

---

# L'enseignement de la programmation logique à l'École des Mines de Nantes

**Narendra Jussien**

*École des Mines de Nantes  
4 rue Alfred Kastler – BP 20722  
F-44307 Nantes Cedex 3  
Narendra.Jussien@emn.fr*

---

*RÉSUMÉ. La programmation logique est enseignée à l'École des Mines de Nantes depuis 1994. Suite à un doublement du volume horaire en 1998 (de 15 à 30 heures), le contenu du cours a beaucoup évolué pour proposer un cheminement plus complet depuis la logique formelle jusqu'à la programmation en logique. Cet article présente ce cours dans son organisation et sa philosophie.*

*ABSTRACT. Logic programming is taught at École des Mines de Nantes since 1994. Our course is organized to provide a complete path from logic to logic programming. This document presents the current organization of the course.*

*MOTS-CLÉS : programmation logique, logique formelle, enseignement*

*KEYWORDS: logic programming, logic, teaching*

---

## 1. Introduction

La programmation logique est enseignée à l'École des Mines de Nantes depuis 1994. Suite à une augmentation du volume horaire en 1998 (de 15 à 30 heures), le contenu du cours a évolué pour proposer un cheminement plus complet depuis la logique formelle (calcul propositionnel et calcul des prédicats) jusqu'à la programmation en logique. Ce cours comporte de nombreux travaux pratiques mettant l'accent sur les applications de PROLOG plutôt que sur le langage lui-même. Ce document présente l'organisation du cours telle qu'il est dispensé actuellement.

Après une présentation du contexte spécifique de l'enseignement dans une école d'ingénieurs comme l'École des Mines de Nantes, les objectifs du cours sont détaillés, puis le découpage en cours-TP est précisé. Enfin une description détaillée des sujets de TP est proposée, avant de faire un bilan de cet enseignement.

## 2. Contexte

Créée en 1990 par le ministère de l'industrie avec le concours des collectivités locales, l'École des Mines de Nantes forme en quatre ans<sup>1</sup> des ingénieurs compétents en génie des systèmes industriels, qui maîtrisent les méthodes nécessaires à la conception, l'étude, le développement et la maintenance des systèmes de production.

Pour développer cet ensemble de compétences, l'École des Mines de Nantes axe son enseignement sur trois domaines majeurs :

- l'informatique et les systèmes de communication ;
- l'automatique et la productique ;
- les sciences de l'homme et la société.

Ces trois domaines s'enrichissent et se déclinent dans le cadre des options en quatrième année.

Les quatre années d'enseignement se décomposent en :

– un **tronc commun**, dispensé pendant les deux premières années, dont l'informatique est un pilier. Celle-ci est introduite par un cours d'initiation à la programmation de 45 heures. Pour faire le lien avec l'univers mathématique, l'équipe pédagogique a choisi d'utiliser pour ce cours un langage fonctionnel : HASKELL. Ce cours est suivi d'un cours de 60 heures d'algorithmique et structure de données (pour lequel nous introduisons le langage JAVA). La deuxième année introduit la programmation par objets dans un cours de 30 heures (toujours en JAVA). Au cours de cette même année, une introduction de 30 heures aux bases de données et une introduction de 45 heures aux systèmes et réseaux sont finalement proposées.

---

1. Les élèves de l'École des Mines de Nantes sont recrutés par concours (concours commun des écoles des Mines) à l'issue de la première année de Classe Préparatoire aux Grandes Écoles (CPGE).

– une **troisième année** dite de pré-orientation. L'année est décomposée en modules optionnels dont certains sont requis pour les options de quatrième année. Par exemple, pour les options orientées vers l'informatique, on trouve des modules de génie logiciel, de compilation, d'intelligence artificielle, ...

– une **quatrième année** dite d'option qui donne la coloration finale de nos ingénieurs. L'École des Mines de Nantes propose neuf options parmi lesquelles deux émanent du département informatique. L'une d'elles propose une spécialisation en génie informatique pour l'aide à la décision (GIPAD). Cette spécialisation est centrée sur les aspects optimisation et résolution de problème complexes d'aide à la décision (Recherche Opérationnelle, Programmation par Contraintes, ...).

