2

Prolog IV foundations

2.1 Introduction

THE FIRST PRINCIPLE of a language like Prolog IV is to be designed on the basis of an expressive mathematical structure in which we solve complex constraints on unknown elements.

The second principle is to have the freedom to improve this structure by adding new relations, while keeping the ability to solve constraints in the new structure.

Since these principles cannot always be fully respected, the third principle of this language is to be designed in a way which grants the user the ability to grasp clearly the language's shortcomings, and therefore to master them.

Let's now give a more explicit description of these principles.

First principle: An expressive basic structure

The notion of constraint is at the very basis of our language. In order to get us going, here is quite an uncommon constraint, formulated with notations from first order logic and more generally from mathematics:

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

The only variable with free occurrences in this formula is y. Hence this constraint expresses a property about this variable's value. To find this value is Prolog IV's role.

y must by greater than or equal to 5. Furthermore, there must exist three vectors u, v and w and a real number x such that:

- the first element of v is equal to the cosine of the fourth element of v,
- the lengths of u and v are respectively 3 and 10,

- the concatenation of u and v is equal to the concatenation of v and w,
- y is greater than or equal to $2 + 3x$,
- x will be equal to 1 if 74 is greater than $\lfloor 100 \times v_1 \rfloor$ (*i.e.* the integer part of 100 times the first element of v), or to 0 otherwise.

It is not exactly straightforward, but there is a unique y which verifies this constraint, which means that this constraint can be simplified in

$y = 5$

How do we use Prolog IV to solve this constraint? We start the system, which displays a prompt, indicating that it awaits a constraint. We then write the constraint above with the right syntax, Prolog IV solves it, displays the solution, and awaits the next constraint:

```
>> 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.
```

Now, a quick word about the syntax we used. Prolog IV conforms to the ISO Prolog standard[1], and that of course includes its syntax. Any expression must therefore be a « term, » as defined by this standard[2]. Since the terms as defined in this standard are not the same as those used in logic, we will call them ISO-terms. These ISO-terms are essentially of one of the three forms

1 variable,
2 number or identifier,
3 identifier(ISO-term,. . . ,ISO-term),

with infixed variations, specific notations for lists and the convention that the names of variables start by an uppercase letter or an underscore character. These variations allow us to write

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

where the relations `index/3`, `conc/3`, `plus/3`, `times/3` are used with functional notations.

Let's now explain the semantics of the constraint. The logic connector $\wedge$ (here noted by a comma) and the existential quantification $\exists x p$ (here noted x `ex` p) have their usual meaning; we will get back to this later. The variables represent unknown elements, taken from a given domain, while the other symbols represent relations and operations over this same domain.

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

[2] On page 5 of the book *Prolog: The Standard*, by Pierre Deransart and AbdelAli Ed-Dbali, we can read: « In Standard Prolog a unique data structure is used: terms. »

This domain and its associated operations and relations are part of a full structure, chosen once and for all, the basic structure π_4. As we can guess by looking at our example, this structure is quite rich. Our aim is to have a structure in which we can handle simultaneously (1) symbols, (2) numbers, and (3) complex objects built with symbols and numbers. We have been particularly ambitious in the numerical domain, since the structure π_4 includes over 100 relations over **R** (the set of real numbers), **Q** (the set of rational numbers), **Z** (the set of relative integer numbers), and **B** (the set of Boolean values, 0 and 1). We of course take into consideration the inclusions $\mathbf{B} \subset \mathbf{Z} \subset \mathbf{Q} \subset \mathbf{R}$.

Second principle : An enrichable structure

When writing a Prolog IV program, one defines new relations which enrich the basic structure π_4. This enrichment is more quantitative than qualitative: the new relations allow us to use a single constraint to express the conjunction of a great number of atomic constraints which can be directly expressed in π_4. For instance the program

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

defines the link which exists between a list and the sum of its elements, by a conjunction of constraints which express the link between two numbers and their sum. Syntactical conventions allow us to write this program as a block of clauses, more usual in the Prolog world:

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

Third principle : shortcomings are explicit

Prolog IV cannot be perfect; it is not reasonable to expect it to solve exactly constraints in a structure as rich as π_4, and even less reasonable to let the user extend the structure without discipline and still expect that Prolog IV will still be able to solve all the constraints which can be expressed in the new structure.

Our aim has been to make Prolog IV perfect in its imperfection, or in other words we have aimed at making its shortcomings explicit. This is based on a precise logical foundation, which can be outlined in three points.

(1) Syntactically the Prolog IV language may be seen as a first-order language with equality, and constraints as (partially) existentially quantified positive formulae.

(2) Semantically, and without considering any program, the Prolog IV language may be seen as a first order theory with equality $\mathcal{T}(\pi_4)$, *i.e.* a set of axioms which express properties about the π_4 structure. These properties may only be used to solve constraints, but they should be used best. When a programmer gives to Prolog IV a constraint p to solve, he expects the system to return a constraint q equivalent to p in π_4 (*i.e.* which has the same solutions in π_4), whose solutions are more « visible » than they were in p. In particular, he expects Prolog IV to return the constraint *false* when p has no solution in π_4.

In fact, the constraint q returned by Prolog IV will not only be equivalent to p in π_4, but also in all structures ϕ which have the properties $\mathcal{T}(\pi_4)$, *i.e.*

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

Similarly, Prolog IV will return *false* not only if p has no solutions in π_4, but also if it has no solution in any structure ϕ which have the properties $\mathcal{T}(\pi_4)$, *i.e.*

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

(3) Semantically and in presence of a program $\mathcal{P}$, the language Prolog IV may be seen as the theory $\mathcal{T}(\pi_4) \cup \mathcal{P}$. The program $\mathcal{P}$ is an additional set of axioms which we add to π_4. Each such axiom defines the essential property of a relation r and it is of the form

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

where p is a formula in which the only variables with free occurrences are $x_1, \ldots, x_n$. Of course, this is straightforward if p only contains the relations and operations from π_4. However, if p contains other user-defined relations, and in particular the relation r, we then get recursive definitions. The resolution of such a constraint p must then be seen as the computation of an equivalent constraint q in the theory $\mathcal{T}(\pi_4) \cup \mathcal{P}$, *i.e.*the theory such that

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

Structure of the chapter

The present introduction is the first section of the chapter. Section 2 is devoted to the definition of basic logical concepts (constraint, structure, ...). Section 3 describes precisely the π_4 structure. Section 4 is devoted to the axiomatisation of this structure. Section 4 describes the notion of program and the Prolog IV machinery. Finally, section 6 contains various examples of programs written in Prolog IV.

2.2 Syntax and semantics

2.2.1 Semantic objects

The only semantic objects that we will need are relations and operations. A few precise definitions and comments are needed. Let D be a set and n a non-negative integer.

A *relation* ρ is a subset of D^n. We say that ρ is a n-ary relation (or a relation of *arity* n) over D. Since a relation is seen as a set of n-uples, which all have the same length, the empty set $\emptyset$ is a relation which may have any non-negative arity.

An *operation* φ is a function from D^n into D. We say that φ is a n-ary operation (or a function of *arity* n) over D. It is convenient to consider operations as particular relations; we therefore do not distinguish between a n-ary operation φ and the $n+1$-ary relation

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

A n-ary operation therefore is a $n+1$-ary relation ρ which verifies the property:

for each $(a_1, \ldots, a_n) \in D^n$,
there exists a unique $a_0 \in D$ such that $(a_0, a_1, \ldots, a_n) \in \rho$.

A $n+1$-ary relation ρ which verifies the weaker property:

for each $(a_1, \ldots, a_n) \in D^n$,
there exists at most one $a_0 \in D$ such that $(a_0, a_1, \ldots, a_n) \in \rho$.

will be called a *partial operation* or a *pseudo-operation*.

2.2.2 Syntactic objects

The only syntactic objects that we will need are terms and formulae. Terms are used to represent the elements of a domain, and formulae are used to represent the properties of these elements.

We consider once and for all an infinite universal set V of *variable symbols* which will represent items from a domain. We also consider the sets F and R of operation and relation symbols. To each such symbol we associate a non-negative integer, its *arity*.

Terms are expressions of one of the following forms:

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

with $x \in V$, $f \in F$ of arity n and the t_i terms which are shorter than the one that is currently defined.

Formulae are expressions of one of the following 11 forms

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

where the t_i's are terms, r is a n-ary relation symbol from R, and p, q are formulae which are shorter than the one currently being defined. In the right-hand side column, we give for each form, a glimpse on the Prolog IV notation, when it exists.

