LE MANUEL DE PROLOG IV

Tutoriel
Concepts de base
Primitives
Prolog ISO
Syntaxe
Environnement

PrologIA

Parc Technologique de Luminy – Case 919

13288 Marseille cedex 09 – FRANCE

PROLOG IV est un produit de la société PrologIA, conçu en coopération étroite avec l'équipe *Programmation par Contraintes* du LIM, le Laboratoire d'Informatique de Marseille qui associe l'Université de la Méditerranée, l'Université de Provence et le Centre National de la Recherche Scientifique dans l'URA 1789.
Ont participé à sa réalisation Frédéric BENHAMOU, Pascal BOUVIER, Alain COLMERAUER, Henri GARRETA, Jean-Luc MASSAT, Guy Alain NARBONI, Stéphane N'DONG, Jean-François PIQUE, TOURAÏVANE, Michel VAN CANEGHEM, Eric VÉTILLARD, avec l'aide de Bruno GILLETA, Robert PASERO et Jianyang ZHOU.
Son développement a bénéficié d'études menées dans le cadre de deux projets ESPRIT : le projet de recherche ACCLAIM 7195, portant sur la programmation concurrente par contraintes, et le projet PRINCE 5246, portant sur l'environnement et la partie contraintes d'un Prolog à caractère industriel et financier.

Pour toutes questions concernant ce manuel et ce logiciel, contactez :

PrologIA
Parc Technologique de Luminy – Case 919
13288 Marseille cedex 09 – FRANCE

Tél : 33 91 26 86 36
Fax : 33 91 41 96 37
E-mail: `prologia@prologianet.univ-mrs.fr`

Table des matières

Introduction **1**

1 Tutoriel et Survol de Prolog IV **11**

1.1 Une session prolog sous Prolog IV 11

1.2 D'autres exemples, avec des contraintes ! 14

1.3 L'environnement de programmation 42

1.4 De Prolog III à Prolog IV 50

2 Les Bases de Prolog IV **61**

2.1 Introduction 61

2.2 Syntaxe et sémantique 65

2.3 La structure de base π_4 70

2.4 Axiomatisation de π_4 79

2.5 Structures enrichies 92

2.6 Exemples de programmes en Prolog IV 97

3 Relations Prolog IV **111**

3.1 Introduction 111

3.2 Liste alphabétique 117

4 Prédicats prédéfinis Prolog IV **213**

4.1 Préliminaires 213

4.2 Liste alphabétique 219

5 Prédicats prédéfinis ISO **251**

5.1 Introduction 251

5.2 Préalables 258

5.3 Liste alphabétique 271

6 Syntaxe de Prolog IV **339**

6.1 Mini-glossaire . . . 339
6.2 Les modes d'utilisation de Prolog IV . . . 340
6.3 La logique dans Prolog . . . 340
6.4 Les objets du langage . . . 341
6.5 Lecture de règles et de requêtes . . . 347

7 La syntaxe complète de Prolog ISO **353**

7.1 Notations . . . 353
7.2 Textes et données Prolog . . . 354
7.3 Termes . . . 356
7.4 Unités lexicales . . . 363
7.5 Caractères du processeur . . . 371
7.6 Table des codes des caractères . . . 373
7.7 Lexique . . . 373

8 Environnement **375**

8.1 Les options de lancement . . . 375
8.2 Interruption utilisateur . . . 376
8.3 Compilation des règles . . . 377
8.4 Le débogueur Prolog IV . . . 378

9 L'environnement graphique de Prolog IV **389**

9.1 Lancement de Prolog IV . . . 389
9.2 Configuration initiale . . . 390
9.3 Le panneau principal . . . 390
9.4 La console Prolog IV . . . 391
9.5 La Console Tcl/Tk . . . 393
9.6 Les Editeurs . . . 393
9.7 Le Débogueur . . . 396
9.8 Dialogues . . . 398
9.9 Clavier et souris . . . 400
9.10 Informations diverses . . . 401
9.11 Primitives graphiques . . . 402
9.12 Quelques Problèmes et limitations . . . 404

Introduction

Généralités

LA PROGRAMMATION LOGIQUE AVEC CONTRAINTES, aujourd'hui un domaine scientifique à part entière, matérialise la convergence de deux tendances majeures dans la recherche en informatique de ces dernières décennies.