Le cours de **programmation logique** est un module optionnel du premier trimestre de troisième année. Il est requis pour l'option GIPAD. Il est aussi requis pour un module d'intelligence artificielle<sup>2</sup> de 30 heures proposé au deuxième trimestre.

### 3. Les objectifs du cours de Programmation Logique

Les objectifs du cours de Programmation Logique sont multiples :

- introduction à une nouvelle façon de penser la programmation ;
- résolution de problématiques nouvelles ;
- présentation d'un outil privilégié pour le maquettage et l'implantation d'algorithmes liés à l'intelligence artificielle ;
- introduction aux concepts de base de la programmation par contraintes pour préparer à la fois le module d'intelligence artificielle et l'option GIPAD.

On notera que malgré le contexte spécifique d'une école des Mines, le but de ce cours n'est pas nécessairement de montrer que PROLOG est *utile* dans l'industrie. Il s'agit en effet plus un cours d'ouverture sur un mode de pensée que de l'apprentissage d'un *n*ième langage. Trente heures (soit 24 séances d'une heure et quart) sont dédiées à ce cours.

### 4. Organisation du cours

L'organisation du cours est orientée par un choix délibéré d'éviter de se contenter de l'aspect *magique* de PROLOG mais plutôt sur une compréhension des mécanismes conduisant à un tel niveau de déclarativité. Ainsi, la première partie du cours est consacrée à des *rappels* de logique formelle (calcul propositionnel et calcul des prédicats) pour expliquer concrètement comment on peut *programmer* avec la logique. Ensuite,

---

2. Ce module, dont les TP sont réalisés principalement en PROLOG, aborde différents thèmes : algorithmes de recherche, principes de programmation des jeux, problèmes de satisfaction de contraintes, recherches locales, traitement automatique du langage naturel, représentation des connaissances et raisonnement temporel.

une présentation classique et relativement rapide du langage est proposée, illustrée par des travaux pratiques (axés sur des applications plutôt qu'uniquement sur la manipulation du langage).

#### 4.1. *Les cours magistraux*

L'enseignement en cours magistraux occupe la moitié du temps disponible, soit 12 séances d'une heure et quart. Nous présentons ici le contenu et l'objectif pédagogique de chacun de ces cours dans l'ordre chronologique.

##### – Calcul propositionnel (3 séances)

Nous en présentons les aspects syntaxiques, les aspects sémantiques (tables de vérité, diagrammes de Karnaugh, tautologies, formes normales) et les aspects déductifs (conséquence logique, démonstration, principe de résolution, propriétés).

Le but de ces trois séances est de mettre en évidence l'intérêt d'une forme normale pour les démonstrations, la différence entre conséquence logique et démonstration, le principe de résolution et la notion de preuve par réfutation.

À l'appui de ces séances, un certain nombre d'exercices<sup>3</sup> sont proposés sous forme de pause ludique pour illustrer les principaux résultats. En voici deux exemples :

##### **Exemple 1 (Exercice d'application des diagrammes de Karnaugh) :**

Une banque veut installer un nouveau coffre-fort. Le coffre-fort ne doit pouvoir être ouvert que :

- par le directeur et le secrétaire général ;
- ou par le directeur, le comptable et le caissier ;
- ou par le secrétaire général, le comptable et l'adjoint du caissier.

Ceci implique bien sûr qu'aucune de ces personnes ne peut ouvrir le coffre-fort seule.

**Combien faut-il installer de serrures au minimum et comment répartir les clés de ces serrures ?**

##### **Exemple 2 (Démonstration dans le calcul propositionnel) :**

Quatre personnes sont accusées d'un crime : André, Bernard, Charles et Didier. Après enquête, on peut dire que :

- si André et Bernard sont coupables alors Charles est complice ;
- si André est coupable alors Bernard et/ou Charles sont/est complice(s) ;
- si André est innocent alors Charles est coupable.