An occurrence of variable x in a formula q is *bound* if it occurs within a formula of form 7 or 11 in q. An occurrence which is not bound is *free*. A formula which contains no free variable occurrence is a *proposition*.

We also choose the word *constraint* to denote positive existential formulae, *i.e.* formulae which are built only with the 7 first forms. An *atomic* constraint or formula is a formula of form 1, 2, 3 or 4.

2.2.3 Links between syntactic and semantic objects

If we assign a meaning to each symbol which occurs in a term or in a formula, then we will be able to systematically extend this meaning to the whole term or formula. We will now define the extension of meaning. Let's call I a mapping which maps

- each variable $x \in V$ into an element $I(x)$ of a set D,
- each n-ary operation symbol $f \in F$ into a n-ary operation $I(f)$ over the same set D,
- each n-ary relation symbol $f \in F$ into a n-ary relation $I(f)$ over the same set D.

The set D is called the *domain* of I and will be denoted by dom(I), the set F of the operation symbols interpreted by I will be denoted by op(I), and the set R of the relation symbols interpreted by I will be denoted by rel(I).

The interpretation I is a mathematical object which associates a meaning to each symbol in $V \cup F \cup R$. This interpretation can be extended by an application I^* which associates to each term t built over $V \cup F$ an element $I^*(t)$ from dom(I) and to each formula p built over $V \cup F \cup R$ a truth value $I^*(p)$ taken in $\{$true, false$\}$ as follows.

The mapping I^* is defined recursively over the two forms of terms by:

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

and over the 11 forms of formulae by :

$$
\begin{array}{lll}
1 & I^*(\mathit{true}) & = \text{true} \\
2 & I^*(\mathit{false}) & = \text{false} \\
3 & I^*(t_1 \doteq t_2) & = \begin{cases} \text{true} & \text{if } I^*(t_1) = I^*(t_2), \\ \text{false} & \text{otherwise} \end{cases} \\
4 & I^*(r\, t_1 \ldots t_n) & = \begin{cases} \text{true} & \text{if } (I^*(t_1), \ldots, I^*(t_n)) \in I(r) \\ \text{false} & \text{otherwise} \end{cases} \\
5 & I^*(p \wedge q) & = \begin{cases} \text{true} & \text{if } I^*(p) = \text{true and } I^*(q) = \text{true} \\ \text{false} & \text{otherwise} \end{cases} \\
6 & I^*(p \vee q) & = \begin{cases} \text{true} & \text{if } I^*(p) = \text{true or } I^*(q) = \text{true} \\ \text{false} & \text{otherwise} \end{cases} \\
7 & I^*(\exists x p) & = \begin{cases} \text{true} & \text{if there exists } J \in \text{next}(I, x) \text{ with } J^*(p) = \text{true}, \\ \text{false} & \text{otherwise} \end{cases} \\
8 & I^*(\neg p) & = \begin{cases} \text{true} & \text{if } I^*(p) = \text{false} \\ \text{false} & \text{otherwise} \end{cases} \\
9 & I^*(p \Rightarrow q) & = \begin{cases} \text{false} & \text{if } I^*(p) = \text{true and } I^*(q) = \text{false} \\ \text{true} & \text{otherwise} \end{cases} \\
10 & I^*(p \equiv q) & = \begin{cases} \text{true} & \text{if } I^*(p) = I^*(q) \\ \text{false} & \text{otherwise} \end{cases} \\
11 & I^*(\forall x p) & = \begin{cases} \text{true} & \text{if for each } J \in \text{next}(I, x) \text{ we have } J^*(p) = \text{true}, \\ \text{false} & \text{otherwise} \end{cases}
\end{array}
$$

by considering that next(I, x) is the set of interpretations made of the interpretation I, together with all interpretations J which only differ from I by the fact that $I(x) \neq J(x)$.

Now is the time for an important remark: the value $I^\star(p)$ of formula p does not depend on the values $I(x)$ taken by the variables x without free occurrences in p. Therefore, if p is a proposition, *i.e.* a constraint without any free variable occurrence, $I^\star(p)$ does not depend on the way in which variables are interpreted.

If $I^\star(p) =$ true, the we say that p *is true in* the interpretation I, but also that I *satisfies* p. If $\mathcal{T}$ is a set of formulae and p a formula, we will say that p *is true in* $\mathcal{T}$ and we will write

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

if any interpretation which satisfies each formula of $\mathcal{T}$ also satisfies p.

In many reasonings, we will have variations of the variable symbols interpretation, while keeping the same interpretation for the other symbols. It is then interesting to decompose an interpretation I into a couple (ϕ, σ) of mappings, defined as follows:

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

The mapping σ is a *variable assignment*, *i.e.* a mapping from V to a set that we will call dom(V). The mapping ϕ is a *structure*, *i.e.* a family of operations $\phi(f)$ and relations $\phi(r)$ over a domain dom(ϕ), indexed by operation symbols f taken in a set op(ϕ) and by relation symbols r taken in a set rel(ϕ).

When the interpretation I satisfies formula p and I is decomposed into a couple (ϕ, σ), we can then use the mathematical jargon rather than the logical one, and say that the assignment σ is a *solution* of the formula p in the structure ϕ.

2.2.4 Functional representation of relations

In order to represent in a functional way the items in some relations, we see the $(n+1)$-ary relation ρ over D as the following mapping of D^n into the set $\mathcal{P}(D)$ of all subsets of D:

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

We then build expressions which, unlike terms, represent sets of elements of D. We will call these expressions *pseudo-terms*. They have one of the following two forms:

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

with $x \in V$, $f \in F$ of arity n, $r \in R$ of arity n, the s_i's are terms or pseudo-terms shorter than the one currently defined. The form 2 expression needs to contain at least one pseudo-term. In the following table, we give a generalised definition of logical formulae, by adding to the 11 initial forms (see page 46) the two forms

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

where the s_i's are terms or pseudo-terms and where $r \in R$ is n-ary. Atomic formulae are now the formulae of form 1, 2, 3, 3bis, 4 or 4bis, and the notions of bound or free variable occurrences remain unchanged. The right-hand side column gives a glimpse at the Prolog IV notations.

The symbol $\sim$ means *is compatible with*. If a and b are elements or subsets of D, it has the following meaning:

$$a \sim b \overset{def}{\equiv} \begin{cases} a = b, & \text{if } a \text{ and } b \text{ are elements of } D, \\ a \in b & \text{if } a \text{ is an element and } b \text{ a subset of } D, \\ a \ni b & \text{if } a \text{ is a subset and } b \text{ an element of } D, \\ a \cap b \neq \emptyset & \text{if } a \text{ and } b \text{ are subsets of } D. \end{cases}$$

The atomic constraint $r\, s_1 \ldots s_n$ means that there exist some items $a_1, \ldots, a_n$ in the relation r which are compatible with $s_1, \ldots, s_n$.

More precisely, considering an interpretation I, the truth value $I^\star(p)$ of a generalised formula p of form 3bis or 4bis is defined as

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

$$4bis \quad I^*(r\, s_1 \ldots s_n) \quad = \begin{cases} \textit{true} & \text{if there exist some } a_i\text{'s with} \\ & a_1 \sim I^*(s_1), \ldots, a_n \sim I^*(s_n) \\ & \text{et } (a_1, \ldots, a_n) \in I(r) \\ false, & \text{otherwise} \end{cases}$$

where the value $I^\star(s)$ of a pseudo-term is a set defined on both pseudo-term forms by

$$
\begin{array}{llcl}
1 & I^*(r\,s_1 \dots s_n) & = & \left\{ \begin{array}{l} \text{the set of all } b\text{'s such that there exist some } a_i\text{'s with} \\ a_1 \sim I^*(s_1), \dots, a_n \sim I^*(s_n) \text{ and } (b, a_1, \dots, a_n) \in I(r) \end{array} \right. \\
2 & I^*(f\,s_1 \dots s_n) & = & \left\{ \begin{array}{l} \text{the set of all } b\text{'s such that there exist some } a_i\text{'s with} \\ a_1 \sim I^*(s_1), \dots, a_n \sim I^*(s_n) \text{ and } b = I(f)(a_1, \dots, a_n). \end{array} \right.
\end{array}
$$

All the other notions linked to interpretations remain unchanged.

