \geq 0, c \geq 0$ et $a = \text{ppcm}(b,c)\}$
89 intdiv/3	$\mapsto$	$\{(a,b,c) \in \mathbf{Z}^3 \mid$ il existe $d \in \mathbf{Z}$ tel que $b = ac + d$ et $c > d \geq 0\}$

Réels, pseudo-opérations courantes

90 uplus/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a = +b\}$
91 upluslin/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a = +b\}$
92 uminus/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a = -b\}$
93 uminuslin/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a = -b\}$
94 abs/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a = \lvert b \rvert\}$
95 plus/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a = b + c\}$
96 pluslin/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a = b + c\}$
97 minus/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a = b - c\}$
98 minuslin/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a = b - c\}$
99 times/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a = b \times c\}$
100 timeslin/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a = b \times c\}$
101 div/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid c \neq 0$ et $a = b/c\}$
102 divlin/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid c \neq 0$ et $a = b/c\}$
103 mod/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid 0 \leq a < c$ et $a = b \bmod c\}$
104 min/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a = \min(b,c)\}$
105 max/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a = \max(b,c)\}$

Réels, pseudo-opérations correspondant aux fonctions spéciales

106 pi/1	$\mapsto$	$\{a \in \mathbf{R} \mid a = \pi\}$
107 square/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a = b^2\}$
108 sqrt/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid b \geq 0$ et $a = \sqrt{b}\}$
109 power/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^2\times\mathbf{Z} \mid c \geq 0$ et $a = b^c\}$
110 root/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^2\times\mathbf{Z} \mid c \geq 0$ et $a = \sqrt[c]{b}\}$
111 arccos/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid 0 \leq a \leq \pi$ et $b = \cos a\}$
112 arcsin/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid -\pi/2 \leq a \leq \pi/2$ et $b = \sin a\}$
113 arctan/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid -\pi/2 < a < \pi/2$ et $b = \tan a\}$
114 cos/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a = \cos b\}$
115 cosh/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a = \cosh b\}$
116 cot/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid b \bmod 2\pi \neq 0$ et $a = \cot b\}$
117 coth/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a = \coth b\}$
118 exp/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a = \mathrm{e}^b\}$
119 ln/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid b > 0$ et $a = \ln b\}$
120 log/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid b > 0$ et $a = \log b\}$
121 sin/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a = \sin b\}$
122 sinh/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a = \sinh b\}$
123 tan/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid b \bmod 2\pi \neq \pi/2$ et $a = \tan b\}$
124 tanh/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a = \tanh b\}$

Relations internes, pour chaque sous-domaine privilégié A

125 dif′/2	$\mapsto$	$\{(a,b) \in \mathbf{A}^2 \mid a \neq b\}$
126 in$A$/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ est élément du sous-domaine privilégié $A\}$

A : ensemble des arbres, **B** : ensemble $\{0,1\}$,
L : ensemble des listes, **Z** : ensemble des entiers, **R** : ensemble des nombres réels.

2.4 Axiomatisation de π_4

2.4.1 Généralités

Soit p une contrainte dans laquelle $x_1, \ldots, x_n$ sont les seules variables qui ont des occurrences libres. Résoudre p consiste à trouver les jeux de valeurs possibles de ces variables qui respectent la condition exprimée par p dans la structure π_4. Il faut donc trouver l'ensemble des solutions σ de p dans la structure π_4, c'est-à-dire l'ensemble des affectations σ de variables qui sont telles que la combinaison de σ avec la structure π_4 forment une interprétation I pour laquelle

$I^*(p) = \text{vrai}$.

Cet ensemble de solutions étant généralement infini, on s'attachera plutôt à simplifier p en une contrainte équivalente q qui rend l'ensemble recherché de solutions plus « visible ». En particulier si p n'a pas de solution dans π_4 on aimerait que q se réduise à la constante logique *false*.

Compte tenu de la richesse de la structure π_4, il est complètement illusoire de vouloir atteindre ce dernier objectif. Il faut donc se résigner à utiliser un « solveur » de contraintes incomplet. Mais comment, sans entrer dans tous les détails de son implantation faire connaître au programmeur toutes les capacités et incapacités du solveur ? C'est là qu'intervient l'axiomatisation de la structure π_4. L'idée est de retenir l'ensemble exact $\mathcal{T}(\pi_4)$ des propriétés de π_4 utilisées par l'algorithme de résolution et de garantir que ces propriété sont utilisées au mieux. Chaque propriété p retenue sera une proposition du premier ordre telle que

$\pi_4^*(p) = \text{vrai}$

et l'ensemble $\mathcal{T}(\pi_4)$ de ces p sera une théorie du premier ordre qui constituera une axiomatisation partielle de π_4 c'est-à-dire qu'il existera des propositions p tels qu'à la fois

$\mathcal{T}(\pi_4) \not\models p \quad \text{et} \quad \mathcal{T}(\pi_4) \not\models \neg p$

Lorsque le programmeur donnera une contrainte p à résoudre à Prolog IV le système lui retournera une contrainte q non seulement équivalente à p dans la structure π_4 mais dans toutes les structures π_4' ayant les propriétés $\mathcal{T}(\pi_4)$, ce qui s'écrit

$\mathcal{T}(\pi_4) \models p \equiv q$

Cette contrainte q ne se réduira à la constante logique *false* que si elle n'a pas de solutions dans la structure π_4 et si en plus elle n'a de solutions dans aucune autre structure qui a les propriétés $\mathcal{T}(\pi_4)$, c'est-à-dire si

$\mathcal{T}(\pi_4) \models p \equiv \mathit{false}$

Comme le montre le tableau 2.8 la théorie $\mathcal{T}(\pi_4)$ est composé de 25 schémas possibles d'axiomes regroupés en 8 catégories. Il s'agit d'un premier essai d'axiomatisation et il est fort probable qu'il y ait encore des erreurs.

TAB. 2.8 – *Propriétés retenues de la structure π_4*

1	Sous-domaine vide	Sous-domaines privilégiés
2	Sous-domaine universel	” ”
3	Intersection de sous-domaines	” ”
4	Opérations distinctes, résultats distincts	Opérations sur les arbres
5	Au moins une solution	” ”
6	Propagation des égalités	” ”
7	Sous-domaines singletoniques	Interactions des sous-domaines avec les opérations
8	Réduction de sous-domaines	” ”
9	Arbres infinis	” ”
10	Taille finie des listes	” ”
11	Réduction de sous-domaines	Relation générales
12	Construction de contrainte	” ”
13	Multiplication par scalaire positif	Contraintes linéaires
14	Somme de contraintes	” ”
15	Respect de l'ordre	” ”
16	Constante cachée	” ”
17	Linéarisation de l'égalité	” ”
18	Typage numérique	Relations à linéariser
19	Linéarisation de contrainte	” ”
20	Détection d'égalités	Relations dif, bdif et beq
21	Egalités, bdif et beq	” ”
22	Détection de diségalités	” ”
23	Diségalités, bdif et beq	” ”
24	Propagation du dif	Propagation supplémentaire d'égalités
25	Propagation du dif$'$	” ”

2.4.2 Axiomatisation des sous-domaines privilégiés

Les premières propriétés retenues concernent l'appartenance à un sous-domaine privilégié :

1 $\forall x$
$\text{in}\emptyset\, x \;\equiv\; \textit{false}$

2 $\forall x$
$\text{in}\mathbf{A}\, x$

3 $\forall x$
$$\begin{pmatrix} \quad \text{in}A\, x \\ \wedge\; \text{in}B\, x \end{pmatrix} \;\equiv\; \text{in}A{\cap}B\, x$$

où **A** est l'ensemble de tous les arbres c'est-à-dire tout le domaine et A, B des sous-domaines privilégiés quelconques.

La propriété 1 exprime que l'ensemble vide ne contient aucun élément, la propriété 2 que tout individu est un arbre et la troisième qu'un individu qui appartient à deux sous-domaines privilégiés appartient aussi à leur intersection, qui comme nous l'avons vu est aussi un sous-domaine privilégié. Voici quelques requêtes pour illustrer ceci.

```
>> X ~ ntree.
false.

>> X = X.
X ~ tree.

>> X ~ cc(1,5), X ~ cc(2,6).
X ~ cc(2,5).
```

2.4.3 Axiomatisation des opérations sur les arbres

On retient 3 propriétés :

4 $\forall x_1 \forall y_1 \forall u_1 \ldots \forall u_m \forall v_1 \ldots \forall v_n$
$$\left(\begin{array}{ll} & x_1 = y_1 \\ \wedge & x_1 = f\, u_1 \ldots u_m \\ \wedge & y_1 = g\, v_1 \ldots v_n \end{array}\right) \equiv \mathit{false}$$

5 $\forall u_1 \ldots \forall u_m \exists x_1 \ldots \exists x_n$
$$\left(\begin{array}{ll} & x_1 = t_1(x_1, .., x_n, u_1, \ldots, u_m) \\ & \ldots \\ \wedge & x_n = t_n(x_1, .., x_n, u_1, \ldots, u_m) \end{array}\right)$$

6 $\forall u_1 \ldots \forall u_m \forall v_1 \ldots \forall v_m \forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_n$
$$\left(\begin{array}{ll} & x_1 = t_1(x_1, .., x_n, u_1, .., u_m) \\ & \ldots \\ \wedge & x_n = t_n(x_1, .., x_n, u_1, .., u_m) \\ \wedge & y_1 = t_1(y_1, .., y_n, v_1, .., v_m) \\ & \ldots \\ \wedge & y_n = t_n(y_1, .., y_n, v_1, .., v_m) \end{array}\right) \Rightarrow \left(\begin{array}{c} \left(\begin{array}{ll} & u_1 = v_1 \\ & \ldots \\ \wedge & u_m = v_m \end{array}\right) \\ \equiv \\ \left(\begin{array}{ll} & x_1 = y_1 \\ & \ldots \\ \wedge & x_n = y_n \end{array}\right) \end{array}\right)$$

où f et g sont des symboles d'opérations m-aires et n-aires distincts, où les $t_i(x_1, .., x_n, u_1, .., u_m)$ sont des termes construits avec une occurrence de symbole d'opération et des variables prises dans $\{x_1, .., x_n, u_1, .., u_m\}$ et où les $t_i(y_1, .., y_n, v_1, .., v_m)$ sont les mêmes termes dans lequels on a remplacé les x_j par des y_j et les u_j par des v_j.