La première, en amont, concerne le souci constant de faire évoluer le langage Prolog, inventé par Alain Colmerauer à Marseille au tout début des années 70. Cette évolution naturelle s'est orientée vers la conception et la réalisation d'un système de développement de plus en plus ouvert, de plus en plus efficace et de plus en plus tourné vers la conception d'applications réelles. Ce que nous appelons ici application réelle regroupe tout type de développement destiné à résoudre un ou plusieurs problèmes physiques dans un environnement économique complexe. Autrement dit, il ne s'agit pas seulement d'expérimenter, il s'agit de produire, que ce soit des applications utilisées quotidiennement dans l'entreprise, des résultats de recherche performants, ou encore un enseignement adapté à de nouvelles exigences professionnelles. L'un des chemins de cette évolution est celui des contraintes, qui ont permis d'enrichir considérablement le langage en redéfinissant son principe même de fonctionnement.

Cette révolution ne s'est toutefois pas faite en un jour. La définition de Prolog II, au début des années 80 comportait déjà les idées majeures et la machinerie théorique qui allait plus tard donner naissance aux langages de programmation par contraintes. La seconde innovation marquante devra attendre la fin de ces mêmes années 80 et voir l'avènement d'un traitement des problèmes numériques, notamment, en accord avec la philosophie générale du langage, à savoir sa déclarativité et son caractère relationnel.

La seconde tendance mentionnée plus haut est diamétralement opposée. Il s'agit en effet, non pas d'améliorer le langage Prolog pour le rendre plus adapté au développement d'applications industrielles, mais de montrer que ces applications, souvent déjà exprimées en termes de contraintes, ont tout à gagner de la déclarativité, du caractère relationnel et du non-déterminisme inhérent à Prolog. Pour permettre à Prolog d'être efficace dans cet environnement, une évolution est souhaitable, et cette évolution se trouve converger vers les principes d'amélioration du cœur même du langage que nous avons mentionnés plus haut.

Le résultat pratique de ces réflexions s'est traduit à la fin des années 80 par le développement de langages de programmations par contraintes parmi lesquels Prolog III et d'autres langages ont démontré les potentialités du concept dans le monde industriel.

Prolog IV n'est pourtant pas une nouvelle version de Prolog III. Ce précédent langage s'est amélioré au fil du temps, son environnement de programmation s'est enrichi, les plates-formes matérielles et logicielles sur lesquelles il est disponible se sont diversifiées, mais le cœur du langage est resté inchangé depuis son lancement, en 1989. Prolog IV est le résultat à la fois des recherches en amont qui se sont poursuivies depuis lors, et des besoins fondamentaux des utilisateurs tels qu'ils sont apparus au fil du temps. Enfin, une autre étape fondamentale dans l'utilisation industrielle des langages dérivés de Prolog atteint sa phase terminale, puisque l'établissement d'une norme ISO pour Prolog vient d'être publiée.

Pour toutes ces raisons, la famille des Prolog marseillais se devait donc de s'enrichir d'un nouveau membre, mais à l'image de ses prédécesseurs, celui-ci se devait de réunir un certain nombre de qualités indispensables : efficacité du système, rigueur théorique et nouveauté des concepts.

L'efficacité du système est une condition cruciale à la réussite d'un langage, mais aussi à la réussite de tous les langages construits autour de Prolog. Celle-ci est assurée en Prolog IV grâce à un compilateur optimisé et à l'utilisation de résolveurs de contraintes extrêmement performants que nous présenterons plus en détail dans la suite de ce document.