A generalised formula can always be transformed in an equivalent formula in which no pseudo-term or $\sim$ symbol appear. This may be done automatically by repeating the following transformations:

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

where t_1 and t_2 are terms, $p(r\,s_1 \dots s_n)$ is an atomic formula in which we have outlined the occurrence of a form 1 pseudo-term $r\,s_1 \dots s_n$, and where $p(x)$ is the same atomic formula in which we have substituted a new variable x (*i.e.* a variable without occurrence in the formula) to the outlined occurrence of the pseudo-term.

Transformation 2 is used to eliminate all the relation symbols embedded in atomic constraints, hence to eliminate all the pseudo-terms. Transformation 1 is then used to eliminate all the $\sim$ symbols.

If we apply these transformations to the constraint

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

from table 2.3, page 53 we obtain (by taking into account the fact that `cc(1,2) u co(4,8)` is the infixed variant of `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)
```

Let's finish this section with a remark about syntax. As we just saw, the the introduction of the $\sim$ symbol is not absolutely necessary, since we could have used the equality symbol by extending its meaning to pseudo-terms. However, it would have been shocking to link by an equality symbol some sets which may be different. Nevertheless, in a constraint $x \sim s$, where s is a pseudo-term which represents a set of 1 or 0 element, the equality sign is better aesthetically. For all these reasons, the Prolog IV compiler allows the user to replace any occurrence of the $\sim$ symbol by an equality symbol. Actually, this is exactly what we have done in our very first constraint example, page 41.

2.3 The basic structure π_4

2.3.1 Domain of the structure π_4

The domain $\mathrm{dom}(\pi_4)$ of the structure π_4 is the set of all trees whose nodes (which may be in infinite numbers) are labeled

1. either by ISO-Prolog identifiers (completed by an arity),
2. either by real numbers (whose arity is considered to be zero).

We do not distinguish between trees reduced to a single node and such a node's label. Identifiers and real numbers are assimilated to single-node trees labeled by identifiers and reals numbers. Several subsets of our domain have a distinctive role, let's now examine them.

Set of rational numbers These are the real numbers of the form $\pm\,\frac{p}{q}$, where q is a positive integer, and p a positive or null integer. We will write them simply as `p/q`, possibly prefixed by a sign.

Set of decimal rational numbers These are rational numbers of the form $\pm\,p \times 10^{\pm q}$, where p and q are positive or null integers. They are written in the usual way, with the possible occurrence of a decimal point and an exponential part introduced by the letter `E`. One may note that integer numbers are a particular case of decimal numbers.

Set of IEEE rational numbers These are binary floating-point numbers, as defined by the IEEE standard for single-precision numbers[3]. They are of the form

$\pm\,p \times 2^q$ with $0 \le p \le 2^{24}–1$ et $-149 \le q \le 104$,

where p and q are integers. There are $2n+1$ of them, with $n = 2^{31} - 2^{23} - 1$, which makes a total of $2\,139\,095\,039$. If we number them from $-n$ to n and call q_i the number with rank i, then the integer $|i|$ written in binary over 31 bits anc prefixed by a bit which shows the sign of i is the IEEE encoding of q_i. This encoding is without redundancy, except for 0, which is encoded twice (one with a plus sign, and once with a minus sign).

The set of IEEE rationals is a subset of the set of decimal rationals[4] and therefore they can be noted in decimal form with many numbers. In order to have a compact notation, we write '$<x$' and '$>x$' to designate the IEEE rationals which respectively precede and follow immediately the unsigned decimal number x.

[3] 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

[4] If q is non-negative, then they are integers and therefore decimal rationals, and if q is negative, they are of the form $\pm\,(5^{|q|} \times p) \times 10^q$.

Interval An interval is a set of real numbers of one of the following ten forms:

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

where a and b, the lower and upper bounds, are real numbers and the last column shows the Prolog IV notations for these intervals (as pseudo-terms).

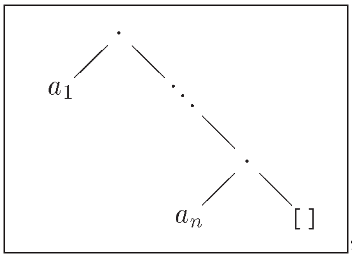
IEEE interval This is an interval whose lower and upper bounds a and b, when they exist, are IEEE rational numbers. One can note that the set of IEEE intervals has as elements the empty set and the set of all reals, and that this set is closed under finite and infinite intersection. This very last point comes from the fact that the set of IEEE rational numbers is finite (and the set of IEEE intervals as well).

Union of IEEE intervals It is a set of reals of the form $E_1 \cup \cdots \cup E_n$, where each E_i is an IEEE interval.

Set of rational trees This is the set of all trees which have a finite number of subtrees. We recall that there exist trees which have an infinite number of nodes and a finite number of (non isomorphic) subtrees.

Set of twice rational trees This is the set of all rational trees whose numerical nodes are all labeled with rational numbers.

Set of lists Lists are the trees a of the form



,

where the a_i's are trees. Such a list a is noted $[a_1, \ldots, a_n]$, amd we denote by $|a|$ its (possibly null) *size* n. The concatenation of two possibly empty lists is defined and denoted by $[a_1, \ldots, a_m] \circ [a_{m+1}, \ldots, a_n] = [a_1, \ldots, a_n]$.

2.3.2 Privileged subdomains

The whole Prolog IV approximated resolution mechanism is based on the fact that we attribute to each variable a subset of the set dom(π_4) and that we constantly try

1. to « reduce » these subsets by taking into account their interactions with each constraint considered separately, and

2. to « globalise » each local constraint, *i.e.* to replace a complex constraint by a more basic one that we can solve globally.

The subsets that we use in this process are not just any sets. They are the *privileged subdomains*. They are obtained by multiple intersection of the 15 basic forms of table 2.1, which represent subsets which are not always disjoint.

Table 2.1: Privileged subdomains

	Basic forms
1	the set of all trees
2	theset of all finite trees
3	the set of all infinite trees
4	the set of all trees which are lists
5	the set of all trees which are not lists
6	for each integer n positive or null, the set of all trees which are lists of size n
7	for each n-uple $A_1, \ldots, A_n$ of IEEE intervals, the set of all lists of the form $[a_1, \ldots, a_n]$, with $a_i \in A_i$
8	the set of all single-node trees
9	the set of all trees with more than one node
10	the set of identifiers
11	the set of all trees which are not identifiers
12	for each IEEE interval A, the set A
13	the set of all trees which are not a simple rational number
14	for each twice rational tree a, the set $\{a\}$
15	the empty set

A possible variant for forms 7 and 12 is given in table 2.2. The modalities for using this variant are described in the « Environment » chapter.

Table 2.2: Variant of the privileged subdomains

	Basic forms
7	for each n-uple $A_1, \ldots, A_n$ of unions A_i of IEEE intervals, the set of all lists of the form $[a_1, \ldots, a_n]$
12	for each union A of IEEE intervals, the set A

Each privileged subdomain is of category (a), of category (b) or of both categories. The category (a) subdomains are used in the subdomain reduction process, and the category (b) subdomains are used in the constraint globalisation process.

- In category (a), we have all the privileged subdomains, excepted those of form 14, when a is a number which is not a IEEE rational.

- In category (b), we have all the privileged subdomains, excepted those of forms 7 or 12, when one of the A_i or A is a set which is neither the empty set nor the set **R** of all reals.

We can note that the set of category (a) privileged subdomains is closed by finite or infinite intersection, and that it contains the set of all trees. Therefore, for any subset A of dom(π_4), there exists a smallest (for inclusion) category (a) privileged subdomain which contains A.

Furthermore, this set is equal to the intersection of all category (a) privileged subdomains which contain A (among which there always is the set dom(π_4) itself).

In table 2.3, you can find examples of the Prolog IV notation which expresses that X belongs to one of the 15 possible forms of privileged subdomains, with both variants. All these examples are presented in the form of a mltiple disjunction which makes a query.

Table 2.3: Examples of privileged subdomains