La propriété 4 exprime que deux arbres construits par des opérations nommées par des symboles distincts sont distincts. Les requêtes qui suivent illustrent ce phénomène.

```
>> f(X)=g(Y).
false.

>> a=b.
false.

>> 1/3=1/4.
false.

>> f(X)=f(Y,X).
false.
```

On notera que dans la dernière requête les deux occurrences de `f` sont considérées comme faisant référence à deux symboles distincts `f/1` et `f/2`. L'arité du symbole est en fait codée dans les virgules et les parenthèses.

La propriété 5 exprime qu'un type particulier de conjonction d'équations admet au moins une solution

```
>> X ex Y ex
   X=f(X,Y), Y=g(X,Y).
true.
```

La propriété 6 permet de décomposer une équation en plusieurs équations.

```
>> f(X,Y) = f(U,V).
Y = V,
X = U,
V ~ tree,
U ~ tree.
```

Elle affirme aussi que la conjonction d'équations de l'axiome 5 admet au plus une solution. Ceci permet des simplifications importantes.

```
>> X = f(f(f(X))).
X = f(X).

>> X = g(g(g(X,Y),Y),Y).
X = g(X,Y),
Y ~ tree.
```

Soit ϕ la structure obtenue en privant π_4 de ses symboles de relations. Il existe d'autres structures que cette structure d'arbre ϕ dans laquelle toutes ces propriétés sont vraies. Cependant Michael Maher[7] a montré qu'il existe un ensemble de propriétés, en fait équivalent à l'ensemble $\mathcal{T}$ des propriétés 4,5,6, qui forme une axiomatisation complète de ϕ. Autrement dit si p est une proposition alors, pour toute structure ϕ' dans laquelle les propositions $\mathcal{T}$ sont vraies, on a

$$\phi(p) = \phi'(p).$$

2.4.4 Axiomatisations des interactions des sous-domaines avec les opérations

Les 4 propriétés retenues qui suivent concernent les interactions des contraintes de sous-domaines et des opérations :

7. Complete axiomatizations of the algebra of finite, rational and infinite trees. *Technical report, IBM T.J. Watson Research Center*, 1988

7 $$\forall x_1 \ldots \forall x_n \left(\begin{array}{ll} & x_1 = t_1(x_1, \ldots, x_n) \\ & \ldots \\ \wedge & x_n = t_n(x_1, \ldots, x_n) \end{array}\right) \equiv \left(\begin{array}{ll} & \mathrm{in}\{a_1\}\, x_1 \\ & \ldots \\ \wedge & \mathrm{in}\{a_n\}\, x_n \end{array}\right)$$

8 $$\forall x_1 \ldots \forall x_n \left(\begin{array}{ll} & x_0 = f x_1 \ldots x_n \\ \wedge & \mathrm{in} A_0\, x_0 \\ \wedge & \mathrm{in} A_1\, x_1 \\ & \ldots \\ \wedge & \mathrm{in} A_n\, x_n \end{array}\right) \equiv \left(\begin{array}{ll} & x_0 = f x_1 \ldots x_n \\ \wedge & \mathrm{in} A'_0\, x_0 \\ \wedge & \mathrm{in} A'_1\, x_1 \\ & \ldots \\ \wedge & \mathrm{in} A'_n\, x_n \end{array}\right)$$

9 $$\forall x_1 \ldots \forall x_n \left(\begin{array}{ll} & x_1 = s_1(x_2) \\ \wedge & x_2 = s_2(x_3) \\ & \ldots \\ \wedge & x_n = s_n(x_1) \end{array}\right) \Rightarrow \mathrm{in}\mathbf{I}\, x_1$$

10 $$\forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_n \left(\begin{array}{ll} & y_1 = .\, x_1 y_2 \\ \wedge & y_2 = .\, x_2 y_3 \\ & \ldots \\ \wedge & y_n = .\, x_n y_1 \end{array}\right) \Rightarrow \mathrm{in}\mathbf{K}\, y_1$$

où les $t_i(x_1, \ldots, x_n)$ sont des termes construits avec un symbole d'opération et des variables prises dans $\{x_1, \ldots, x_n\}$, où les a_i sont des arbres doublement rationnels vérifiant l'équivalence 7, où les A_i et A'_i sont des sous-domaines privilégiés de catégorie (a) vérifiant l'équivalence 8, où les $s_i(x_j)$ sont des termes construits avec un symbole d'opération et au moins une occurrence de x_j, où **I** est l'ensemble des arbres infinis et où **K** est l'ensemble des arbres qui ne sont pas des listes. Rappelons que les listes sont construites à l'aide du constructeur point.

La propriété 7 établit l'équivalence entre deux propriétés : « être contraint par les opérations et les égalités à n'avoir qu'une seule valeur » et « appartenir à un sous-domaine privilégié singletonique ».

```
>> X ~ cc(2,2).
X = 2.
```

La propriété 8 permet de réduire les sous-domaines des variables intervenant dans une opération et entre autres d'obtenir le sous-domaine vide, qui d'après la propriété 1 à la page 80 va produire *false*.

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

>> X = [Y],
   X ~ real,
   Y ~ real.
false.
```

La propriété 9 correspond au fameux « occurs-check » qui détecte les arbres infinis.

```
>> X = f(g(X,U)), X ~ finite.
false.
```

Enfin la propriété 10 interdit qu'une liste ait une infinité d'éléments.

```
>> X = [1|X], X ~ list.
false.
```

2.4.5 Axiomatisation des relations générales

Les deux prochaines propriétés retenues concernent tous les symboles de relation r à l'exception de ceux qui se terminent par « lin » et de ceux qui sont inaccessibles au programmeur.

$$11 \quad \forall x_1 \dots \forall x_n \left(\begin{array}{l} \quad r\, x_1 \dots x_n \\ \wedge \ \ \mathrm{in}A_1\, x_1 \\ \quad \dots \\ \wedge \ \ \mathrm{in}A_n\, x_n \end{array} \right) \equiv \left(\begin{array}{l} \quad r\, x_1 \dots x_n \\ \wedge \ \ \mathrm{in}A'_1\, x_1 \\ \quad \dots \\ \wedge \ \ \mathrm{in}A'_n\, x_n \end{array} \right)$$

$$12 \quad \forall x_1 \dots \forall x_n \left(\begin{array}{l} \quad r\, x_1 \dots x_n \\ \wedge \ \ \mathrm{in}B_1\, x_1 \\ \quad \dots \\ \wedge \ \ \mathrm{in}B_n\, x_n \end{array} \right) \equiv c(x_1, \dots, x_n)$$

Ici les A_i et A'_i sont des sous-domaines privilégiés de catégorie (a) tels que l'équivalence 11 soit vérifiée. La formule $c(x_1, \dots, x_n)$ est une *formule de base* de l'une des 7 formes :

1	*true*,	constante logique,
2	*false*,	constante logique,
3	$p \wedge q$,	conjonction,
4	$\exists x p$,	quantification,
5	$x \in C$,	sous-domaine privilégié,
6	$x = y$,	égalité,
7	$x_0 = f\, x_1 \dots x_n$,	construction,

où p et q sont elles-mêmes des formules de base et C un sous-domaine privilégié quelconque. Les B_i sont des sous-domaines privilégiés de catégorie (b) tels que l'équivalence 12 soit respectée dans π_4.

La propriété 11 permet de réduire les sous-domaines des variables intervenant dans une relation générale et entre autres d'obtenir le sous-domaine vide, qui d'après l'axiome 1 à la page 80 va produire *false*.

```
>> X = conc(Y,Z).
Z ~ list,
Y ~ list,
X ~ list.

>> X = index(Y,I).
X ~ tree,
Y ~ list,
I ~ ge(1).

>> B = beq(X,Y).
Y ~ tree,
X ~ tree,
B ~ cc(0,1).

>> X = cos(X).
X ~ oo('>0.739085','>0.7390852')

>> gt(square(X),1).
X ~ real

>> X = Y.*.Z,
   X ~ cc(1,3),
   Y ~ cc(-0.5,1.5),
   Z ~ cc(-0.5,1.5).
Z ~ cc('>0.6666666',1.5),
Y ~ cc('>0.6666666',1.5),
X ~ cc(1,2.25).

>> X = plus(Y,Z),
   X ~ cc(3,4),
   Y ~ cc(0,1),
   Z ~ cc(0,1).
false.
```

La propriété 12 exprime le fait que certaines formes de contraintes, équivalentes dans π_4 à des formules de base, doivent se comporter comme telles. Dans les parties numériques ceci a souvent pour effet de passer à un calcul en précision infinie.

```
>> X = conc(Y,Z),
   X ~ [tree,tree,tree,tree],
   Y ~ tree,
   Z ~ [tree,tree].
A ex B ex C ex D ex
Z = [B,A],
Y = [D,C],
X = [D,C,B,A],
A ~ tree,
B ~ tree,
C ~ tree,
D ~ tree.

>> X = index(Y,I),
   X ~ tree,
   Y ~ [tree,tree,tree,tree],
   I ~ cc(3,3).
I = 3,
Y ~ [tree,tree,X,tree],
X ~ tree.

>> B = beq(X,Y),
   B ~ cc(1,1).
X = Y,
B = 1,
Y ~ tree.

>> square(X,Y),
   X ~ tree,
   Y ~ cc(1/3,1/3).
Y = 1/3,
X = 1/9.

>> X = 1/3.*.6/7.
X = 2/7.