**Charles est-il coupable ?**

---

3. Beaucoup proviennent de [FRI 86, L'H 98].

– **Calcul des prédicats** (3 séances)

Nous présentons d'abord les limites du calcul propositionnel. Puis, les aspects syntaxiques (quantificateurs), sémantiques (termes/atomes, modèle/interprétation, forme clausale) et déductifs (théorème de Herbrand, validation de raisonnement, unification) du calcul des prédicats sont introduits.

Le but de ces trois séances est de mettre l'accent sur la validation de raisonnement et de montrer que l'unification peut s'apparenter à une forme de calcul (d'où la notion de *programmation* en logique).

Là encore, ces séances sont illustrées par des exercices<sup>4</sup> ludiques pour illustrer les résultats.

**Exemple 3 (Validation de raisonnement) :**  
**Le raisonnement suivant est-il valide ?**

- un dragon est heureux si tous ses enfants peuvent voler ;
- les dragons verts peuvent voler ;
- un dragon est vert s'il a au moins un parent vert ou rose ;
- *donc* : les dragons verts sont heureux.

**Exemple 4 (Raisonnement logique) :**

On sait que :

- Quand je résous un exercice de logique sans ronchonner, vous pouvez être sûr que c'en est un que je comprends ;
- Ces sorites ne sont pas disposés régulièrement, comme ceux auxquels je suis habitué ;
- Aucun exercice facile ne me donne mal à la tête ;
- Je ne comprends pas les exercices qui ne sont pas disposés régulièrement, comme ceux auxquels je suis habitué ;
- Je ne ronchonne jamais devant un exercice, à moins qu'il ne me donne mal à la tête.

**Que peut-on en conclure ?**

– **Introduction à PROLOG** (4 séances)

Nous consacrons ensuite 4 séances aux premiers pas en PROLOG : une base de connaissances sur la famille – DATALOG, premiers prédicats, retour arrière, lien avec la logique, réversibilité, arithmétique, listes, coupure, négation par l'échec, termes structurés. Quelques exercices de bases<sup>5</sup> viennent appuyer ce cours :

4. La plupart proviennent du cours d'Yvon L'Hospitalier à l'Institut de Mathématiques Appliquées, Angers.

5. [STE 86, O'K 92, GIA 85] ont été de bonnes sources d'inspiration.

- sur l'arithmétique, des prédicats de base (`min/3`, `max/3`, `pgcd/3`, ...), des petits exercices (repas équilibrés, arithmétique réversible – successeurs de zéro, ...);
- sur les listes : des prédicats de base (`efface/3`, `reverse/2`, `permutation/2`, ...) et d'autres petits exercices dans l'esprit de l'exemple 5 ci-après;
- sur la coupure et la négation par l'échec : des exercices autour d'une bonne utilisation de la coupure et des propriétés de la négation par l'échec ;
- sur les termes structurés : réalisation d'un évaluateur d'expressions booléennes et réalisation d'un dérivateur symbolique.

**Exemple 5 (Les mutants) :**

Considérons la base de faits suivante :

```
animal(alligator). animal(tortue). animal(caribou). animal(ours).
animal(vache). animal(cheval). animal(lapin). animal(pintade).
```

**Écrire le prédicat mutant/1 tel que :**

```
?- mutant(X).
   X = alligatortue ;
   X = caribours ;
   X = vacheval ;
   X = chevalligator ;
   X = chevalapin ;
   X = lapintade ;
   No
```

– **Méta-interprétation** (2 séances)

Enfin, nous utilisons les deux dernières séances pour réaliser un méta-interprète PROLOG en PROLOG. Le but est de montrer comment on peut modifier rapidement le fonctionnement de base du langage. Ce cours est illustré par un TP spécifique, détaillé dans la section 4.2.

Nous concluons le module de programmation logique en répondant à une préoccupation récurrente des élèves dans le contexte de l'école des Mines : l'utilisation de PROLOG dans l'industrie. C'est l'occasion de présenter quelques systèmes commerciaux et de donner quelques pointeurs.