```
>> 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 variant
   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 variant
   X ~ nreal;                                   % 13
   Y ex X = trio(X,Y,6), Y = duo(X,Y);          % 14
   X ~ ntree.                                   % 15
```

2.3.3 Operations in the structure π_4

The set op(π_4) of operation symbols f is made of

- any identifier with its arity id/n, which is not a reserved identified from table 2.4,
- any constant representing a rational number.

Table 2.4: Identifiers reserved for the relation symbols of 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	

The operation $\pi_4(f)$ that the structure π_4 associates to the n-ary symbol f consists in taking a sequence $a_1, \ldots, a_n$ of trees and then in building a tree whose root is labeled (approximately) by f, and whose immediate children are $a_1, \ldots, a_n$. More precisely, the operation $\pi_4(f)$ is the mapping

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

where $\ddot{f}$ is either an identifier equal[5] to f, or the rational number represented by constant f. Of course, the a_i's are elements of dom(π_4), and n cannot be zero.

[5]If the identifier f starts with the character ASCII ^ (hat), then this character is removed from $\ddot{f}$. This allows us to create trees which are labeled by reserved identifiers.

2.3.4 Relations of the structure π_4

The set $\mathrm{rel}(\pi_4)$ of relation symbols r is made of

- any identifier with its arity id/n, from table 2.4, page 54,
- the symbol $\mathrm{dif}'/2$,
- any symbol of the form $\mathrm{in}A/1$, where A is a privileged subdomain.

The symbol $\mathrm{dif}'/2$ and the symbols $\mathrm{in}A/1$ are not made available to the programmer. They have been introduced in order the state the properties used by the constraint solving algorithms.

Tables 2.5, 2.6 et 2.7 define the mapping of these symbols into relations

$r \mapsto \pi_4(r)$

for each symbol r of $\mathrm{rel}(\pi_4)$. Beware, the relations whose names are prefixed with a dot in table 2.4 are not implemented in the current version of Prolog IV.

Table 2.5: Relations of the basic structure π_4, first part

Lists and trees, relations

1 tree/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ can be any tree$\}$
2 ntree/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ cannot exist$\}$
3 finite/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ is finite$\}$
4 infinite/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ is infinite$\}$
5 list/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ is a list$\}$
6 nlist/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ is not a list$\}$
7 leaf/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ is a single-node tree$\}$
8 nleaf/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ is not a single-node tree$\}$
9 real/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ is a real number$\}$
10 nreal/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ is not a real number$\}$
11 identifier/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ is an identifier$\}$
12 nidentifier/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ is not an identifier$\}$
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	$\mapsto$	$\{(a, b) \in \mathbf{A}{\times}\mathbf{L} \mid a$ is an element of $b\}$
16 outlist/2	$\mapsto$	$\{(a, b) \in \mathbf{A}{\times}\mathbf{L} \mid a$ is not an element of $b\}$

Lists and trees, pseudo-operations yielding Booleans

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

Lists and trees, usual pseudo-operations

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	$\mapsto$	$\{(a, b, c) \in \mathbf{A}{\times}\mathbf{L}{\times}\mathbf{Z} \mid 1 \leq c \leq \lvert b\rvert$ and a is the $c^{\text{i-th}}$ element of $b\}$
34 u/3	$\mapsto$	$\{(a, b, c) \in \mathbf{A}^3 \mid a = b$ or $a = c\}$
35 n/3	$\mapsto$	$\{(a, b, c) \in \mathbf{A}^3 \mid a = b$ and $a = c\}$
36 if/4	$\mapsto$	$\{(a, b, c, d) \in \mathbf{A}{\times}\mathbf{B}{\times}\mathbf{A}^2 \mid$ if $b = 1$ then $a = b$ else $a = c\}$

Booleans, 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	$\mapsto$	$\{(a, b) \in \mathbf{B}^2 \mid a = 1$ or $b = 1\}$
40 and/2	$\mapsto$	$\{(a, b) \in \mathbf{B}^2 \mid a = 1$ and $b = 1\}$
41 impl/2	$\mapsto$	$\{(a, b) \in \mathbf{B}^2 \mid$ if $a = 1$ then $b = 1\}$
42 equiv/2	$\mapsto$	$\{(a, b) \in \mathbf{B}^2 \mid a = b\}$

$\mathbf{A}$: set of all trees, $\mathbf{B}$: set $\{0, 1\}$,
$\mathbf{L}$: set of lists, $\mathbf{Z}$: set of integers, $\mathbf{R}$: set of reals.

Table 2.6: Relations of the basic structure π_4, second part

Booleans, pseudo-operations

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

Reals, relations

49 int/1 $\mapsto$ $\{a \in \mathbf{R} \mid a \text{ is an integer}\}$
50 nint/1 $\mapsto$ $\{a \in \mathbf{R} \mid a \text{ is not an integer}\}$
51 prime/1 $\mapsto$ $\{a \in \mathbf{R} \mid a \text{ is a prime integer}\}$
52 nprime/1 $\mapsto$ $\{a \in \mathbf{R} \mid a \text{ is a non-prime integer}\}$
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 $\mapsto$ $\{(a,b,c) \in \mathbf{R}^3 \mid a \notin (b,c)\}$

Reals, pseudo-operations yielding Booleans

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

A : set of all trees, **B** : set $\{0,1\}$,
L : set of lists, **Z** : set of integers, **R** : set of reals.

Table 2.7: Relations of the basic structure π_4, third part

Reals and integers, usual pseudo-operations

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	$\mapsto$	$\{(a,b,c) \in \mathbf{Z}^3 \mid a \geq 0, b \geq 0, c \geq 0$ and $a = \mathrm{gcd}(b,c)\}$
88 lcm/3	$\mapsto$	$\{(a,b,c) \in \mathbf{Z}^3 \mid a \geq 0, b \geq 0, c \geq 0$ and $a = \mathrm{lcm}(b,c)\}$
89 intdiv/3	$\mapsto$	$\{(a,b,c) \in \mathbf{Z}^3 \mid$ there exists $d \in \mathbf{Z}$ such that $b = ac + d$ and $c > d \geq 0\}$

Reals, usual pseudo-operations

90 uplus/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a = +b\}$
91 uplusIin/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	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid c \neq 0$ and $a = b/c\}$
102 divlin/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid c \neq 0$ and $a = b/c\}$
103 mod/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid 0 \leq a < c$ and $a = b \bmod c\}$
104 min/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a = \min(b,c)\}$
105 max/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^3 \mid a = \max(b,c)\}$

Reals, pseudo-operations corresponding to special functions

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	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid b \geq 0$ and $a = \sqrt{b}\}$
109 power/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^2\times\mathbf{Z} \mid c \geq 0$ and $a = b^c\}$
110 root/3	$\mapsto$	$\{(a,b,c) \in \mathbf{R}^2\times\mathbf{Z} \mid c \geq 0$ and $a = \sqrt[c]{b}\}$
111 arccos/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid 0 \leq a \leq \pi$ and $b = \cos a\}$
112 arcsin/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid -\pi/2 \leq a \leq \pi/2$ and $b = \sin a\}$
113 arctan/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid -\pi/2 < a < \pi/2$ and $b = \tan a\}$
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	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid b \bmod 2\pi \neq 0$ and $a = \cot b\}$
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	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid b > 0$ and $a = \ln b\}$
120 log/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid b > 0$ and $a = \log b\}$
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	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid b \bmod 2\pi \neq \pi/2$ and $a = \tan b\}$
124 tanh/2	$\mapsto$	$\{(a,b) \in \mathbf{R}^2 \mid a = \tanh b\}$

Internal relations, for each privileged subdomain A

125 dif$'$/2	$\mapsto$	$\{(a,b) \in \mathbf{A}^2 \mid a \neq b\}$
126 in$A$/1	$\mapsto$	$\{a \in \mathbf{A} \mid a$ is an element of the privileged subdomain $A\}$

A : set of all trees, **B** : set $\{0,1\}$,
L : set of lists, **Z** : set of integers, **R** : set of reals.

2.4 Axiomatisation of π_4

2.4.1 Generalities

Let's consider a constraint p in which $x_1, \ldots, x_n$ are the only variables with free occurrences. Solving p consists in finding all the possibles combinations of values for these variables which verify the condition expressed by p in the structure π_4, *i.e.* in finding all the variable assignments σ such that the combination of σ with π_4 make an interpretation I such that

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

Since this set of solutions is usually infinite, we will rather try to simplify p into an equivalent constraint q which makes the set of solutions « more visible. » In particular, if p has no solution in π_4, we would expect q to be equal to the logical constant *false*.

Because of the richness of the structure π_4, it is hopeless to reach this last objective. We therefore need to yield to the use of an incomplete solver. In this context, how can we, without getting into the details of the solver's implementation, let the user know about the solver's capacities and shortcomings? This is where we use π_4's axiomatisation. The idea is to define the exact set $\mathcal{T}(\pi_4)$ of π_4 properties which are used in the resolution algorithm and to guarantee that these properties will be used as much as possible. Each property p will be a first-order proposition such that

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

and the set $\mathcal{T}(\pi_4)$ of all these p's will be a first-order theory building a partial axiomatisation of π_4, *i.e.* there will exist propositions p verifying at the same time

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

When the programmer will give Prolog IV a constraint p to solve, the system will return a constraint q which will be equivalent to p not only in the structure π_4, but also in any other structure which has the same property has π_4, *i.e.* such that:

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

This constraint q will only be reduced to the logical constraint *false* if it has no solution in the π_4 structure and if it has no solution either in any other structure with the properties in $\mathcal{T}(\pi_4)$, *i.e.* if:

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

As shown in table2.8, the theory $\mathcal{T}(\pi_4)$ is made of 25 possible axiom schemas, grouped in 8 categories. This is a first attempt of axiomatisation, which may still contain some errors.

Table 2.8: Selected properties of the structure π_4

1	Empty subdomain	Privileged subdomains
2	Universal subdomain	,, ,,
3	Subdomain intersection	,, ,,
4	Distinct operations, distinct results	Operations over trees
5	At least one solution	,, ,,
6	Equality propagation	,, ,,
7	One-element subdomains	Interactions between subdomains and operations
8	Subdomain reductions	,, ,,
9	Infinite trees	,, ,,
10	Finite list size	,, ,,
11	Subdomain reduction	General relations
12	Constraint building	,, ,,
13	Product by a positive scalar	Linear constraints
14	Sum of constraints	,, ,,
15	Enforcing order	,, ,,
16	Hidden constant	,, ,,
17	Equality linearisation	,, ,,
18	Numerical typing	Linearised relations
19	Constraint linearisation	,, ,,
20	Equality detection	Relations dif, bdif and beq
21	Equality, bdif et beq	,, ,,
22	Inequality detection	,, ,,
23	Inequality, bdif et beq	,, ,,
24	dif propagation	Additional equality propagation
25	dif$'$ propagation	,, ,,

2.4.2 Axiomatisation of the privileged subdomains

The first properties concern the belonging to a privileged subdomain:

1 $\forall x$

$$\text{in}\emptyset\, x \;\equiv\; \mathit{false}$$

2 $\forall x$

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

3 $\forall x$

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

where **A** is the set of all trees (*i.e.* the whole domain), and A, B are any privileged subdomains.

Property 1 expresses the fact that the empty set contains no element. Property 2 expresses the fact that any item is a tree. Property 3 expresses the fact that if an item belongs to any two privileged subdomains, it also belongs to their intersection (which is a privileged subdomain, as we have seen before). Here are a few queries which illustrate all this.