>> 1 = plus(1/3,X).
   X = 2/3.
false.
```

Restriction Dans les axiomes 11 et 12, si r est un des symboles binlist/3, boutlist/3, conc/3, index/3, inlist/2, outlist/2, alors les A_i, A'_i et B_i doivent être des sous-domaines privilégiés que l'on peut obtenir sans faire intervenir l'ensemble des arbres finis ou l'ensemble des arbres infinis.

2.4.6 Axiomatisation des contraintes linéaires

Pour exprimer les axiomes des contraintes linéaires nous introduirons la notation

$$a_0 + a_1x_1 + \cdots + a_nx_n \geq 0 \tag{2.5}$$

souvent abrégée en $a_0 + \sum a_ix_i \geq 0$ pour désigner l'inégalité linéaire écrite sous forme fonctionnelle où le produit correspond à la relation «timeslin», la somme à la relation «pluslin» et le signe $\geq$ à la relation «gelin», où les a_i sont des entiers relatifs et bien entendu les x_i des variables distinctes. On considère aussi que la contrainte 2.5 désigne toute contrainte obtenue en permutant ou associant différemment les monômes a_ix_i qui la composent ou en ajoutant d'autres monômes dont les coefficients a_i seraient nuls. Nous retiendrons 5 propriétés :

13 $\forall x_1 \ldots \forall x_n$
$$a_0 + \sum a_i x_i \geq 0 \;\equiv\; ka_0 + \sum ka_i x_i \geq 0$$

14 $\forall x_1 \ldots \forall x_n$
$$\left(\begin{array}{ll} & a_0 + \sum a_i x_i \geq 0 \\ \wedge & b_0 + \sum b_i x_i \geq 0 \end{array}\right) \;\Rightarrow\; a_0 + b_0 + \sum (a_i + b_i) x_i \geq 0$$

15 $-a_0 \geq 0 \;\equiv\; \mathit{false}$

16 $\forall x_1$
$$\left(\begin{array}{ll} & + a_0 + a_1 x_1 \geq 0 \\ \wedge & - a_0 - a_1 x_1 \geq 0 \end{array}\right) \;\equiv\; x_1 = -a_0/a_1$$

17 $\forall x_1 \forall x_2$
$$\left(\begin{array}{ll} & x_1 = x_2 \\ \wedge & \mathrm{in}\mathbf{R}\, x_1 \\ \wedge & \mathrm{in}\mathbf{R}\, x_2 \end{array}\right) \;\Rightarrow\; \left(\begin{array}{ll} & + x_1 - x_2 \geq 0 \\ \wedge & - x_1 + x_2 \geq 0 \end{array}\right)$$

où k est un entier strictement positif, où a_0 est strictement positif dans l'axiome 15 et où **R** est l'ensemble des réels.

Les propriétés 13 et 14 expriment que, si on se limite à des coefficients positifs, la conjonction de deux contraintes implique toute combinaison linéaire de celles-ci. La propriété 15 exprime qu'un entier ne peut être à la fois positif et négatif.

```
>> gelin(-5+3*X-2*Y,0),
   gelin(8-6*X+4*Y,0).
false.
```

La propriété 16 permet de détecter les variables qui n'ont qu'une seule valeur possible.

```
>> gelin(-5+3*X,0),
   gelin(5-3*X,0).
X = 5/3.
```

La propriété 17 permet de coder une égalité numérique sous forme d'inégalités linéaires.

```
>> X = Y, gelin(0,X), gelin(Y,1).
false.
```

2.4.7 Axiomatisation des relations à linéariser

Une *contrainte linéaire générale* est une contrainte de l'une des 6 formes

1	*true*,	constante logique,
2	*false*,	constante logique,
3	$p \wedge q$,	conjonction,
4	$\exists x p$,	quantification,
5	dif xy,	non-égalité,
6	$a_0 + a_1 x_1 + \cdots + a_n x_n \geq 0$,	contrainte linéaire de base.

où p et q sont elles-mêmes des contraintes linéaires générales.

Voici les deux propriétés retenues des contraintes construites avec des symboles r de relation se terminant en «lin».

18 $\forall x_1 \ldots \forall x_n$

$$r\, x_1 \ldots x_n \;\Rightarrow\; \left(\begin{array}{rl} & \mathrm{in}\mathbf{R}\, x_1 \\ & \ldots \\ \wedge & \mathrm{in}\mathbf{R}\, x_n \end{array}\right)$$

19 $\forall x_1 \ldots \forall x_n$

$$\left(\begin{array}{rl} & r\, x_1 \ldots x_n \\ \wedge & \mathrm{in}B_1\, x_1 \\ & \ldots \\ \wedge & \mathrm{in}B_n\, x_n \end{array}\right) \;\equiv\; c(x_1, \ldots, x_n)$$

Ici **R** est l'ensemble des réels, $c(x_1, \ldots, x_n)$ est une contrainte linéaire générale et les B_i des sous-domaines privilégiés de catégorie (b) qui vérifient l'équivalence 19 dans π_4.

La propriété 18 signale que les arguments d'une contrainte atomique en «lin» sont toujours des nombres réels.

```
>> ltlin(X,Y).
Y ~ real,
X ~ real.

>> X = timeslin(Y,Z).
Z ~ real,
Y ~ real,
X ~ real.
```

La propriété 19 exprime le fait que certaines formes de contraintes, équivalentes dans π_4 à des formules linéaire générales, doivent se comporter comme ces formules linéaires.

```
>> Y = timeslin(A,X),
   Y = timeslin(B,X),
   A = 2,
   B = 3.
B = 3,
X = 0,
A = 2,
Y = 0.

>> gelin(X,Y),
   lelin(X,Y),
   X = 4.
Y = 4,
X = 4.

>> gelin(X,Y),
   ltlin(X,Y).
false.
```

2.4.8 Axiomatisation des relations dif, bdif et beq

Le traitement des contraintes construites avec les symboles de relation dif, bdif et beq est très complet. Il respecte 6 axiomes dont voici les 4 premiers :

20 $\forall x_1 \forall x_2$
$$\left(\begin{array}{c} (\ x_1 = x_2\ ) \\ \vee \\ \left(\begin{array}{rl} & +\,x_1 - x_2 \geq 0 \\ \wedge & -\,x_1 + x_2 \geq 0 \end{array} \right) \end{array} \right) \Rightarrow \neg \mathrm{dif}\, x_1 x_2$$

21 $\forall x_0 \forall x_1 \forall x_2$
$$\neg \mathrm{dif}\, x_1 x_2 \Rightarrow \left(\begin{array}{rl} & (\mathrm{beq}\, x_0 x_1 x_2 \Rightarrow x_0 = 1) \\ \wedge & (\mathrm{bdif}\, x_0 x_1 x_2 \Rightarrow x_0 = 0) \end{array} \right)$$

22 $\forall x_1 \forall x_2 \forall u_1 \ldots \forall u_m \forall v_1 \ldots \forall v_n$
$$\left(\begin{array}{c} \left(\begin{array}{rl} & \mathrm{in} A_1\, x_1 \\ \wedge & \mathrm{in} A_2\, x_2 \end{array} \right) \\ \vee \\ \left(\begin{array}{rl} & x_1 = f\, u_1 \ldots u_m \\ \wedge & x_2 = g\, v_1 \ldots v_n \end{array} \right) \\ \vee \\ \left(\begin{array}{rl} & +\,a_0 + x_1 - x_2 \geq 0 \\ \wedge & -\,a_0 - x_1 + x_2 \geq 0 \end{array} \right) \end{array} \right) \Rightarrow \mathrm{dif}'\, x_1 x_2$$

23 $\forall x_0 \forall x_1 \forall x_2$
$$\mathrm{dif}'\, x_1 x_2 \Rightarrow \left(\begin{array}{rl} & (\mathrm{bdif}\, x_0 x_1 x_2 \Rightarrow x_0 = 1) \\ \wedge & (\mathrm{beq}\, x_0 x_1 x_2 \Rightarrow x_0 = 0) \end{array} \right)$$

où A_1 et A_2 sont des sous-domaines privilégiés disjoints de catégorie (a), où f et g sont des symboles d'opération distincts et où a_0 un entier non nul.

La propriété 20 permet de détecter un certain nombre d'égalités qui sont prises en compte dans la propriété 21.

```
>> X = Y, dif(X,Y).
false.

>> gelin(X,Y), gelin(Y,X), dif(X,Y).
false.

>> X = Y, Z = beq(X,Y).
Z = 1,
X = Y,
Y ~ tree.

>> gelin(X,Y), gelin(Y,X), Z = bdif(X,Y).
Z = 0,
Y ~ real,
X ~ real.
```

La propriété 22 permet de détecter un certain nombre de diségalités qui sont prises en compte dans la propriété 23.

```
>> X ~ finite, Y ~ infinite, Z = beq(X,Y).
Z = 0,
Y ~ infinite,
X ~ finite.

>> X = f(U), Y = g(V), Z = bdif(X,Y).
Z = 1,
Y = g(V),
X = f(U),
V ~ tree,
U ~ tree.