**4.2. Travaux pratiques**

Les travaux pratiques pour le cours de Programmation Logique occupent aussi 12 séances d'une heure un quart. Les deux premiers TP (4 séances) sont consacrés à des manipulations du langage.

– **Manipulations de listes et parcours de graphes** (2 séances)

Ce premier TP est une application directe des exercices sur la manipulation des listes en PROLOG vus en cours et une application à la recherche de chemins (valués ou non) dans un graphe.

– **Césure des mots** (2 séances)

Ce deuxième TP<sup>6</sup> est consacré à une application de PROLOG pour réaliser un outil de césure des mots basé sur un certain nombre de règles simples (et leurs cas particuliers). Ce TP permet d'introduire l'utilisation de la coupure et de préciser le fonctionnement d'un interprète PROLOG (importance de l'ordre des clauses, ...).

**Exemple 6 (Césure des mots) :**

On souhaite définir un prédicat PROLOG qui réalise la décomposition de mots en syllabes. Pour cela, chaque mot est représenté sous la forme d'une liste de caractères. On souhaite obtenir un prédicat *coupure/2* tel que :

?- *coupure*( [p,r,o,l,o,g], X ).

X = [p,r,o,-,l,o,g]

?- *coupure*( [c,o,u,p,u,r,e], X ).

X = [c,o,u,-,p,u,-,r,e]

*La suite du sujet présente un certain nombre de règles qui président à la césure des mots. Par exemple :*

- une consonne placée entre deux voyelles commence une nouvelle syllabe.  
Exemples : la-mi-noir, pro-log, cou-pu-re.

- de deux consonnes placées entre deux voyelles, la première termine la syllabe courante, et la seconde commence la syllabe suivante. Exemples : ar-  
gent, mas-sif, ar-bus-te.

Les trois TP suivants (y compris l'évaluation<sup>7</sup>) sont plus proches de véritables applications. Pour ces TP, le sujet est nettement plus étoffé (entre 4 et 7 pages) : une introduction générale à la problématique abordée est proposée, suivie d'une partie partie assez dirigée pour obtenir un outil complètement fonctionnel rapidement et d'une partie plus ouverte pour étendre de manière significative l'outil réalisé.

– **Un système-expert de reconnaissance d'annélides polychètes**<sup>8</sup> (3 séances)

Ce premier TP applicatif est consacré à la réalisation d'un système-expert de reconnaissance de vers marins. Ce TP permet de montrer que l'on peut réaliser très rapidement en PROLOG un vrai système-expert. Une base de connaissances est fournie<sup>9</sup> et le TP est découpé en trois parties :

- réalisation d'un moteur d'inférences en chaînage arrière. Ce moteur est amélioré par le stockage des informations démontrées pour accélérer la suite de la démonstration.

- réalisation de modules d'explications pour le moteur : un module d'explication *comment* (explication des démonstrations), et un module d'explication *pourquoi*

6. Ce TP, comme le précédent, est largement inspiré de sujets de TP proposés par Patrice Boizumault successivement à l'Institut de Mathématiques Appliquées, Angers et à l'École des Mines de Nantes.

7. L'évaluation est toujours un sujet très important pour les élèves !

8. Vers marins.

9. Il s'agit d'une partie d'une base réelle provenant du système NEREIS [JUS 94].

(explication des questions posées par le système).

- amélioration de la convivialité du système développé en prenant en compte des faits négatifs. Cette partie n'est pas du tout guidée.

– **Méta-programmation** (2 séances)

Ce TP, plus abstrait, introduit à la réalisation de méta-interprètes PROLOG en PROLOG. Il est organisé en trois parties :

- la première partie présente le méta-interprète de base que l'élève est amené à munir d'une trace explicite (indentée).

- la seconde partie introduit un méta-interprète avec continuation (ce méta-interprète sert de point de départ au TP suivant). Pour cette version, une base à *trous* est fournie (cf. exemple 7). L'idée est aussi d'introduire une trace explicite pour bien comprendre les différences entre les deux versions.