```
>> X ~ ntree.
false.

>> X = X.
X ~ tree.

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

2.4.3 Axiomatisation of the tree operations

We have three properties:

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

where f and g are two distinct operation symbols (of respective arities m and n), where the $t_i(x_1, .., x_n, u_1, .., u_m)$ are terms which are built with one occurrence of an operation symbol and variables taken in $\{x_1, .., x_n, u_1, .., u_m\}$, and where the $t_i(y_1, .., y_n, v_1, .., v_m)$ are the same terms, in which the x_j's have been replaced by the y_j's and the u_j's by the v_j's.

Property 4 indicates that two trees which are built with two operations named by different symbols are distinct. The following queries illustrate this phenomenon:

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

>> a=b.
false.

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

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

We can note in the last query that the two occurrences of `f` are considered as referencing to two different symbols `f/1` and `f/2`. The arity of a symbol is encoded in the comma and the parentheses.

Property 5 indicates that a particular type of conjunction admits at least one solution:

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

Property 6 allows us to decompose an equation into several equations:

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

It also indicates that the conjunction of equations shown in axiom 5 has at most one solution, which lets us perform important simplifications:

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

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

Let's consider ϕ, a structure obtained by removing all relation symbols from π_4 (keeping only operation symbols). There exist other structures than ϕ in which all the properties above are true. However, Michael Maher[6] has shown that there exists a set of properties (in fact equivalent to the set $\mathcal{T}$ of properties 4, 5 and 6, which is a complete axiomatisation of ϕ. This means that if p is proposition then, for any structure ϕ' in which the propositions $\mathcal{T}$ are true, we have

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

2.4.4 Axiomatisations of the interactions between subdomains and operations

The four properties below concern the interactions between the subdomain constraints and the operations:

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

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

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

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

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

where the $t_i(x_1, \ldots, x_n)$ are terms built with an operation symbol and some variables taken in $\{x_1, \ldots, x_n\}$, where the a_i's are twice rational trees which verify equivalence 7, where the A_i's and A''s are category (a) privileged subdomains which verify equivalence 8, where the $s_i(x_j)$'s are terms built with an operation symbol and at least one occurrence of x_j, where **I** is the set of all infinite trees, and where **K** is the set of all trees which are not lists. We can recall that lists are built with the symbol « dot. »

Property 7 shows the equivalence between two properties: « to be constrained by operations and equalities to have a single value », and « to belong to a one-element privileged subdomain. »

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

Property 8 is used to reduce the subdomains of the variables involved in an operation, and among other things to get to the empty subdomain, which according to property 1 (page 60) will produce *false.*

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

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

Property 9 corresponds to the famous « occurs-check » which is used to detect infinite trees.

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

Finally, property 10 forbids a list from having an infinite number of elements.

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

2.4.5 Axiomatisation of general relations

The two following properties concern all the relation symbols r, except those which are postfixed by « lin » and those which are not made accessible to the programmer.

11 $\forall x_1 \ldots \forall x_n$

$$\begin{pmatrix} & r\,x_1 \ldots x_n \\ \wedge & \mathrm{in}A_1\,x_1 \\ & \ldots \\ \wedge & \mathrm{in}A_n\,x_n \end{pmatrix} \equiv \begin{pmatrix} & r\,x_1 \ldots x_n \\ \wedge & \mathrm{in}A'_1\,x_1 \\ & \ldots \\ \wedge & \mathrm{in}A'_n\,x_n \end{pmatrix}$$

12 $\forall x_1 \ldots \forall x_n$

$$\begin{pmatrix} & r\,x_1 \ldots x_n \\ \wedge & \mathrm{in}B_1\,x_1 \\ & \ldots \\ \wedge & \mathrm{in}B_n\,x_n \end{pmatrix} \equiv c(x_1, \ldots, x_n)$$

Here, the A_i's and A'_i are category (a) privileged subdomains such that equivalence 11 is verified. The formula $c(x_1, \ldots, x_n)$ is a *basic formula* of one of the 7 following forms

1	*true*,	logical constant,
2	*false*,	logical constant,
3	$p \wedge q$,	conjunction,
4	$\exists x p$,	quantification,
5	$x \in C$,	privileged subdomain,
6	$x = y$,	equality,
7	$x_0 = f\,x_1 \ldots x_n$,	construction,

where p and q are basic formulae themselves, and C is any privileged subdomain. The B_i's are category (b) privileged subdomains such that the equivalence 12 is respected in π_4.

Property 11 is used to reduce the subdomains of variables which are involved in a general relation and, among other things, to obtain the empty subdomain (which according to axiom 1 (page 60, will produce *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.
```

Property 12 indicates that some particular constraints, which are equivalent in π_4 to a basic formula, must perform as such a formula. In numerical parts, this often triggers some infinite precision computations.

```
>> 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 In axioms 11 and 12, if r is one of the symbols binlist/3, boutlist/3, conc/3, index/3, inlist/2, outlist/2, then the A_i's, A'_i's and B_i's must be privileged subdomains which do not involve the set of finite trees or the set of infinite trees.

2.4.6 Axiomatisation of linear constraints

In order to express the axioms for linear constraints, we will introduce the notation

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

which is often noted as $a_0 + \sum a_ix_i \geq 0$. This denotes the linear inequation written in functional form, where the product is made using the relation « timeslin, » the sum using « pluslin, » and the comparison $\geq$ using « geqlin. » Of course, the a_i's are relative numbers, and the x_i's are distinct variables. We also consider that the constraint 2.5 also represents any constraint which is obtained by a permutation of the monomials a_ix_i or by the addition of terms in which the a_i would be 0. We then have the following 5 properties:

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

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

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

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

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

where k is a strictly positive integer, a_0 is strictly positive in axiom 15, and **R** is the set of real numbers.

Properties 13 and 14 indicate that, if we only have positive coefficients, the conjunction of two constraints implies all of their linear combinations. Property 15 indicates that an integer cannot be at the same time positive and negative.