>> X = Y+1, Z = beq(X,Y).
Z = 0,
Y ~ real,
X ~ real.
```

2.4.9 Axiomatisation de la propagation supplémentaire d'égalités

Par négation des relations nommées dif et dif′ ont obtient des égalités. Ces égalités doivent respecter l'équivalent de l'axiome 6 à la page 81 :

24 $\forall u_1 \ldots \forall u_m \forall v_1 \ldots \forall v_m \forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_n$

$$\left(\begin{array}{ll} & x_1 = t_1(x_1,..,x_n,u_1,..,u_m) \\ & \ldots \\ \wedge & x_n = t_n(x_1,..,x_n,u_1,..,u_m) \\ \wedge & y_1 = t_1(y_1,..,y_n,v_1,..,v_m) \\ & \ldots \\ \wedge & y_n = t_n(y_1,..,y_n,v_1,..,v_m) \end{array}\right) \Rightarrow \left(\begin{array}{c} \left(\begin{array}{ll} & \neg\mathrm{dif}\, u_1 v_1 \\ & \ldots \\ \wedge & \neg\mathrm{dif}\, u_m v_m \end{array}\right) \\ \equiv \\ \left(\begin{array}{ll} & \neg\mathrm{dif}\, x_1 y_1 \\ & \ldots \\ \wedge & \neg\mathrm{dif}\, x_n y_n \end{array}\right) \end{array}\right)$$

25 $\forall u_1 \ldots \forall u_m \forall v_1 \ldots \forall v_m \forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_n$

$$\left(\begin{array}{ll} & x_1 = t_1(x_1,..,x_n,u_1,..,u_m) \\ & \ldots \\ \wedge & x_n = t_n(x_1,..,x_n,u_1,..,u_m) \\ \wedge & y_1 = t_1(y_1,..,y_n,v_1,..,v_m) \\ & \ldots \\ \wedge & y_n = t_n(y_1,..,y_n,v_1,..,v_m) \end{array}\right) \Rightarrow \left(\begin{array}{c} \left(\begin{array}{ll} & \neg\mathrm{dif}'\, u_1 v_1 \\ & \ldots \\ \wedge & \neg\mathrm{dif}'\, u_m v_m \end{array}\right) \\ \equiv \\ \left(\begin{array}{ll} & \neg\mathrm{dif}'\, x_1 y_1 \\ & \ldots \\ \wedge & \neg\mathrm{dif}'\, x_n y_n \end{array}\right) \end{array}\right)$$

Voici des exemples de propagation d'égalités par l'axiome 24

```
>> gelin(X,Y), gelin(Y,X),
   U = f(U,X), V = f(V,Y),
   Z = beq(U,V).
Z = 1,
U = f(U,X),
V = f(V,Y),
Y ~ real,
X ~ real.

>> gelin(X,Y), gelin(Y,X),
   U = f(U,X), V = f(V,Y),
   dif(U,V).
false.
```

et en voici par l'axiome 25

```
>> X ~ finite, Y ~ infinite, Z = beq([X,U],[Y,V]).
Z = 0,
V ~ tree,
U ~ tree,
Y ~ infinite,
X ~ finite.

>> X = f(U), Y = g(V), Z = bdif([X,U],[Y,V]).
Z = 1,
Y = g(V),
X = f(U),
V ~ tree,
U ~ tree.

>> X = Y+1, Z = beq([X,U],[Y,V]).
Z = 0,
V ~ tree,
U ~ tree,
Y ~ real,
X ~ real.
```

2.5 Structures enrichies

2.5.1 Notion générale de programme

Etant donnée une structure ϕ, comprenant notamment une famille de relations $\phi(r)$ indicées par des symboles de rel(ϕ), on s'intéresse à définir de nouvelles relations indicées par des symboles dits *de prédicats* pris dans un ensemble *pred* disjoint de rel(ϕ). Une *définition de symbole de prédicat* n-aire r sera une formule de la forme

$$\forall x_1 \ldots \forall x_n \, (r \, x_1 \ldots x_n \equiv p). \tag{2.6}$$

où p est une formule construite sur l'ensemble de symboles $V \cup$ op$(\phi) \cup$ rel$(\phi) \cup$*pred* ne faisant pas intervenir d'autres occurrences libres de variables que celles de $x_1, \ldots, x_n$. Un ensemble de définitions de prédicats distincts sera un *programme*.

En Prolog IV, où la structure ϕ sera bien entendu π_4, on se limitera à des formules p qui sont des contraintes, c'est-à-dire, on le rappelle, des formules positives existentielles. L'ensemble *pred* sera constitué des couples id$/n$ où id est un identificateur ISO non réservé dans l'arité n (voir figure 2.4 à la page 74).

2.5.2 Programme sous forme clausale

La syntaxe classique d'un programme Prolog reflète plus le fait que le membre droit d'une définition de la forme 2.6 implique le membre gauche que le fait qu'il lui est équivalent. La définition est écrite sous forme d'une ou de plusieurs implications. Dans chacune de ces implications on a coutume de ne pas faire figurer la quantification universelle extérieure et d'adopter une syntaxe de clause.

Une *clause* est une expression de la forme (à droite on donne la syntaxe Prolog IV)

$$r \, s_1 \ldots s_n \;\; :- \;\; p \qquad\qquad \texttt{r}(s_1\texttt{,}\ldots\texttt{,}s_n) \;\; :- \;\; p\texttt{.} \tag{2.7}$$

où r est un prédicat n-aire, les s_i des termes ou des pseudo-termes et p une contrainte construite sur l'ensemble de symboles $V \cup$op$(\pi_4) \cup$rel$(\pi_4) \cup$*pred*. Si p est la contrainte *true* on peut remplacer l'expression 2.7 par $r \, s_1 \ldots s_n$.

Une *forme normale* de la clause 2.7 est une clause de la forme

$$r \, x_1 \ldots x_n \;\; :- \;\; \exists y_1 \ldots \exists y_m \, (x_1 \sim s_1 \wedge \cdots \wedge x_n \sim s_n \wedge p)$$

où $x_1, \ldots, x_n$ sont des variables distinctes, ne figurant ni dans les s_i ni dans p et où les y_i forment un sur-ensemble des variables qui ont des occurrences libres dans p. Lorsque les s_i sont des termes, les signes $\sim$ peuvent bien entendu être remplacés par des signes d'égalité.

Un *paquet* de clauses définissant le prédicat r est une suite non vide de clause, toutes ayant le même prédicat de tête r. La définition du prédicat r représentée par ce paquet est

$$\forall x_1 \ldots \forall x_n \, (r \, x_1 \ldots x_n \;\equiv\; p_1 \vee \cdots \vee p_m)$$

où $(r \, x_1 \ldots x_n \;\; :- \;\; p_1)$, ..., $(r \, x_1 \ldots x_n \;\; :- \;\; p_m)$ est la suite des clauses du paquet mises sous forme normale de façon à ce qu'elles aient toutes le même membre gauche $r \, x_1 \ldots x_n$.

Un *programme sous forme clausale* est une suite de paquets de clauses définissant des prédicats distincts. Le programme représenté par cette suite de paquets de clauses est constitué de l'ensemble des définition de symboles de prédicats $r \in$ *pred* qui sont

1. soit représentés par un paquet de clauses définissant r,
2. soit de la forme $\forall x_1 \ldots \forall x_n\ (r\, x_1 \ldots x_n \quad \equiv \quad \mathit{false})$, lorsque r est un symbole prédicat de qui n'est défini par aucun paquet de clauses.

Par exemple pour définir la somme des éléments d'une liste de trois éléments on introduira la définition du prédicat binaire sigma (écrite avec quelques virgules et parenthèses en plus et le pseudo-terme $a+b+c$),

$$\forall x \forall l\ (\mathrm{sigma}(x,l)\ \equiv\ \exists a \exists b \exists c (x \sim a+b+c \wedge l = \mathtt{.}(a\mathtt{.},\, \mathtt{.}(b,\, \mathtt{.}c,\, \mathtt{[]}))))$$

qui s'écrira sous forme clausale en Prolog IV

```
sigma(X,L) :- A ex B ex C ex
              X = A.+.B.+.C, L = [A,B,C].
```

Après avoir compilé ce programme on pourra poser la requête

```
>> sigma(X,.(1,.(2,.(3,[])))).
X = 6.
```

Si d'une façon générale on veut exprimer la relation qui lie une liste numérique avec la somme de ses éléments on écrira :

$$\forall l \forall x \left(\mathrm{sigma}(x,l) \equiv \left(\begin{array}{c} l = \mathtt{[]} \wedge x = 1 \\ \vee \\ \exists l' \exists y' \exists x' \left(\begin{array}{l} \quad l = \mathtt{.}(y', l') \\ \wedge\ \mathrm{plus}(y', x', x) \\ \wedge\ \mathrm{sigma}(x', l') \end{array} \right) \end{array} \right) \right) \tag{2.8}$$

Pour calculer la somme de 2,3 et 4 on résoudra alors la contrainte

$$\exists l\ (l = \mathtt{.}(2,\, \mathtt{.}(3,\, \mathtt{.}(4,\, \mathtt{[]}))) \ \wedge\ \mathrm{sigma}(x,l))$$

Ceci amènera à concevoir le programme Prolog IV

```
sigma(X,L) :- L = [], X = 0;
              Lp ex Yp ex Xp ex
              L = .(Yp,Lp), plus(X,Yp,Xp), sigma(Xp,Lp).
```

qui peut aussi s'écrire

```
sigma(0,[]).
sigma(X,.(Yp,Lp)) :- plus(X,Yp,Xp), sigma(Xp,Lp).
```

et à lancer la requête