Contrairement à quelques idées reçues, il serait illusoire de croire que la rigueur théorique se résume à un souci d'esthétique. Les langages de programmation par contraintes tels que Prolog IV, par une volonté constante d'amélioration de leur puissance d'expression et de leurs performances deviennent complexes dans leur conception. L'unique garantie que l'exécution des programmes calcule les résultats attendus (et parfois des résultats tout simplement corrects) réside dans la précision de la définition d'une sémantique formelle en amont du développement. Dans ce contexte, la notion de formalisme s'interprète très pragmatiquement pour le programmeur en termes de correction, de robustesse et de réutilisabilité, en un mot de qualité et de fiabilité du logiciel. Le cas de Prolog IV est exemplaire. L'introduction d'un certain degré d'approximation pour le traitement des contraintes, la cohabitation et la coopération de résolveurs de contraintes complexes au cœur même des mécanismes fondamentaux du langage ont constitué un perpétuel challenge à la fois pour la définition d'un langage simple et cohérent et pour l'élaboration d'un canevas formel qui rende compte très précisément du comportement du système lors de l'exécution des programmes. Cette tâche seule a nécessité plusieurs années de recherche.

Les principaux fondements théoriques ainsi que la sémantique formelle de Prolog IV, dus à Alain Colmerauer, sont présentés dans ce document. Le lecteur peu rompu à la pratique de ce genre de littérature pourra bien entendu tirer un parti optimal du langage sans prendre connaissance de cette partie du

manuel de référence dans la mesure où tous les concepts importants sont également exposés de manière plus pratique et peut être plus intuitive tout au long des manuels de référence et d'utilisation. Il n'en reste pas moins que cet exposé donnera un éclairage différent au lecteur curieux et constituera une base de travail privilégiée pour préparer un cours sur le langage, en second cycle des Universités, par exemple.

Enfin, la nouveauté des concepts marque la lignée des Prologs marseillais. Prolog IV n'échappe pas à la règle : il réalise à la fois l'unification des résultats obtenus jusqu'à ce jour et généralise la résolution de contraintes en y incorporant de nouvelles techniques. Tous les résolveurs de contraintes ont été revus, que ce soit par souci d'efficacité (contraintes linéaires) ou par souci de simplification et respect de la norme ISO (contraintes sur les listes). Un nouveau résolveur, dont les fondements proviennent de l'arithmétique d'intervalles permet d'aborder les contraintes non-linéaires sur les réels, et unifie les contraintes sur les domaines finis et les Booléens. Cette généralisation autorise, en outre, le mélange harmonieux de contraintes sur des algèbres jusqu'alors totalement séparées.

Qu'est ce que la programmation par contraintes ?

Avant d'introduire brièvement les caractéristiques du langage, nous revenons sur le concept même de programmation par contraintes et nous proposons quelques éléments de réponses à certaines questions préalables qui peuvent se poser à l'utilisateur peu familier du domaine.

Que sont les contraintes ? On appelle *contrainte* l'expression de toute relation qui lie un certain nombre d'objets qui prennent leurs valeurs dans un certain domaine. A ce titre, les équations de la physique qui imposent à certaines quantités d'être égales, les impératifs qui imposent des relations particulières aux éléments d'une machine, au déroulement d'un process, ou qui imposent des conditions pour la réalisation d'une tâche sont des contraintes. Le but de la programmation par contraintes est de permettre au programmeur de se concentrer sur l'expression de ces relations (la modélisation du problème) par opposition à la recherche de solutions au système constitué de l'expression de ces relations (la résolution du problème). On demande généralement à ces solutions soit d'être réalisables (elles obéissent aux contraintes) soit d'être optimales par rapport à une certaine fonction de coût.