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

Property 16 allows us to detect the variables which can only take one value:

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

Property 17 allows us to encode a numerical equality as a combination of linear inequalities.

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

2.4.7 Axiomatisation of the relations to be linearised

A *general linear constraint* is a constraint of one of the following 6 forms:

1	*true*,	logical constant,
2	*false*,	logical constant,
3	$p \wedge q$,	conjunction,
4	$\exists x p$,	quantification,
5	dif $x y$,	non-equality,
6	$a_0 + a_1 x_1 + \cdots + a_n x_n \geq 0$,	basic linear constraint.

where p and q are general linear constraint themselves.

Here are two properties about constraints which are built with relation symbols r which are postfixed by « lin. »

18 $\forall x_1 \ldots \forall x_n$

$$r\, x_1 \ldots x_n \;\Rightarrow\; \begin{pmatrix} & \mathrm{in}\mathbf{R}\, x_1 \\ & \ldots \\ \wedge & \mathrm{in}\mathbf{R}\, x_n \end{pmatrix}$$

19 $\forall x_1 \ldots \forall x_n$

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

Here, **R** is the set of real numbers, $c(x_1, \ldots, x_n)$ is a general linear constraint, and the B_i's are category (b) privileged subdomains which verify equivalence 19 in π_4.

Property 18 indicate that the arguments of a « lin » atomic constraint always are real numbers.

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

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

Property 19 indicates that some constraint forms, which are equivalent in π_4 to general linear formulae, must behave like these linear formulae.

```
>> 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 of the relations dif, bdif et beq

The handling of the constraints built with the relation symbols dif, bdif and beq is very complete. It verifies 6 axioms, including the 4 following ones:

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

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

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

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

where A_1 and A_2 are disjoint category (a) privileged subdomains, where f and g are distinct operation symbols, and where a_0 is a non-zero integer.

Property 20 is used to detect some equalities which are then taken into account in property 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.
```

Property 22 is used to detect some diequations which are taken into account in property 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 of the additional equality propagation

By negation of the relations dif and dif$'$, we can obtain some equalities. These equalities must respect a property which is equivalent to axiom 6 (page 61):

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

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

Here are some examples of equality propagations through axiom 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.
```

and some more, using axiom 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 Enriched structures

2.5.1 General notion of program

Let's consider a structure ϕ, including a family of relations $\phi(r)$, indexed by by symbols from rel(ϕ). We are interested in defining new relations, indexed by *predicate* symbols, taken from a set *pred*, which must be disjoint from rel(ϕ). A *predicate symbol definition* will be a formula of the form

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

where p is a formula built on the set of symbols $V \cup \mathrm{op}(\phi) \cup \mathrm{rel}(\phi) \cup \mathit{pred}$ in which the only variables with free occurrences are $x_1, \ldots, x_n$. A set of distinct predicate definition is a *program*.

In Prolog IV, where the structure ϕ is of course π_4, and the formulae p will all be constraints (*i.e.* positive existential constraints). The set *pred* will be made of couples id$/n$, where id is an ISO identifier, non-reserved with arity n (see figure 2.4, page 54).

2.5.2 Program in clausal form

Using the classical syntax of a Prolog program, it seems that the right side of a definition of the form 2.6 implies the left side rather than it is equivalent to it. The definition is then written as one or more implications. In every one of these implications, the usage is not to write the exterior universal quantification, and to use a clausal syntax.

A clause is an expression of the form (on the right, you have the Prolog IV form):

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

where r is a n-ary predicate, the s_i's are terms or pseudo-terms, and p is a constraint built the symbols in $V \cup \mathrm{op}(\pi_4) \cup \mathrm{rel}(\pi_4) \cup \mathit{pred}$. If p is the constraint *true*, we can replace the expression 2.7 by $r\,s_1 \ldots s_n$.

A *normal form* of the clause 2.7 is a clause of the form

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

where $x_1, \ldots, x_n$ are distinct variables, which occur neither in the s_i's nor in p, and where the y_i's form a superset of the variables which have free occurrences in p. When the s_i's are terms, the symbols $\sim$ may of course be replaced by equality symbols.

A clause *block* defining a predicate r is a non-empty sequence of clauses, which all have the same head predicate r. The definition of the predicate r represented by this block is:

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

where $(r\,x_1 \ldots x_n \;\; :\!- \;\; p_1)$, ..., $(r\,x_1 \ldots x_n \;\; :\!- \;\; p_m)$ is the sequence of clauses from the block, put in normal form, so that they all have the same left side $r\,x_1 \ldots x_n$.

A *program in clausal form* is a sequence of clause blocks which define distinct predicates. The program represented by this sequence of blocks is made of the set of predicate symbol definitions $r \in R$ which are

1. either represented by a clause block which defines r,
2. or of the form $\forall x_1 \ldots \forall x_n\ (r\, x_1 \ldots x_n\ \equiv\ \mathit{false})$, when r is a predicate symbol which is not defined by any clause block.

For instance, in order to define the sum of the elements of a 3-element list, we will introduce the definition of the binary predicate sigma (here written with a few additional parentheses and commas, and the pseudo-term $a{+}b{+}c$),

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

which will be written in Prolog IV clausal form

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

After compiling this program, we may ask the query:

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

If we want to define a more general relation which relates a numerical list with the sum of its elements we will write:

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

To compute the sum of 2,3 and 4 we will then solve the constraint

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

This is equivalent to the following Prolog IV program

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

which may also be written

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

and to start the query

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

2.5.3 The Prolog IV machine

The definition 2.8 of the relation named sigma includes some references to this function itself, and this deserves a few comments.

Our aim is to compute the solutions of a constraint p in the structure π_4, enriched with a set of relations defined in a program $\mathcal{P}$. This enriched structure is not necessarily unique, so we cannot, so to say, solve p in *the* enriched structure. We need to assume that there is a constraint q such that

1. q only contains symbols from π_4,
2. $\mathcal{P} \cup \mathcal{T}(\pi_4)\ \models\ p \equiv q$.

Then we need to solve q in the structure π_4 (in which case we are back to the problem evoked on page 59. We will therefore try to compute a constraint q which satisfies points 1 and 2, and whose solutions are the most « visible » possible. As mentioned, we will not be able to guarantee that q is *false* every time that q has no solution in π_4. We will ensure that q is the constraint *false* if q does not have any solutions in any structure ϕ which satisfies the axioms $\mathcal{T}(\pi_4)$, *i.e.* if $\mathcal{T}(\pi_4) \models q \equiv \mathit{false}$.

The way in which q is computed from p is the essence of the execution of a Prolog IV program $\mathcal{P}$ from the query p. We will now describe this sequence of events using an abstract machine (the *Prolog IV machine*), formalised by a set of rewriting rules which transform subformulae into subformulae, going from p to q.

Let's call *existential conjunction* a constraint of the form

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

where the a_i's are atomic constraints, and let's call *constraint solved in* $\mathcal{T}(\pi_4)$ a constraint p which only uses the symbols from π_4 and which is necessarily equal to *false* if $\mathcal{T}(\pi_4) \models p \equiv \mathit{false}$.

Prolog IV machine We consider a constraint p and a program $\mathcal{P}$, and we want to compute a constraint q such that:

1. $\mathcal{P} \cup \mathcal{T}(\pi_4) \models p \equiv q$,
2. q is of the form $m_1 \vee \ldots \vee m_n \vee \mathit{false}$ with n possibly equal to 0,
3. each m_i is an existential conjunction, solved in $\mathcal{T}(\pi_4)$.

We start from the constraint

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

the quotes are simply used to show the occurrences of formulae which are more important than others. We then apply as many times as possible the following rewriting rules on the leftmost expression of the form $\langle m \rangle$. If the process stops, we get a formula of the form

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

where each m_i is an existential conjunction. The constraint $m_1 \vee \cdots \vee m_n \vee$ *false* is then the sought-for constraint q. If the process does not stop, it means that the programmer has either entered a wrong query or written a wrong program!

Rewriting rules

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	$\langle m\rangle \wedge r\, s_1 \ldots s_n$	$\Longrightarrow$	$\langle m\rangle \wedge \exists x_1 \ldots \exists x_n \left(\begin{pmatrix} x_1 \sim s_1 \\ \wedge \ldots \wedge \\ x_n \sim s_n \end{pmatrix} \wedge \text{body}(r\, x_1 \ldots x_n) \right)$

with the following conventions

- $p, p(x), q$ are any constraints, and $p(x')$ is the constraint $p(x)$ in which every free occurrence of variable x has been replaced by variable x'.
- x, x' are variables such that in rule 10, neither m nor $p(x)$ have any free occurrence of x'.
- a is an atomic constraint which includes noo symbol which are not in π_4.
- m is an existential conjunction, solved in $\mathcal{T}(\pi_4)$, which is not equal to *false*.
- quasisolved(r) is an existential conjunction solved in $\mathcal{T}(\pi_4)$, such that $\mathcal{T}(\pi_4) \quad \models \quad r \equiv \text{quasisolved}(r),$
- r is a predicate symbol, the s_i's are terms or pseudo-terms and the x_i's are variables which do not appear in the s_i's,
- body$(r\, x_1, \ldots, x_n)$ is a constraint such that, except for variable names, the following proposition belongs to $\mathcal{P}$, $\forall x_1 \ldots \forall x_n \;(r\, x_1, \ldots, x_n \equiv \text{body}(r\, x_1, \ldots, x_n)).$

Rules 7 and 8 distribute the $\wedge$ connector over the $\vee$ connector in order to obtain a disjunction of conjunctions. This is the simplest way to solve a Boolean constraint, and the implementation of this distribution is inside the « backtracking » machinery which is found in any Prolog implementation. For instance, the resolution of the constraint $(x=1 \vee x=2 \vee x=3) \wedge (y=1 \vee y=2 \vee y=3)$ produces:

```
>> (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.
```

We can recall that the variables always start by uppercase letters, that the comma replaces the connector $\wedge$, the semicolon replaces the connector $\vee$, and the period marks the end of a constraint. This transformation of a disjunctive form is highly combinatorial, since the size of the final formula obtained may grow exponentially with the size of the initial formula. Of course, any such combinatorial explosion is the sole responsability of the programmer!

Rules 6 and 9 favor the left-to-right associations for conjunctions and disjunctions. Rules 5 and 10 move the quantifiers to the needed level by increasing their span in conjunctions and by distributing them in disjunctions. These rewriting rules have a minor effect which may be simulated by using a standard encoding for conjunctions, *i.e.* by renaming some variables and by keeping track of those which are not quantified.

Rules 1, 2, 3, 4 and 11 simplify the formula by using a « solver » from the theory $\mathcal{T}(\pi_4)$. For instance, for the constraint $x > y \wedge (x = 1 \vee x = 2 \vee x = 3) \wedge (y{=}1 \vee y{=}2 \vee y{=}3)$ we get:

```
>> 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.
```

and for the constraint $\exists x \exists y\ (z{=}[x, y] \wedge x \geq y \wedge (x{=}1 \vee x{=}2 \vee x{=}3) \wedge (y{=}1 \vee y{=}2 \vee y{=}3))$ we get

```
>> X ex Y ex Z = [X,Y], ge(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].
```

Finally, rule 12 (approximately) replaces the left side of a predicate definition by its right side, in a way similar to a macro-expansion.

2.6 Examples of Prolog IV programs

To conclude this chapter, here are a few examples of Prolog IV programs which, we hope, will give you an idea of the language's main features: list handling, computations with integers and Booleans, general constraint solving over real numbers, and linear constraint solving.

2.6.1 List constraints

Among the Prolog IV constraints which handle lists, two are particularly useful: the concatenation, and the selection of the i-th element of a list. The four first examples illustrate this point.

Bidirectional translational between arabic and roman numbers

This first example shows the interest in having a list concatenation constraint, even with axioms as poor as axioms 12 and 13 (page 64). The goal here is to translate a number written in decimal (*i.e.* arabic) notation into roman notation, and vice-versa. The arabic numeral is described by a list of digits, and the roman numeral by a list of characters chosen among 'I', 'V', 'X', 'L', 'C', 'D', 'M' and '?'. These characters respectively represent 1, 5, 10, 50, 100, 500, 1000 and 5000. Since these characters are uppercase characters, they need to be quoted in order to avoid the confusion with variables. The program follows:

```
arabicroman(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).
```

Let's recall that the notation `X o Y` stands for `conc(X,Y)`. The following queries translate numbers from arabic to roman and then translate them again into arabic:

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

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

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

Shur coloring problem

When concatenating two lists, the size of the resulting list is the sum of the sizes of the concatenated lists. This second example illustrates the pseudo-arithmetic which is hidden there. It solves a coloring problem which is linked to a property usually know as Shur's lemma.

We consider m colors and a positive integer n. The problem is here to assign to each integer i of the finite sequence $1, 2, \ldots, n$ a color x_i taken in the m colors, such that for all i, j and k between 1 and m, we have:

$i + j = k$ implies the colors x_i, x_j, x_k are not all identical.

I. Shur showed that for any number of colors m, there exists an integer n such that this problem has no solution.

Let's focus on the particular case where $m = 3$, the colors being identified by red, green, blue. The coloring $[x_1, \ldots, x_n]$ can be defined recursively by:

```
coloring([]).
coloring(X o [C]):-
   coloring(X),
   addcorrect(X, C),
   color(C).

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

color(a).
color(b).
color(c).
```

For $n = 13$, we get the 18 following colorings :

```
>> size(X) = 13, coloring(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].
```

starting from $n = 14$, we don't get any coloring :

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

Matrix transposition with concatenation

In Prolog, it is quite difficult to write a matrix transposition program. The natural representation of a matrix is a list of rows, where each row is a list of numbers. The main difficulty lies in the fact that, when writing a transposition program, one must access the columns as well. This is done by the `columnone/3` predicate which separates a matrix into its first column and the rest of the matrix. It is also necessary to create a list where all the rows are empty; this is done by the `zerocolumn/1` predicate which uses a concatenation constraint in order to build a list of all empty rows. For the sake of symmetry, we have also introduced the predicates `rowone/3` and `zerorow/1`.

```
transposition(A,B) :-
    zerorow(A),
    zerocolumn(B).
transposition(A,B) :-
    rowone(A,X,C),
    columnone(B,X,D),
    transposition(C,D).

zerorow([]).

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

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

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

Here is a query which transposes a matrix, and then does it again in order to get the original matrix.

```
>> 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]].
```

Matrix transposition with indexing

When thinking about matrix transposition, one immediately thinks about the formula $a_{ij} = b_{ji}$. Here is another version of the transposition program, which is based on this formula:

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

constraints(0,A,L).
constraints(I,A,L) :-
   ge(I,1),
   constraint(A,[I|L]),
   constraints(I.-.1,A,L).

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

The expression `A:I:J` is interpreted as `((A:I):J)` which in turn is interpreted as the pseudo-term `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 Constraints over integers and Booleans

In many combinatorial problems, integers and Booleans (*i.e.* 0 and 1) play an important role. Here are a few examples of such problems.

Enumerating natural integers

This program enumerates, without forgetting any, all the lists of natural integers, *i.e.* all the lists of positive or null integers. Since this set of lists is infinite, the program only stops when called in a context which binds these integers. The enumeration is made faster by conflicts between the various constraints added. Here is the program:

```
enumeration(L) :- nonnegative(L), from(0,L).

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

from(N,[]).
from(N,La) :-
   dif(La,[]),
   greater(N,La,Lb),
   from(N.+.1,Lb).

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

Here is now an enumeration with a context:

```
>> 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.
```

For your information, here is another enumeration program, which may be more efficient, but which uses two Prolog IV predefined meta-predicates, `glb/2` and `int_size/2`. These are described in the chapter about the Prolog IV predefined predicates. In this program, we assume that the elements on which we are to enumerate have been previously constrained to be integers:

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

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

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

Magic series

Given n, we are interested in finding the finite series x of the form $(x_0, \ldots, x_{n-1})$ such that each x_i is the number of occurrences of the integer i in x. The solutions are the solutions of the conjunction $P_1(x) \wedge \cdots \wedge P_n(x)$ of the following constraints

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

where the expression $(x_j = i-1)$ is 1 or 0, depending on the truth value of the equality constraint in it. Here is the program, which use the integer enumeration program defined above:

```
magicseries(X) :-
   nfirstconstraints(size(X),X),
   enumeration(X).

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

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

Here are a few examples using this program:

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

The 21-square problem (with known sizes)

We consider n squares, of dimensions $a_1 \times a_1, \ldots, a_n \times a_n$, and we want to place them in a square of size $m \times m$. The data are $n = 21$, $m = 112$, and the n-uple $(a_1, \ldots, a_n)$ is equal to

$(55, 42, 37, 35, 33, 29, 27, 25, 24, 19, 18, 17, 16, 15, 11, 9, 8, 7, 6, 4, 2)$.