```
>> L ex L = .(2,.(3,.(4,[]))), sigma(X,L).
X = 9.
```

2.5.3 La machine Prolog IV

La définition 2.8 de la relation nommée sigma en fonction d'elle-même nécessite une mise au point.

Ce que l'on cherche à faire c'est de calculer les solutions d'une contrainte p dans la structure π_4 enrichie d'un ensemble de relations définies par un programme $\mathcal{P}$. Cette structure enrichie n'étant pas forcément unique, on ne peut, à vrai dire, résoudre p dans «la structure enrichie». Il faut supposer qu'il existe une contrainte q telle que

1. q ne fasse intervenir que des symboles de π_4,
2. $\mathcal{P} \cup \mathcal{T}(\pi_4) \quad \models \quad p \equiv q$,

et résoudre q dans la structure π_4. On est ramené au problème évoqué à la page 79. On s'attachera donc à calculer une contrainte q qui satisfait les points 1 et 2 précédents et dont les solutions sont le plus «visibles» possible. Comme nous l'avons aussi mentionné, on ne pourra assurer que q soit la contrainte *false* si q n'a aucune solution dans π_4. On assurera que q soit la contrainte *false* si q n'a de solutions dans aucune des structures ϕ qui satisfonts les axiomes $\mathcal{T}(\pi_4)$, c'est-à-dire si $\mathcal{T}(\pi_4) \quad \models \quad q \equiv \mathit{false}$.

La façon dont q est calculée à partir de p est l'essence même du déroulement d'un programme P à partir de la requête p. Nous allons décrire ce déroulement à l'aide d'une machine abstraite dite *machine Prolog IV* formalisée à l'aide d'un ensemble de règles de réécriture de sous-formules en sous-formules permettant de transformer p en q.

Appelons *conjonction existentielle* une contrainte de la forme

$$\exists x_1 \ldots \exists x_k\, (a_1 \wedge \cdots \wedge a_l),$$

où les a_i sont des contraintes atomiques et appelons *contrainte résolue dans* $\mathcal{T}(\pi_4)$ une contrainte p qui ne fait pas intervenir d'autres symboles que ceux de π_4 et qui est obligatoirement la contrainte *false* si $\mathcal{T}(\pi_4) \quad \models \quad p \equiv \mathit{false}$.

Machine Prolog IV On se donne une contrainte p et un programme $\mathcal{P}$ et l'on veut calculer une contrainte q telle que :

1. $\mathcal{P} \cup \mathcal{T}(\pi_4) \quad \models \quad p \equiv q$,
2. q est de la forme $m_1 \vee \ldots \vee m_n \vee \mathit{false}$ avec n éventuellement nul,
3. chaque m_i est une conjonction existentielle résolue dans $\mathcal{T}(\pi_4)$.

On part de la contrainte

$$\langle \mathit{true} \rangle \wedge (p \vee \mathit{false}), \tag{2.9}$$

les chevrons servant uniquement à délimiter des occurrences de sous-formules plus importantes que d'autres. On applique autant de fois que possibles les règles de réécriture qui suivent sur l'occurrence la plus à gauche possible d'une expression de la forme $\langle m \rangle$. Si le processus s'arrête on obtient une formule de la forme

$$\langle m_1 \rangle \vee \cdots \vee \langle m_n \rangle \vee \langle \mathit{false} \rangle,$$

chaque m_i étant une conjonction existentielle. La contrainte $m_1 \vee \cdots \vee m_n \vee$ *false*, est alors la contrainte recherchée q. Si le processus ne s'arrête pas c'est que le programmeur a posé une mauvaise requête ou écrit un mauvais programme !

Règles de réécriture

1	$\exists x\, \langle \mathit{false} \rangle$	$\Longrightarrow$	$\langle \mathit{false} \rangle$
2	$\langle \mathit{false} \rangle \vee p$	$\Longrightarrow$	p
3	$\langle \mathit{false} \rangle \wedge p$	$\Longrightarrow$	$\langle \mathit{false} \rangle$
4	$\exists x\, \langle m \rangle$	$\Longrightarrow$	$\langle \text{quasisolved}(\exists x\, m) \rangle$
5	$\exists x\, (\langle m \rangle \vee p)$	$\Longrightarrow$	$(\exists x\, \langle m \rangle) \vee (\exists x\, p)$
6	$(\langle m \rangle \vee p) \vee q$	$\Longrightarrow$	$\langle m \rangle \vee (p \vee q)$
7	$(\langle m \rangle \vee p) \wedge q$	$\Longrightarrow$	$(\langle m \rangle \wedge p) \vee (p \wedge q)$
8	$\langle m \rangle \wedge (p \vee q)$	$\Longrightarrow$	$(\langle m \rangle \wedge p) \vee (\langle m \rangle \wedge q)$
9	$\langle m \rangle \wedge (p \wedge q)$	$\Longrightarrow$	$(\langle m \rangle \wedge p) \wedge q$
10	$\langle m \rangle \wedge \exists x p(x)$	$\Longrightarrow$	$\exists x'(\langle m \rangle \wedge p(x'))$
11	$\langle m \rangle \wedge a$	$\Longrightarrow$	$\langle \text{quasisolved}(m \wedge a) \rangle$
12	$\langle m \rangle \wedge r\, s_1 \ldots s_n$	$\Longrightarrow$	$\langle m \rangle \wedge \exists x_1 \ldots \exists x_n \left(\begin{pmatrix} x_1 \sim s_1 \\ \wedge \ldots \wedge \\ x_n \sim s_n \end{pmatrix} \wedge \text{body}(r\, x_1 \ldots x_n) \right)$

avec les conventions suivantes

– $p, p(x), q$ sont des contraintes quelconques et $p(x')$ est la contrainte $p(x)$ dans laquelle on a substitué la variable x' à chaque occurrence libre de la variable x,

– x, x' sont des variables telles que dans la règle 10, ni m, ni $p(x)$, n'ait d'occurrence libre de x'.

– a est une contrainte atomique ne faisant pas intervenir d'autres symboles que ceux de π_4,

– m est une conjonction existentielle résolue dans $\mathcal{T}(\pi_4)$ et distincte de *false*,

– quasisolved(r) est une conjonction existentielle résolue dans $\mathcal{T}(\pi_4)$ telle que
$\mathcal{T}(\pi_4) \quad \models \quad r \equiv \text{quasisolved}(r)$,

– r est un symbole de prédicat, les s_i sont des termes ou des pseudo-termes et les x_i sont des variables ne figurant pas dans les s_i,

– body$(r\, x_1, \ldots, x_n)$ est une contrainte telle que, au nom des variables près, la proposition qui suit appartienne à $\mathcal{P}$,
$\forall x_1 \ldots \forall x_n\ (r\, x_1, \ldots, x_n \equiv \text{body}(r\, x_1, \ldots, x_n))$.

Les règles 7 et 8 distribuent le connecteur $\wedge$ sur le connecteur $\vee$ pour obtenir une disjonction de conjonctions. C'est la façon la plus simpliste de résoudre une contrainte booléenne et la mise en œuvre de cette distribution échoit à la machinerie « backtrackante » que l'on retrouve dans toutes les implantations de Prolog. Par exemple la résolution de la contrainte $(x = 1 \vee x = 2 \vee x = 3) \wedge (y = 1 \vee y = 2 \vee y = 3)$ produit :

```
>> (X=1; X=2; X=3), (Y=1; Y=2; Y=3).
Y = 1, X = 1;
Y = 2, X = 1;
Y = 3, X = 1;
Y = 1, X = 2;
Y = 2, X = 2;
Y = 3, X = 2;
Y = 1, X = 3;
Y = 2, X = 3;
Y = 3, X = 3.
```

C'est l'occasion de rappeler que les variables sont ou commencent par des majuscules, que la virgule remplace le connecteur $\wedge$, le point virgule le connecteur $\vee$, et que le point marque la fin de la contrainte. Cette mise sous forme disjonctive est hautement combinatoire puisque la taille de la formule finale peut être une fonction exponentielle de la taille de la formule de départ. Bien entendu toute explosion combinatoire est de la responsabilité du programmeur !

Les règles 6 et 9 privilégient l'association gauche-droite pour les conjonctions et les disjonctions. Les règles 5 et 10 déplacent les quantificateurs aux bons niveaux en augmentant leur portée dans les conjonctions et en les distribuant sur les disjonctions. Ces règles de réécriture ont un effet relativement mineur qui peut être simulé en utilisant un codage standard pour les conjonctions, en renommant certaines variables et en notant celles qui ne sont pas quantifiées.

Les règles 1,2,3,4 et 11 simplifient la formule en faisant appel à un «solveur» dans la théorie $\mathcal{T}(\pi_4)$. Par exemple pour la contrainte $x > y \wedge (x = 1 \vee x = 2 \vee x = 3) \wedge (y = 1 \vee y = 2 \vee y = 3)$ on obtient :

```
>> gt(X,Y), (X=1; X=2; X=3), (Y=1; Y=2; Y=3).
Y = 1, X = 2;
Y = 1, X = 3;
Y = 2, X = 3.
```

et pour la contrainte $\exists x \exists y\ (z = [x, y] \wedge x > y \wedge (x = 1 \vee x = 2 \vee x = 3) \wedge (y = 1 \vee y = 2 \vee y = 3))$ on obtient :

```
>> X ex Y ex Z = [X,Y], gt(X,Y),
     (X =1; X=2; X=3), (Y=1; Y=2; Y=3).
Z = [1,1];
Z = [2,1];
Z = [3,1];
Z = [3,2];
Z = [3,3].
```

Enfin, à peu de choses près, la règle 12 remplace le membre gauche d'une définition de prédicat par son membre droit ainsi que le ferait l'expansion d'une macro-définition.

2.6 Exemples de programmes en Prolog IV

Pour conclure voici un ensemble d'exemples diversifiés de programme Prolog IV qui, on l'espère, donnera un bon aperçu des fonctionnalités du langage : le traitement de listes, le traitement d'entiers et de booléens, la résolution de contraintes générales sur les nombres réels et la résolution de contraintes linéaires.

2.6.1 Contraintes sur les listes

Parmi les contraintes Prolog IV portant sur les listes deux retiennent particulièrement l'attention : la concaténation et la sélection du ième élément d'une liste. Les 4 premiers exemples sont là pour illustrer ce point.

Traduction bidirectionnelle de nombres arabes en nombres romains

Ce premier exemple montre l'intérêt de disposer d'une contrainte de concaténation de liste, même avec des axiomes aussi pauvres que les axiomes 12 et 13 de la page 84. Il s'agit de traduire la notation décimale, en fait arabe, d'un entier en sa notation romaine et vice-versa. L'entier arabe est écrit sous forme d'une liste de chiffres et l'entier romain sous forme d'une liste de caractères pris dans 'I', 'V', 'X', 'L', 'C', 'D', 'M' et '?'. Ces caractères représentent respectivement 1, 5, 10, 50, 100, 500, 1000 et 5000. Conformément à la norme ISO ils sont représentés par des identificateurs d'une lettre écrits entre apostrophes. Voici le programme.

```
araberomain(U,W) :-
    tr(U,['?','M','D','C','L','X','V','I'], W).

tr([], S, []).
tr(U o [0], S o [V,I], W) :-
                  dif([0] o U, U o [0]), tr(U, S, W).
tr(U o [1], S o [V,I], W o [I]) :-       tr(U, S, W).
tr(U o [2], S o [V,I], W o [I,I]) :-     tr(U, S, W).
tr(U o [3], S o [V,I], W o [I,I,I]) :-   tr(U, S, W).
tr(U o [4], S o [V,I], W o [I,V]) :-     tr(U, S, W).
tr(U o [5], S o [V,I], W o [V]) :-       tr(U, S, W).
tr(U o [6], S o [V,I], W o [V,I]) :-     tr(U, S, W).
tr(U o [7], S o [V,I], W o [V,I,I]) :-   tr(U, S, W).
tr(U o [8], S o [V,I], W o [V,I,I,I]) :- tr(U, S, W).
tr(U o [9], S o [X,V,I], W o [I,X]) :-   tr(U, S o [X], W).
```

Rappelons que la notation `X o Y` remplace `conc(X,Y)`. Les requêtes qui suivent traduisent des nombres de l'arabe vers le romain puis les retraduisent en arabe.