- Enfin, un méta-interprète en largeur est proposé. Un certain nombre d'outils sont introduits et la notion de programme normalisé est présentée. L'accent est mis au final sur l'intérêt d'un tel interprète pour dépasser les limitations ou écueils habituels de PROLOG.

**Exemple 7 (Méta-interprétation avec continuation) :**

```

prouve_cont(true, ?? ).
prouve_cont(?? , ??) :-
    prouve_cont(??,??).
prouve_cont((A,B), ?? ) :-
    prouve_cont(A, ?? ).
prouve_cont( But,Cont ) :-
    predefini(But), !,
    But,
    prouve_cont(?? , ??).
prouve_cont(But, ??) :-
    clause(But,Corps),
    prouve_cont(??,??).

```

– **Un solveur de contraintes** (3 séances)

Ce TP est consacré à l'ouverture des élèves à une extension particulière de PROLOG : la programmation par contraintes. Ce TP est utilisé, d'une part, comme une introduction à la programmation par contrainte et, d'autre part, comme l'illustration de l'intérêt de la méta-programmation pour modifier radicalement le fonctionnement du langage. Ce TP se décompose en trois parties :

- après une sensibilisation aux limitations de PROLOG (en particulier sur la manipulation délicate de l'arithmétique), on propose d'étendre l'interprète naturel de PROLOG pour qu'il prenne en compte des contraintes d'égalités (égalité d'une variable à un terme). Pour cela des prédicats de transformation, de test et d'évaluation de l'égalité sont réalisés. Un interprète avec continuation permet une intégration simple dans le langage. En conclusion de cette partie, on constate la possibilité d'un prédicat

factorielle/2 réversible.

- la deuxième partie conduit les élèves à prendre en compte des contraintes de différences.

- enfin, la troisième partie laissée à la libre appréciation des élèves introduit la gestion des contraintes sur les intervalles (entiers) en utilisant des variables à attributs.

- **Résolution de sorites** (3 séances)

Ce dernier TP consacré à la résolution de *sorites* [CAR 66] exprimées en langage naturel. Ce TP permet d'introduire quelques notions de traitement de langage naturel et de faire le lien avec la première partie du cours. Il est décrit en détail comme exemple de sujet de TP la section 5.

Ce TP sert en alternance avec le précédent d'évaluation pour le module.

## 5. Structure des sujets de TP

Nous présentons ici en détail le sujet consacré à la résolution de sorites pour montrer la structure générale des sujets de TP *applicatifs*. Chaque partie du TP est commentée par une présentation plus générale en italique.

### 5.1. Présentation du sujet

*Le contexte de chaque sujet est introduit en quelques paragraphes qui donnent des pointeurs ou des informations générales sur le sujet abordé : les systèmes experts, les extensions de la programmation logique, la méta-programmation. Ensuite, une présentation de l'organisation du TP est décrite.*

Le révérend Dodgson plus connu sous le pseudonyme de Lewis Carroll, écrivain et logicien britannique du siècle dernier (1832 – 1898) a proposé des raisonnements appelés *sorites* dont il faut trouver la conclusion. Voici un exemple <sup>10</sup> de ces raisonnements :

- Quand je résous un exercice de logique sans ronchonner, vous pouvez être sûr que c'en est un que je comprends ;

- Ces sorites ne sont pas disposés régulièrement, comme les exercices auxquels je suis habitué ;

- Aucun exercice facile ne me donne jamais mal à la tête ;

- Je ne comprends pas les exercices qui ne sont pas disposés régulièrement, comme ceux auxquels je suis habitué ;

- Je ne ronchonne jamais devant un exercice, à moins qu'il ne me donne mal à la tête ;

---

10. Ces explications proviennent de [L'H 98] et les sorites elles-mêmes proviennent de [CAR 66].

– Que peut-on en conclure ?