Si le but est d'écrire des équations, des inéquations ou plus généralement des relations pour que le système se charge ensuite de les résoudre, comment se justifie alors l'activité de programmation ? C'est là l'apport fondamental de cette technologie. En effet, nombre de systèmes spécialisés ont pour vocation d'effectuer efficacement ce genre de traitement. La popularité et l'efficacité de la programmation par contraintes viennent du fait que le programmeur dispose en plus d'un langage de programmation complet et généraliste dans lequel ces techniques de résolutions sont profondément intégrées. Ce langage est très souvent indispensable pour :

1. exprimer des contraintes,
2. contrôler leur prise en compte par le système,

3. contrôler certaines phases clefs de leur résolution,
4. programmer d'autres parties du système qui ne se réduisent pas à cette résolution.

Dans ce cas, si les notions de langage hôte et de résolution de contraintes sont distinctes, pourquoi avoir choisi Prolog ? Là encore, la réponse est tout à fait naturelle. Comme nous l'avons esquissé précédemment, les deux étapes principales du traitement de ce type de problème sont :

1. l'expression du problème,
2. le parcours d'un arbre de recherche pour isoler les solutions

A chacune de ces étapes correspond une des deux caractéristiques majeures et discriminantes du langage Prolog : la *déclarativité* et le *non-déterminisme*. Le caractère déclaratif et relationnel de Prolog en fait un outil absolument naturel pour la modélisation. Il n'est pas nécessaire de traduire en *actions* l'expression du problème, puisque le langage lui même en permet un codage déclaratif pratiquement immédiat. D'autre part, il n'est pas non plus nécessaire de coder le parcours de l'arbre de recherche, ce type de traitement faisant déjà partie intégrante du modèle d'exécution du langage.

Présentation succincte du langage

Plus précisément, quelles sont les caractéristiques du langage et quelles sont les nouveautés par rapport à Prolog III ? Dans ce qui suit, on abordera à la fois des généralités conceptuelles (que sont les nombres en Prolog IV, quels sont les types d'algorithmes utilisés pour résoudre les contraintes, etc.) mais également quelques points plus précis. Lors d'une première lecture, le lecteur pourra sauter les parties qu'il juge trop techniques en fonction de son degré de familiarité avec Prolog et la programmation par contraintes et se référer à la section introductive du manuel.

Modes syntaxiques

Tout d'abord la partie Prolog. Celle-ci a évolué pour se conformer au standard ISO. Prolog IV est donc le premier langage de la lignée à ne plus proposer comme option une syntaxe dite «marseillaise». On se rendra donc ici à l'évidence avec quelque nostalgie, les discussions passionnées sur la syntaxe des variables et sur les mérites respectifs des symboles « `:-` » et « `->` » font aujourd'hui partie de l'histoire de l'informatique. Concédons que c'est bien là l'un des objectifs de toute normalisation et il y a fort à parier que la plupart des utilisateurs ne s'en plaindront pas.

En fait, on dispose en Prolog IV de deux modes, appelés *mode ISO* et *mode étendu*. Sans rentrer dans les détails, le mode ISO, comme mentionné précédemment se conforme à la norme, y compris donc la définition d'une longue liste de prédicats prédéfinis que l'on trouvera sous le même nom dans le manuel de référence. Ce qui est plus intéressant est que l'on peut, en mode ISO, utiliser toute la puissance de Prolog IV et en particulier les contraintes. Pour se faire, il suffit d'utiliser les prédicats de contraintes prédéfinis dont on trouvera également le détail dans le manuel de référence. De ce fait, la *totalité* des fonctionnalités de Prolog IV sont accessibles en mode ISO. Ceci introduit donc la

question suivante : « pourquoi un second mode syntaxique ? ». La réponse est très simple. Lorsque l'on utilise les contraintes de manière intensive, un grand nombre de facilités syntaxiques sont souhaitables. La première de ces facilités concerne l'utilisation de termes pour représenter des expressions numériques. Par exemple on souhaite pouvoir écrire la contrainte $2x + y = 5z - t$ de la façon suivante :

```
2*X + Y = 5*Z - T
```

et non pas, comme il faut le faire en mode ISO :

```
timeslin(X1,2,X),
addlin(X2,X1,Y),
timeslin(Y1,5,Z),
minuslin(Y2,Y1,T),
X2 = Y2.
```