```
>> araberomain([1,2,3,4],W), araberomain(U,W).
U = [1,2,3,4],
W = ['M','C','C','X','X','X','I','V'].

>> araberomain([5,6,7,8],W), araberomain(U,W).
U = [5,6,7,8],
W = [?,'D','C','L','X','X','V','I','I','I'].

>> araberomain([9,9,9],W), araberomain(U,W).
U = [9,9,9],
W = ['C','M','X','C','I','X'].
```

Problème de coloration de Shur

Dans une concaténation, la taille de la liste obtenue est la somme des tailles des listes concaténées. Ce deuxième exemple illustre le début d'arithmétique caché dans ce fait. Il concerne un problème de coloration liée à une propriété connue sous le nom de lemme de I. Shur.

On se donne m couleurs et un entier positif n. Il s'agit d'attribuer à chaque entier i de la suite finie $1, 2, \ldots, n$, une couleur x_i prise parmi les m couleurs de façon à ce que pour tout i, j, k pris entre 1 et n on ait

$i + j = k$ entraîne les couleurs x_i, x_j, x_k ne sont pas toutes identiques.

I. Shur a montré que quel que soit m il existe toujours un entier n suffisamment grand pour lequel ce problème n'a pas de solution.

Intéressons nous au cas particulier où $m = 3$, les couleurs étant a, b, c. Le coloriage $[x_1, \ldots, x_n]$ se définit récursivement sur n par :

```
coloriage([]).
coloriage(X o [C]):-
   coloriage(X),
   ajoutcorrect(X, C),
   couleur(C).

ajoutcorrect([], C).
ajoutcorrect([A], C) :-
   dif(A, C).
ajoutcorrect([A] o X o [B], C) :-
   ajoutcorrect(X, C),
   dif([A,B], [C,C]).

couleur(a).
couleur(b).
couleur(c).
```

Pour $n = 13$, on obtient les 18 coloriages :

```
>> size(X) = 13, coloriage(X).
X = [a,b,b,a,c,c,a,c,c,a,b,b,a];
X = [a,b,b,a,c,c,b,c,c,a,b,b,a];
X = [a,b,b,a,c,c,c,c,c,a,b,b,a];
X = [a,c,c,a,b,b,a,b,b,a,c,c,a];
X = [a,c,c,a,b,b,b,b,b,a,c,c,a];
X = [a,c,c,a,b,b,c,b,b,a,c,c,a];
X = [b,a,a,b,c,c,a,c,c,b,a,a,b];
X = [b,a,a,b,c,c,b,c,c,b,a,a,b];
X = [b,a,a,b,c,c,c,c,c,b,a,a,b];
X = [b,c,c,b,a,a,a,a,a,b,c,c,b];
X = [b,c,c,b,a,a,b,a,a,b,c,c,b];
X = [b,c,c,b,a,a,c,a,a,b,c,c,b];
X = [c,a,a,c,b,b,a,b,b,c,a,a,c];
X = [c,a,a,c,b,b,b,b,b,c,a,a,c];
X = [c,a,a,c,b,b,c,b,b,c,a,a,c];
X = [c,b,b,c,a,a,a,a,a,c,b,b,c];
X = [c,b,b,c,a,a,b,a,a,c,b,b,c];
X = [c,b,b,c,a,a,c,a,a,c,b,b,c].
```

et à partir de $n = 14$ on n'obtient plus de coloriages :

```
>> size(X) = 14, coloriage(X).
false.
```

Transposition de matrice par concaténations

Ecrire un programme de transposition de matrice en Prolog est un exercice plus difficile qu'il ne semble à première vue. La matrice se représente naturellement comme un liste de ligne, chaque ligne étant une liste de nombres. La difficulté provient du fait qu'il faut aussi pouvoir accéder aux colonnes. Pour ceci on prévoit un prédicat `colonneun/3` qui permet d'obtenir la première colonne et le reste de la matrice amputée de celle-ci. Il faut aussi pouvoir parler de la matrice formée d'une liste de lignes toutes vides. C'est là qu'intervient le prédicat `zerocolonnes/1` qui utilise une contrainte de concaténation produisant une liste d'éléments identiques, ici des listes vides. Par symétrie nous avons aussi introduit les prédicats plus faciles `ligneun/3` et `zerolignes/1`. Voici le programme :

```
transposition(A,B) :-
   zerolignes(A),
   zerocolonnes(B).
transposition(A,B) :-
   ligneun(A,X,C),
   colonneun(B,X,D),
   transposition(C,D).

zerolignes([]).

zerocolonnes(B) :-
   conc([[]],B) = conc(B,[[]]).

ligneun([X|A],X,A).

colonneun([],[],[]).
colonneun([[Aaa|Aa]|A],[Aaa|X],[Aa|B]) :-
   colonneun(A,X,B).
```

et voici une requête qui transpose une matrice et la retranspose pour obtenir la matrice d'origine.

```
>> transposition([[1,3,5],[2,4,6]],B),
    transposition(B,A).
A = [[1,3,5],[2,4,6]],
B = [[1,2],[3,4],[5,6]].
```

Transposition de matrice par indiçages

Qui pense à la transposition d'une matrice pense à la formule $a_{ij} = b_{ji}$. Voici un deuxième programme de transposition reposant sur cette formule :

```
transpositionbis(A,B) :-
   M = size(A),
   contraintes(M,c0,[A,N]),
   N = size(B),
   contraintes(N,c0,[B,M]),
   contraintes(M,c1,[A,B]).

contraintes(0,A,L).
contraintes(I,A,L) :-
   ge(I,1),
   contrainte(A,[I|L]),
   contraintes(I.-.1,A,L).

contrainte(c0,[I,A,N]) :-
   N = size(A:I).
contrainte(c1,[I,A,B]) :-
   N = size(A:I),
   contraintes(N,c2,[I,A,B]).
contrainte(c2,[J,I,A,B]) :-
   A:I:J = B:J:I.
```

L'écriture `A:I:J` est interprétée comme `((A:I):J)` qui elle-même est interprétée comme le pseudo-terme `index(index(A,I),J)`.

```
>> transpositionbis([[1,2],[3,4],[5,6]],B),
    transpositionbis(B,A).
A = [[1,2],[3,4],[5,6]],
B = [[1,3,5],[2,4,6]].
```

2.6.2 Contraintes sur les entiers et les booléens

Dans de nombreux problèmes combinatoires les entiers et les booléens, en fait les entiers 0 et 1, jouent un rôle de premier plan. En voici la preuve à travers quelques exemples.

Enumération d'entiers naturels

Ce programme énumère, sans en oublier aucune, toutes les listes d'entiers naturels, c'est-à-dire d'entiers positifs ou nuls. Cet ensemble de liste étant infini, le programme ne s'arrête que s'il est appelé dans un contexte qui borne ces entiers. L'énumération est accélérée par les conflits entre les nombreuses contraintes posées et ce contexte. Voici le programme :

```
enumeration(L) :- nonnegatifs(L), depuis(0,L).

nonnegatifs([]).
nonnegatifs([N|L]) :-
   int(N),
   ge(N,0),
   nonnegatifs(L).

depuis(N,[]).
depuis(N,La) :-
   dif(La,[]),
   superieurs(N,La,Lb),
   depuis(N.+.1,Lb).

superieurs(N,[],[]).
superieurs(N,[N|La],Lb) :-
   superieurs(N,La,Lb).
superieurs(N,[Na|La],[Na|Lb]) :-
   lt(N,Na),
   superieurs(N,La,Lb).
```

et voici une énumération sous contexte :

```
>> X ~ cc(2,5), Y ~ cc(1,4), Z ~ cc(3,6),
   Z = plus(X,Y), enumeration([X,Y,Z]).
Z = 3, Y = 1, X = 2;
Z = 4, Y = 1, X = 3;
Z = 5, Y = 1, X = 4;
Z = 6, Y = 1, X = 5;
Z = 4, Y = 2, X = 2;
Z = 5, Y = 3, X = 2;
Z = 6, Y = 4, X = 2;
Z = 5, Y = 2, X = 3;
Z = 6, Y = 2, X = 4;
Z = 6, Y = 3, X = 3.
```

Pour information voici un autre programme d'énumération, qui peut être plus efficace, mais qui fait appel à deux meta-prédicats prédéfinis `glb/2` et `int_size/2` décrits dans la partie « prédicats prédéfinis Prolog IV » et au prédicat prédéfini ISO de coupure d'espace de recherche. Dans ce programme on suppose que les éléments à énumérer ont été préalablement contraints à être des entiers.