The solutions are the solutions of all the following constraints. Constraint 3 eliminates most of the symmetries by constraining the first square into having its lower-left corner in the lower-left quarter of the large square, under the positive diagonal. The constraints 4a and 4b are redundant, but they are fundamental in improving the propagation of information, The general spirit of these constraints is based on the ideas in a paper from N. Beldiceanu.

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

Here is now the program, which uses the matrix transposition and integer enumeration programs.

```
filling(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],
   constraints(N,c1,[X,A,M]),
   constraints(N,c1,[Y,A,M]),
   constraints(N,c2,[X,Y,A]),
   constraints(1,c3,[X,Y,A,M]),
   constraints(M,c4,[X,A,M]),
   constraints(M,c4,[Y,A,M]),
   transposition([X,Y,A],F),
   enumeration(X),
   enumeration(Y).

% Creating constraint conjunctions

constraints(0,C,L).
constraints(I,C,L) :-
   ge(I,1),
   constraint(C,[I|L]),
   constraints(I.-.1,C,L).

% The various constraints

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

constraint(c2,[J,X,Y,A]) :-
   constraints(minus(J,1),c2bis,[J,X,Y,A]).
constraint(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)).

constraint(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).

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

In order to run the following query, Prolog IV needs to be started with the following parameters:

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

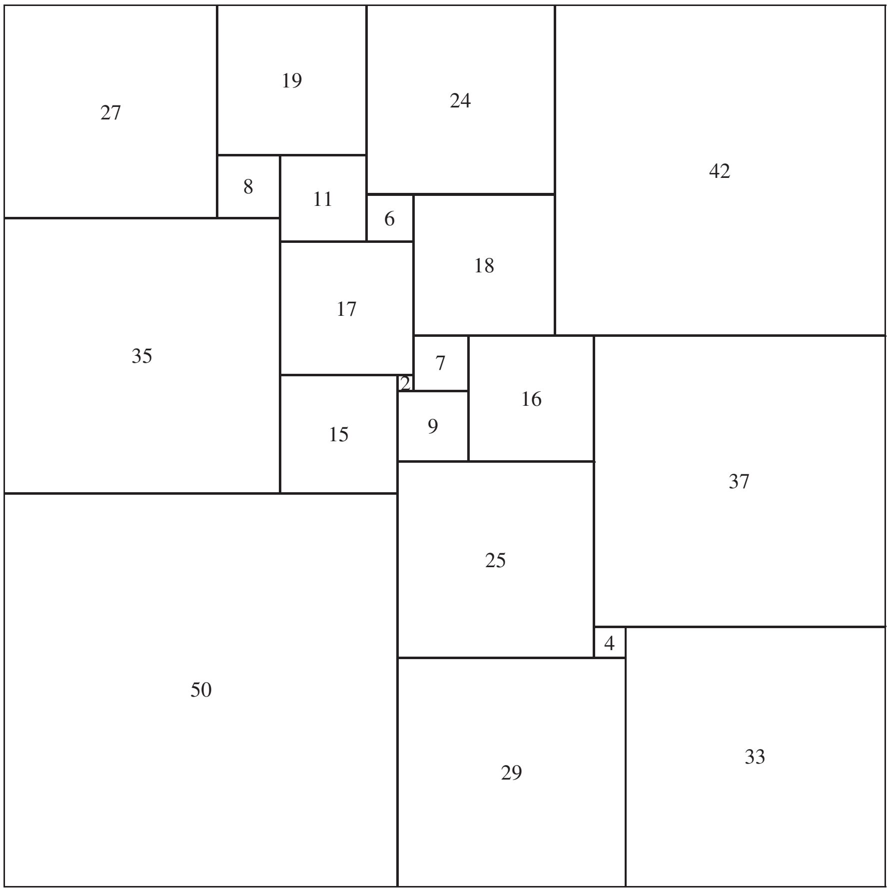
For more information about these parameters, please refer to the *Environment* chapter of the present manual.

```
>> filling(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]]
```

We obtain two symmetrical solutions, which correspond to the drawing below.



2.6.3 Constraints over real numbers

Let's now move on to the handling of continuous problems.

Roots of Wilkinson's polynomial

The following example is a benchmark which is very hard on polynomial root computation programs (in fact, this program goes to the precision limit of IEEE single-precision floating-point numbers). We consider the polynomial:

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

which of course has 20 roots. If we modify it slightly by considering the polynomial:

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

then there are only 10 roots. Let's check whether or not Prolog IV is up to it:

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

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

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

We then use the `realsplit/1`, which is described in the *Prolog IV predefined predicates* chapter of this manual. We obtain the 20 roots by:

```
>> 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.
```

In the case of the altered polynomial, we find many (apparently) parasite solutions but, if we limit x in the interval $[-100, 100]$, we get 10 intervals:

```
>> 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`).
```

Confirming the roots of Wilinson's altered polynomial

In our last example, the 10 intervals we get as answer give us a guarantee that no solution exists outside of these intervals, but they do not guarantee that each interval contains exactly one solution. This doubt can be removed, by performing the following checks:

1. the polynomial's derivative should not have a root within the interval,
2. the values of the polynomial at the interval's bounds do not have the same sign.

It is therefore necessary to compute the Wilkinson's altered polynomial's derivative, which is done by the following predicate:

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

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

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

We can then use a more elaborate query which is able to guarantee the existence and uniqueness of a root (every time that the variable `Certain` takes the value `yes`).

```
>> 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]),yes,no).
Certain = yes, X ~ cc(-`>20.846977`,-`<20.84688`);
Certain = yes, X ~ cc(-`>8.917252`,-`>8.917247`);
Certain = yes, X ~ oo(-`>8.007267`,-`<8.007267`);
Certain = yes, X ~ oo(-`<6.999698`,-`>6.999697`);
Certain = yes, X ~ cc(-`>6.000007`,-`<6.000007`);
Certain = yes, X ~ oo(-5,-`>4.9999995`);
Certain = yes, X ~ oo(-`>4.0000004`,-4);
Certain = yes, X ~ oo(-3,-`>2.9999997`);
Certain = yes, X ~ oo(-`>2.0000002`,-2);
Certain = yes, X ~ oo(-1,-`>0.9999999`).
```

2.6.4 Linear constraints

The two examples below show that it is possible in Prolog IV to use the linear constraints which were also available in Prolog III (when using the pseudo-operations and relations postfixed by `lin`).

Matrix product

This first example is a basic linear example, without order relations. The goal is to express the constraint $A \times B = C$, where A, B and C are matrices (which have the correct size for a matrix product). Here is the program:

```
matrixtimesmatrix(A,B,C) :-
   zeroline(A),
   zeroline(C).
matrixtimesmatrix(A,B,C) :-
   lineone(A,X,Ap),
   lineone(C,Y,Cp),
   size(X) = size(B),
   vectortimesmatrix(X,B,Y),
   matrixtimesmatrix(Ap,B,Cp).

vectortimesmatrix(X,B,[]) :-
   zerocolumn(B).
vectortimesmatrix(X,B,[R|Y]) :-
   columnone(B,Z,Bp),
   vectortimesvector(X,Z,R),
   vectortimesmatrix(X,Bp,Y).

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

zeroline([]).

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

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

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

The notations `R*S` and `R+T` correspond to the pseudo-terms `timeslin(R,S)` and `pluslin(R,T)`. We can now start the query

```
>> 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]],
   matrixtimesmatrix(Ap,B,C),
   matrixtimesmatrix(A,Bp,C),
   matrixtimesmatrix(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]].
```

The 9-square problem (unknown sizes)

This second example has been one of Prolog III's successes[7]. The goal is to assemble n squares of distinct sizes in order to make a rectangle. The sizes of the squares and of the rectangle are unknown. The integer n is known, and

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

$n = 9$. Here is the transposition of the original Prolog III program, which is a good syntax exercise.

```
filledRectangle(A, C) :-
   gelin(A,1),
   distinctSquares(C),
   filledZone([-1,A,1], L, C, []).

distinctSquares([]).
distinctSquares([B|C]) :-
   gtlin(B,0),
   distinctSquares(C),
   outOf(B, C).

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

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

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

Here are the 8 solutions of the problem. In fact, there are two solutions, and the other ones are obtained by symmetry. Here, `A` is the length of the rectangle (its width is set to 1), and `C` is the list of the sizes of the 9 squares.

```
>> size(C)=9, filledRectangle(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].
```