La remarque importante ici est que la première manière d'écrire a également un sens en mode ISO, mais ce n'est pas celui que l'on recherche ici. En effet, le traitement de cette égalité a pour effet de chercher des valeurs pour les variables qui vérifient l'égalité entre les deux termes Prolog `2*X + Y` et `5*Z - T`, c'est à dire une égalité entre deux arbres. Cette équation n'a bien entendu pas de solutions (dans le jargon logique souvent utilisé en Prolog, on dirait que les deux termes ne sont pas *unifiables*). On voit donc immédiatement l'intérêt d'un mode plus approprié au traitement des contraintes. Notons au passage que de nombreuses autres possibilités sont également offertes dans le mode *étendu* de Prolog IV, comme l'utilisation de pseudo-termes (voir la définition de ces objets syntaxiques dans le manuel de référence). Intuitivement, l'utilisation de ces pseudo-termes permet de simplifier un certain nombre de notations en les rendant plus « fonctionnelles » comme par exemple :

```
Y = cos(X)
```

au lieu de la notation sous forme relationnelle :

```
cos(Y,X)
```

ou de fixer des domaines de valeurs possibles pour les variables (très grossièrement, on peut faire un parallèle avec des types) :

```
[X,Y,Z,T] ~ [int n ge(0), le(0), co(1,2), list]
```

Encore une fois sans rentrer dans les détails, on exprime ainsi de manière très concise que `X` doit être un entier positif ou nul, que `Y` doit être un nombre réel strictement négatif, que les valeurs de `Z` sont à prendre dans l'intervalle semi-ouvert $[1, 2)$ et enfin que `T` doit représenter une liste.

Contraintes sur les arbres et les listes

Prolog IV hérite des précédents Prolog de la même lignée (Prolog II, Prolog II+ et Prolog III) du traitement de contraintes sur les arbres rationnels (une certaine famille d'arbres infinis) qui inclut le traitement des équations et des diséquations (il s'agit de la relation $\neq$ ou pour parler différemment du célèbre prédicat `dif/2` de Prolog II).

Les listes sont représentées par des arbres binaires de manière habituelle (et conforme à la norme ISO). La notion même de liste a été homogénéisée par rapport à Prolog III et on ne fait plus de différence entre les listes sur lesquelles on peut définir des contraintes de concaténation et les listes usuelles. Les deux principales contraintes restent la concaténation et la contrainte de taille (ou longueur) d'une liste, auxquelles a été ajoutée la très utile contrainte `index/3` qui exprime qu'un objet est le *énième* élément d'une liste.

Mentionnons également ici que la notion de *retardement*, que l'on retrouvera lorsque l'on abordera les contraintes linéaires, a été conservée. Rappelons en deux mots qu'une contrainte est retardée si elle ne satisfait pas un certain nombre de conditions qui permettent son traitement immédiat. Lors de l'exécution d'un programme, une telle contrainte sera ajoutée au système de contraintes courant si ces conditions sont satisfaites ultérieurement. La définition précise des conditions qui régissent le retardement des contraintes sur les listes sera donnée dans le chapitre consacré à ces contraintes, mais remarquons d'ores et déjà que ces conditions diffèrent de celles qui conditionnent le retardement en Prolog III. Grossièrement toute contrainte du type :

```
L = L1 o L2.
```

qui se lit « la liste `L` est formée de la concaténation des listes `L1` et `L2` » est ajoutée au système de contraintes courant si deux au moins des tailles des listes `L, L1` et `L2` sont connues. La contrainte est retardée sinon.

Il existe de nombreuses autres différences avec Prolog III concernant les listes. Ces différences seront détaillées dans la suite de ce document.