```
metaenumeration([]).
metaenumeration([Xa|X]) :-
   pluspetit([Xa|X],Xa,Y,Ya),
   glb(Ya,M),
   suite(Ya,M),
   metaenumeration(Y).

suite(Ya,Ya).
suite(Ya,M) :- gt(Ya,M).

pluspetit([Xa|X],Xb,Y,Ya) :-
   int_size(Xa,1),!,
   pluspetit(X,Xb,Y,Ya).
pluspetit([Xa|X],Xb,[Xa|Y],Yb) :-
   glb(Xa,M),
   glb(Xb,N),
   lt(M,N),!,
   pluspetit(X,Xa,Y,Yb).
pluspetit([Xa|X],Xb,[Xa|Y],Yb) :-
   pluspetit(X,Xb,Y,Yb).
pluspetit([],Xb,[],Xb).
```

```
>> X ~ int, Y ~ int, Z ~ int,
   X ~ cc(2,5), Y ~ cc(1,4), Z ~ cc(3,6),
   Z = plus(X,Y), metaenumeration([X,Y,Z]).
Z = 3, Y = 1, X = 2;
Z = 4, Y = 1, X = 3;
Z = 5, Y = 1, X = 4;
Z = 6, Y = 1, X = 5;
Z = 4, Y = 2, X = 2;
Z = 5, Y = 3, X = 2;
Z = 6, Y = 4, X = 2;
Z = 5, Y = 2, X = 3;
Z = 6, Y = 2, X = 4;
Z = 6, Y = 3, X = 3.
```

Suite magique

Etant donné n, on s'intéresse à trouver les suites finies x de la forme $(x_0, \ldots, x_{n-1})$ telles que chaque x_i soit le nombre d'occurrences de l'entier i dans x. Les solutions sont les solutions de la conjonction $P_1(x) \wedge \cdots \wedge P_n(x)$ de contraintes

$$P_i(x): \quad x_{i-1} = \sum_{j=0}^{n-1} (x_j = i-1),$$

où l'expression $(x_j = i-1)$ vaut 1 ou 0 suivant que l'égalité reprÝsentée est ou n'est pas vérifiée. Voici le programme qui fait appel au programme précédent d'énumération d'entiers :

```
suitemagique(X) :-
   npremierescontraintes(size(X),X),
   enumeration(X).

npremierescontraintes(0,X).
npremierescontraintes(I,X) :-
   gt(I,0),
   J = I.-.1,
   nboccurrences(J,X,X:I),
   npremierescontraintes(J,X).

nboccurrences(A,[],0).
nboccurrences(A,[B|X],M) :-
   M = N.+.beq(A,B),
   nboccurrences(A,X,N).
```

et voici quelques calculs de solutions :

```
>> size(X) = 4, suitemagique(X).
X = [2,0,2,0];
X = [1,2,1,0].
>> size(X) = 5, suitemagique(X).
X = [2,1,2,0,0].
>> size(X) = 6, suitemagique(X).
false.
>> size(X) = 10, suitemagique(X).
X = [6,2,1,0,0,0,1,0,0,0].
```

Problème des 21 carrés connus

On se donne n carrés de dimensions $a_1 \times a_1, \ldots, a_n \times a_n$ et on se propose de les placer dans un carré de dimensions $m \times m$. Les données sont $n = 21$, $m = 112$ et le n-uplet $(a_1, \ldots, a_n)$ est égal à

$(550, 42, 37, 35, 33, 29, 27, 25, 24, 19, 18, 17, 16, 15, 11, 9, 8, 7, 6, 4, 2)$.

Les solutions sont les solutions de toutes les contraintes qui suivent. La contrainte 3 élimine l'essentiel des symétries en contraignant le premier carré à avoir son coin inférieur gauche à l'intérieur du quart inférieur gauche du grand carré et sous la diagonale positive de ce quart. Les contraintes de la forme 4a et 4b sont redondantes mais sont fondamentales pour augmenter la propagation d'informations. L'esprit général de ce jeu de contraintes respectent les idées exposées dans un papier de N. Beldiceanu.

1a	entier$(x_j) \wedge x_j \in [0, m - a_j]$,	pour $j = 1, \ldots, n$,
1b	entier$(y_j) \wedge y_i \in [0, m - a_j]$,	pour $j = 1, \ldots, n$,
2	$(x_j + a_j \leq x_k \vee x_k + a_k \leq x_j) \vee$ $(y_j + a_j \leq y_k \vee y_k + a_k \leq y_j)$,	pour $j = 1, \ldots, n$, pour $k = j + 1, \ldots, n$,
3	$x_j \leq \frac{1}{2}(m - a_j) \ \wedge \ y_j \leq x_j$	pour $j = 1$,
4a	$m = \sum_{j=1}^{n} a_j \times (i \in (x_j, x_j + a_j])$,	pour $i = 1, \ldots, m$,
4b	$m = \sum_{j=1}^{n} a_j \times (i \in (y_j, y_j + a_j])$,	pour $i = 1, \ldots, m$

Voici le programme qui fait appel à nos programmes précédents de transposition et d'énumération d'entiers.

```
remplissage(F) :-
   M = 112,
   N = size(A),
   A = [50,42,37,35,33,29,27,25,24,
       19,18,17,16,15,11,9,8,7,6,4,2],
   contraintes(N,c1,[X,A,M]),
   contraintes(N,c1,[Y,A,M]),
   contraintes(N,c2,[X,Y,A]),
   contraintes(1,c3,[X,Y,A,M]),
   contraintes(M,c4,[X,A,M]),
   contraintes(M,c4,[Y,A,M]),
   transposition([X,Y,A],F),
   enumeration(X),
   enumeration(Y).

% Creation de conjunctions de contraintes

contraintes(0,C,L).
contraintes(I,C,L) :-
   ge(I,1),
   contrainte(C,[I|L]),
   contraintes(I.-.1,C,L).

% Les différents types de contraintes

contrainte(c1,[I,X,A,M]) :-
   Xi = index(X,I), Ai = index(A,I),
   Xi ~ int, Xi ~ cc(0,M.-.Ai).

contrainte(c2,[J,X,Y,A]) :-
   contraintes(minus(J,1),c2bis,[J,X,Y,A]).
contrainte(c2bis,[K,J,X,Y,A]) :-
   Xj = index(X,J), Yj = index(Y,J), Aj = index(A,J),
   Xk = index(X,K), Yk = index(Y,K), Ak = index(A,K),
   or(boutoo(Xk.-.Xj,.-.Ak,Aj),boutoo(Yk.-.Yj,.-.Ak,Aj)).

contrainte(c3,[J,X,Y,A,M]) :-
   Xj = index(X,J), Yj = index(Y,J), Aj = index(A,J),
   le(Xj,(M.-.Aj)./.2), le(Yj,Xj).

contrainte(c4,[I,[],[],0]).
contrainte(c4,[I,[Xa|X],[Aa|A],Ma.+.M]) :-
   Ma = boc(I,Xa,Xa.+.Aa).*.Aa,
   contrainte(c4,[I,X,A,M]).
```

Attention pour exécuter la requête qui suit il faut avoir lancé Prolog IV avec en paramètres

```
-heap 4000000 -local 40000 -choice 200000
```

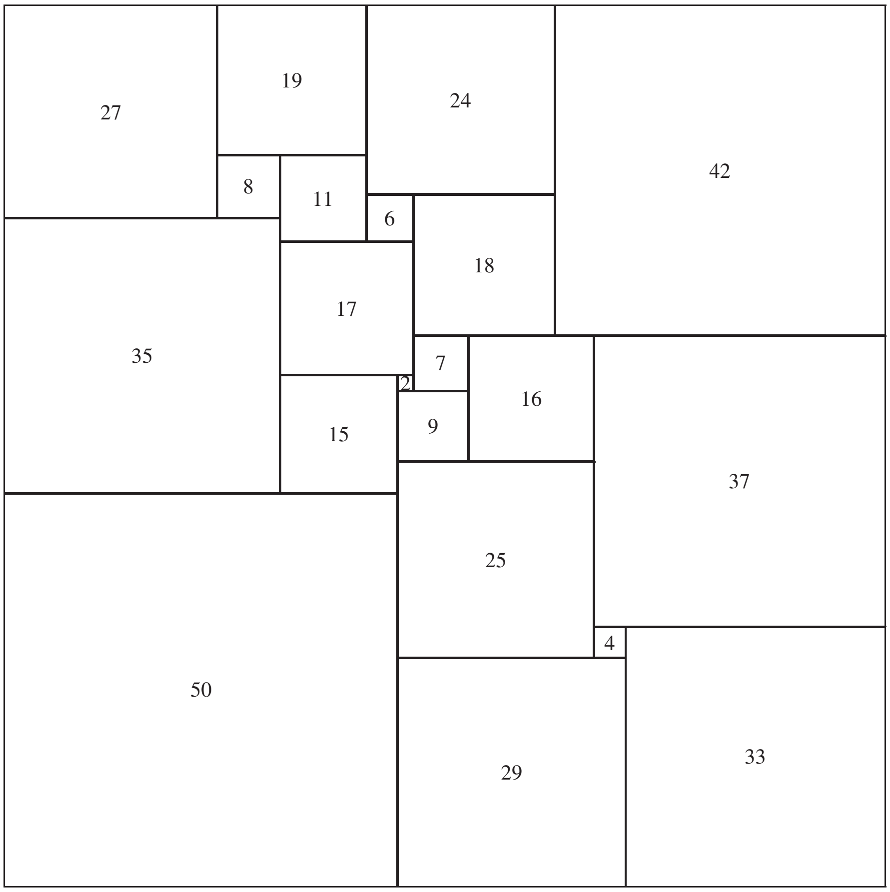
Pour plus d'information sur ces paramètres, voir la partie « environnement général » du manuel.

```
>> remplissage(F).

F = [[0,0,50],[70,70,42],[75,33,37],[0,50,35],
[79,0,33],[50,0,29],[0,85,27],[50,29,25],
[46,88,24],[27,93,19],[52,70,18],[35,65,17],
[59,54,16],[35,50,15],[35,82,11],[50,54,9],
[27,85,8],[52,63,7],[46,82,6],[75,29,4],
[50,63,2]]

F = [[0,0,50],[70,70,42],[33,75,37],[50,0,35],
[0,79,33],[0,50,29],[85,0,27],[29,50,25],
[88,46,24],[93,27,19],[70,52,18],[65,35,17],
[54,59,16],[50,35,15],[82,35,11],[54,50,9],
[85,27,8],[63,52,7],[82,46,6],[29,75,4],
[63,50,2]]
```

On obtient deux solutions symétriques qui se résument au dessin qui suit.



2.6.3 Contraintes sur les réels

Passons au traitement du continu en isolant des nombres algébriques.

Racines des polynomes de Wilkinson

L'exemple qui suit est un benchmark pour mettre à rude épreuve le calcul des racines d'un polynôme, surtout si ce calcul est fait avec des nombres binaires flottants IEEE simple précision. On considère le polynome dit de Wilkinson

$W(x) : (x + 1)(x + 2) \ldots (x + 20)$

qui bien entendu admet 20 racines. Si on l'altère légèrement en considérant le polynôme

$W'(x)$: $\mathrm{W}(x) + 2^{-23}x^{19}$

on ne doit plus que trouver 10 racines. Voyons si Prolog IV est à la hauteur.