Contraintes numériques

C'est de très loin le domaine le plus riche de Prolog IV. Formellement c'est extrêmement limpide, la puissance absolue conférée par l'abstraction mathématique permet de définir toute étiquette numérique d'un arbre rencontré en Prolog IV comme un nombre réel. Cela dit, comme tout mortel rompu aux approximations que la réalité impose à nos réalisations humaines le sait fort bien, la représentation des nombres réels en machine n'est pas sans poser de nombreux problèmes pratiques dont le moindre n'est sans doute pas que, par essence, tous les calculs (ou presque) opérés par un ordinateur sont faux. Ces problèmes sont résolus très imparfaitement par la représentation des nombres réels par des nombres flottants, dont la structure suit généralement les recommandations de la norme IEEE pour ces mêmes flottants. C'est le choix (réaliste) de la norme Prolog qui impose donc le comportement que vous rencontrerez en Prolog IV en mode ISO.

En mode étendu, le comportement est totalement différent. La notion même de nombre flottant disparaît, et chaque valeur numérique est représentée par un rationnel en précision infinie, c'est à dire par un couple d'entiers numérateur-dénominateur où les entiers sont aussi grands que nécessaire (dans la limite de la mémoire disponible bien sûr). Jusque là, rien de très différent de ce à quoi nous avait habitué, dans un souci d'exactitude des calculs, le langage Prolog III.

La différence majeure ainsi que l'une des innovations principales de Prolog IV réside dans le fait que le langage permet de traiter des contraintes numériques linéaires sur les rationnels , non-linéaires (y compris non-polynomiales) sur les réels, des contraintes entières (domaines finis) et enfin des contraintes booléennes ainsi que des contraintes «mixtes» sur plusieurs sous-domaines.

Pour résoudre ces contraintes, deux algorithmes de résolution (on dit parfois résolveurs) principaux sont utilisés. Tout d'abord, un algorithme spécifique est dédié principalement à la résolution des contraintes *linéaires*, c'est à dire les contraintes représentables sous la forme $a_0 + a_1x_1 + a_2x_2 + \ldots + a_nx_n \diamond 0$ où $\diamond$ peut être le signe $=, \neq, >, \geq, <$ ou $\leq$. Cet algorithme, comme en Prolog III est une combinaison d'une procédure d'élimination de Gauss et d'un Simplex incrémental dont la particularité est de détecter les variables *figées* (variables dont l'ensemble des valeurs est réduit à un seul élément). Cet algorithme gère également le traitement retardé de certaines multiplications non-linéaires.

Le second algorithme, qui traite principalement tous les autres types de contraintes numériques est un algorithme de *propagation de domaines*. Essentiellement, cet algorithme calcule localement des domaines de valeurs autorisés pour les variables apparaissant dans une contrainte et propage ces domaines aux autres contraintes qui font intervenir ces mêmes variables. Cette propagation s'opère jusqu'à ce qu'un état stable soit atteint. La séparation des solutions du système initial s'opère alors par une technique de «diviser et conquérir» qui revient principalement à effectuer une dichotomie sur les domaines de valeurs possibles et à explorer l'arbre de recherche binaire correspondant à ces deux choix. Il est extrêmement intéressant de noter que cet algorithme généralisant le traitement des domaines finis et donc des contraintes booléennes permet de ramener plusieurs classes de contraintes à un même niveau et donc de combiner étroitement des domaines généralement considérés comme disjoints. On peut ainsi partager des variables entre des contraintes entières et réelles et même des contraintes booléennes (considérées ici comme des contraintes en 0/1).