```
wilkinson(Y, X) :-
   produitmonomes(Y, X, 20).

produitmonomes(1, X, 0).
produitmonomes(Y.*.(X.+.N), X, N) :-
   ge(N, 1),
   produitmonomes(Y, X, N.-.1).

wilkinsonbis(Y.+.power(1/2,23).*.power(X,19), X) :-
  wilkinson(Y, X).
```

On fait alors usage du prédicat prédéfini `realsplit1`/ décrit dans la partie « prédicats prédéfinis de Prolog IV » du manuel. On onbtient les 20 racines par :

```
>> wilkinson(0, X), realsplit([X]).
X = -20;
X = -19;
X = -18;
X = -17;
X = -16;
X = -15;
X = -14;
X = -13;
X = -12;
X = -11;
X = -10;
X = -9;
X = -8;
X = -7;
X = -6;
X = -5;
X = -4;
X = -3;
X = -2;
X = -1.
```

Dans le cas du polynôme altéré on obtient une multitude de racines apparemment parasites, mais si on limite x à être dans l'intervalle $[-100, 100]$, on trouve bien 10 encadrements :

```
>> X ~ cc(-100,100), wilkinsonbis(0, X),
       realsplit([X]).
X ~ cc(-`>20.846977`,-`<20.84688`);
X ~ cc(-`>8.917252`,-`>8.917247`);
X ~ oo(-`>8.007267`,-`<8.007267`);
X ~ oo(-`<6.999698`,-`>6.999697`);
X ~ cc(-`>6.000007`,-`<6.000007`);
X ~ oo(-5,-`>4.9999995`);
X ~ oo(-`>4.0000004`,-4);
X ~ oo(-3,-`>2.9999997`);
X ~ oo(-`>2.0000002`,-2);
X ~ oo(-1,-`>0.9999999`).
```

Racines confirmées du polynome altéré de Wilkinson

Les 10 encadrements précédents garantissent qu'en dehors de ceux-ci il n'y a pas de racines du polynôme altéré mais ne garantissent pas que dans chaque encadrement il y a en exactement une. Il pourrait n'y en avoir aucune ou il pourrait y en avoir plusieurs. Pour lever ce doute Michel Van Caneghem vérifie que dans chaque encadrement

1. la dérivée du polynôme altéré ne s'annule pas et
2. les valeurs du polynôme altéré, prises à ses bornes, sont de signes contraires.

Il est donc nécessaire de programmer le calcul de la dérivée du polynôme altéré par le prédicat `dwilkinsonbis/2`.

```
dwilkinsonbis(Y.+.19.*.power(1/2,23).*.power(X,18), X) :-
   dwilkinson(Y, X).

dwilkinson(Y, X) :-
   dproduitmonomes(Y, X, 20).

dproduitmonomes(0, X, 0).
dproduitmonomes(Y.+.Z.*.(X.-.N), X, N) :-
   ge(N, 1),
   produitmonomes(Y, X, N.-.1),
   dproduitmonomes(Z, X, N.-.1).
```

On peut alors lancer une requête plus élaborée, qui chaque fois qu'en sortie la variable `Certain` prend la valeur `oui`, garantit l'existence et l'unicité de la racine dans son encadrement.

```
>> X ~ cc(-100,100),
   wilkinsonbis(0, X),
   realsplit([X]),
   Epsilon ex Xa ex Xb ex Xc ex Ya ex Yb ex Yc ex
   Epsilon = 1./.1000,
   Xa = X.-.Epsilon,
   Xc = X.+.Epsilon,
   Xb ~ cc(Xa, Xc),
   wilkinsonbis(Ya, Xa),
   wilkinsonbis(Yc, Xc),
   dwilkinsonbis(Yb, Xb),
   Certain = if(blt(Ya.*.Yc,0).*.bdif([Yb],[0]),oui,non).
Certain = oui, X ~ cc(-'>20.846977',-'<20.84688');
Certain = oui, X ~ cc(-'>8.917252',-'>8.917247');
Certain = oui, X ~ oo(-'>8.007267',-'<8.007267');
Certain = oui, X ~ oo(-'<6.999698',-'>6.999697');
Certain = oui, X ~ cc(-'>6.000007',-'<6.000007');
Certain = oui, X ~ oo(-5,-'>4.9999995');
Certain = oui, X ~ oo(-'>4.0000004',-4);
Certain = oui, X ~ oo(-3,-'>2.9999997');
Certain = oui, X ~ oo(-'>2.0000002',-2);
Certain = oui, X ~ oo(-1,-'>0.9999999').
```

2.6.4 Contraintes linéaires

Les deux exemples qui suivent rappellent qu'en Prolog IV on dispose des mêmes contraintes linéaires qu'en Prolog III à condition d'utiliser d'utiliser des pseudo-opérations et des relations dont les noms se terminent par `lin`.

Produit matriciel

Le premier exemple est la quintessence du linéaire sans relation d'ordre. Il s'agit d'exprimer la contrainte $A \times B = C$, où A, B, C sont des matrices dimensionnées correctement pour la multiplication matricielle, deux d'entre elles au moins étant de dimensions connues. Voici le programme :

```
matricefoismatrice(A,B,C) :-
   zerolignes(A),
   zerolignes(C).
matricefoismatrice(A,B,C) :-
   ligneun(A,X,Ap),
   ligneun(C,Y,Cp),
   size(X) = size(B),
   vecteurfoismatrice(X,B,Y),
   matricefoismatrice(Ap,B,Cp).

vecteurfoismatrice(X,B,[]) :-
   zerocolonnes(B).
vecteurfoismatrice(X,B,[R|Y]) :-
   colonneun(B,Z,Bp),
   vecteurfoisvecteur(X,Z,R),
   vecteurfoismatrice(X,Bp,Y).

vecteurfoisvecteur([],[],0).
vecteurfoisvecteur([R|X],[S|Y],R*S+T) :-
   vecteurfoisvecteur(X,Y,T).

zerolignes([]).

zerocolonnes(B) :-
   conc([[]],B) = conc(B,[[]]).

ligneun([X|A],X,A).

colonneun([],[],[]).
colonneun([[Aaa|Aa]|A],[Aaa|X],[Aa|B]) :-
   colonneun(A,X,B).
```

Les notations `R*S` et `R+T` correspondent aux pseudo-termes `timeslin(R,S)` et `pluslin(R,T)`. On peut alors lancer la requête :

```
>> A ex B ex C ex
   A = [[1,2],[3,4],[5,6]],
   B = [[1,3,5],[2,4,6]],
   C = [[5,11,17],[11,25,39],[17,39,61]],
   matricefoismatrice(Ap,B,C),
   matricefoismatrice(A,Bp,C),
   matricefoismatrice(A,B,Cp).

Cp = [[5,11,17],[11,25,39],[17,39,61]],
Bp = [[1,3,5],[2,4,6]],
Ap = [[1,2],[3,4],[5,6]].
```

Problème des 9 carrés inconnus

Le deuxième exemple a fait le succès de Prolog III[8]. Il s'agit d'assembler n carrés de tailles ditinctes pour former un rectangle. Les tailles des carrés et les tailles du rectangle sont inconnues. Seul l'entier n est connu et vaut 9. Voici la transposition du programme Prolog III en Prolog IV. C'est un bon exercice de syntaxe.

```
rectangleRempli(A, C) :-
   gelin(A,1),
   carresDistincts(C),
   zoneRemplie([-1,A,1], L, C, []).

carresDistincts([]).
carresDistincts([B|C]) :-
   gtlin(B,0),
   carresDistincts(C),
   horsDe(B, C).

horsDe(B, []).
horsDe(B, [Bp|C]) :-
   dif(B, Bp),
   horsDe(B, C).

zoneRemplie([V|L], [V|L], C, C) :-
   gelin(V, 0).
zoneRemplie([V|L], Lppp, [B|C], Cpp) :-
   lt(V, 0),
   carrePlace(B, L, Lp),
   zoneRemplie(Lp, Lpp, C, Cp),
   zoneRemplie([V+B,B|Lpp], Lppp, Cp, Cpp).

carrePlace(B, [H,0,Hp|L], Lp) :-
   gtlin(B, H),
   carrePlace(B, [H+Hp|L], Lp).
carrePlace(B, [H,V|L], [-B+V|L]) :-
   B = H.
carrePlace(B, [H|L], [-B,H-B|L]) :-
   ltlin(B, H).
```

Et voici les 8 solutions du problème, qui en fait sont au nombre de 2 si on tient compte des symétries. Ici `A` est la longueur du rectangle trouvé, sa largeur étant supposée valoir 1, et `C` est la liste des tailles des 9 carrés.

8. Alain Colmerauer. An Introduction to Prolog III, Communications of the ACM, 33(7):68-90, 1990.

```
>> size(C)=9, rectangleRempli(A, C).
A = 33/32,
C = [15/32,9/16,1/4,7/32,1/8,7/16,1/32,5/16,9/32];
A = 69/61,
C = [33/61,36/61,28/61,5/61,2/61,9/61,25/61,7/61,16/61];
A = 33/32,
C = [9/16,15/32,7/32,1/4,7/16,1/8,5/16,1/32,9/32];
A = 69/61,
C = [36/61,33/61,5/61,28/61,25/61,9/61,2/61,7/61,16/61];
A = 33/32,
C = [9/32,5/16,7/16,1/4,1/32,7/32,1/8,9/16,15/32];
A = 69/61,
C = [28/61,16/61,25/61,7/61,9/61,5/61,2/61,36/61,33/61];
A = 69/61,
C = [25/61,16/61,28/61,9/61,7/61,2/61,5/61,36/61,33/61];
A = 33/32,
C = [7/16,5/16,9/32,1/32,1/4,1/8,7/32,9/16,15/32].
```