Le choix de l'algorithme, particulièrement lorsque les contraintes peuvent être traitées par les deux méthodes est laissé au programmeur. Par exemple, pour ajouter la contrainte $2x - \frac{1}{3}y = 7$ au système de contraintes courant traité par Gauss-Simplex on écrira :

```
2*X - 1/3*Y = 7
```

Pour ajouter la même contrainte au système de propagation d'intervalles, on écrira :

```
2.*.X.-.1/3.*.Y = 7
```

Notons au passage que rien n'interdit d'ajouter les deux contraintes à la fois. A ce propos, se pose la question de savoir comment les deux résolveurs communiquent. La réponse est encore une fois très simple. Les résolveurs communiquent uniquement par le biais des variables figées. Ainsi, si le domaine des valeurs possibles d'une variable est réduit à un seul élément par l'un des deux algorithmes et que cette variable apparaît dans une contrainte traitée par l'autre algorithme, ce dernier tiendra compte de cette information. Toute infor-

mation concernant une autre modification de l'ensemble des valeurs possibles restera locale à l'algorithme qui a effectué cette modification.

Environnement de programmation

Prolog IV n'est pas seulement le compilateur d'un concept de programmation sophistiqué. Le système est doté d'un environnement de programmation multi-fenêtré complet et convivial qui comprend son propre éditeur graphique muni de fonctions multiples, un vérificateur de syntaxe et un débogueur symbolique très soigné spécialement adapté à ce type de programmation. Toutes les facilités de communication avec l'extérieur sont disponibles, ainsi que la possibilité de développer des applications graphiques.

Domaines d'applications

Prolog IV est un langage généraliste muni de puissantes capacités de résolution de contraintes complexes. A ce titre, il s'impose comme un choix privilégié dans un très grand nombre d'applications qui peuvent tirer à la fois profit de ces deux caractéristiques. Parmi celle-ci, on peut citer toutes les applications d'ordonnancement et de planification, le développement de systèmes d'aide à la décision, la simulation de processus physiques complexes, l'optimisation sous contraintes et de manière générale la résolution de problèmes. Les techniques de résolution de contraintes mises à la disposition du programmeur ont prouvé leur efficacité, bien souvent supérieure à celle de systèmes spécialisés, particulièrement lorsque le problème à résoudre ne rentre pas exactement dans un cadre mathématique prédéfini (job-shop pur, voyageur de commerce pur, etc), ce qui s'avère être le cas de nombre d'applications réelles où de multiples contraintes, annexes au problème théorique mais cruciales dans la mise en œuvre des solutions, sont à prendre en compte. Enfin, la souplesse d'utilisation et le caractère concis, précis et déclaratif du langage permettent un développement rapide ainsi qu'une évolution et une maintenance simplifiées des systèmes développés.

Présentation des manuels

Le présent manuel est organisé en cinq parties principales. La première partie se compose d'une prise de contact rapide et d'un tutoriel qui aborde graduellement au travers d'exemples très simples les principaux concepts à la base du langage. La seconde partie définit plus en profondeur ces divers concepts et insiste tour à tour sur les différentes structures algébriques supportant les contraintes de Prolog IV, au travers d'exemples de programmes. La troisième partie compose le manuel de référence, dans lequel tous les prédicats prédéfinis sont détaillés. Une attention toute particulière à été portée à l'explicitation des différences avec Prolog III, ainsi qu'à la mise en valeur des fonctionnalités relatives aux contraintes qui ne sont, par essence, pas documentées dans la norme ISO. Les chapitres suivants décrivent précisément la syntaxe dans les deux modes et l'environnement de programmation. Le manuel se termine par un index général.

Pour terminer cette introduction, nous espérons que vous prendrez autant de plaisir à utiliser ce langage que nous en avons eu à le concevoir, l'implémenter et le rendre à la fois le plus précis, le plus efficace et le plus agréable possible à programmer et à utiliser. Le monde de la programmation par contraintes est vaste, excitant et parfois mystérieux mais les réussites sont le plus souvent

à la hauteur de l'énergie investie. Nous sommes fiers que vous ayez choisi Prolog IV comme outil de développement pour la résolution de vos problèmes et nous serons bien entendu très heureux de répondre à toutes vos remarques, critiques et suggestions.