Le but du TP est de définir en PROLOG un certain nombre d'outils pour résoudre automatiquement ces sorites. À la fin du TP, on pourra simplement faire :

```
?- init,  
  premisses("les exercices qui sont resolus sans ronchonner sont ceux  
            que je comprends"),  
  premisses("cet exercice n'est pas dispose regulierement"),  
  premisses("aucun exercice qui est facile n'est migraineux"),  
  premisses("les exercices qui ne sont pas disposes regulierement ne  
            sont pas parmi ceux que je comprends"),  
  premisses("les exercices sont resolus sans ronchonner a moins qu'ils  
            ne soient migraineux"),  
  solution(X).  
X = 'cet exercice n'est pas facile'
```

La particularité des sorites de Lewis Carroll est qu'elles peuvent être résolues en utilisant le principe de résolution car chaque prémisses ne doit servir qu'une et une seule fois. On obtient la solution, en modélisant chaque prémisses sous la forme de clauses et partant d'une quelconque de ces prémisses, en appliquant le principe de résolution en combinaison avec les autres clauses.

Nous allons écrire notre outil en plusieurs étapes. Tout d'abord, nous allons nous attacher à la résolution symbolique de la sorite, *i.e.* en supposant les prémisses déjà modélisées. Puis nous allons nous occuper de la reconnaissance de la structure de chaque phrase afin de modéliser automatiquement chaque prémisses. Il s'agit en fait d'une sorte de traitement du langage naturel.

## 5.2. Outils de base

*La première partie de chacun des TP applicatifs est consacrée à la réalisation des outils de base qui servent pour réaliser l'application demandée.*

Ici, il s'agit simplement de demander la réalisation des outils nécessaires pour la résolution au sens logique de la sorite. On suppose à ce moment du sujet que les prémisses ont été modélisées et qu'elles sont accessibles à travers la base de faits PROLOG.

## 5.3. Nouvelles notions

*Une deuxième partie est ensuite consacrée à l'introduction de nouvelles notions. Ces nouvelles notions sont introduites par l'intermédiaire de prédicats à compléter ou de prédicats proches commentés qui nécessitent une modification substantielle.*

Ici, pour le traitement des expressions représentant les prémisses, un processus en trois temps est adopté :

- 1) Décomposition de la chaîne de caractères en une liste d'atomes (mots). Les séparateurs d'atomes (mots) seront les espaces (code 32).
- 2) Reconnaissance d'un *modèle* (pattern) dans la liste représentant la chaîne de caractères. À ce modèle correspond une règle de modélisation.
- 3) Enregistrement des propriétés : correspondance entre une série de mots et un prédicat au sens logique.

Les premiers prédicats demandés sont de simples outils de manipulations de chaînes de caractères. Pour reconnaître des *modèles* dans les phrases représentant les prémisses, on analyse chaque phrase pour reconnaître un certain nombre de *motifs* : *les A sont B, nul n'est A quand B, ...* Pour cela, on introduit un prédicat de filtrage avec joker (comme montré sur l'exemple 8) puis on demande aux élèves de réaliser un prédicat `filtrage/2` avec variables libres qui les unifie avec les atomes remplacés en cas de succès.

**Exemple 8 (Filtrage avec joker) :**

```
filtrage([], []).
filtrage([], [*]).           % une \'{e}toile remplace du vide !
filtrage([X|Xs], [?|Ys]) :- % un seul caract\ '{e}re
    filtrage(Xs, Ys).
filtrage([X|Xs], [*|Ys]) :- % premier cas pour * : remplace vide
    filtrage([X|Xs], Ys).
filtrage([X|Xs], [*|Ys]) :- % deuxieme cas, remplace au moins X
    filtrage(Xs, [*|Ys]).
filtrage([X|Xs], [X|Ys]) :-
    filtrage(Xs, Ys).
```

Enfin, un lien est réalisé entre ce qui a été identifié dans une phrase rentrée par l'utilisateur et sa représentation sous forme de fait PROLOG.

#### 5.4. Réalisation de l'outil

*La troisième partie est consacrée à la réalisation de l'outil. La plupart du temps il s'agit juste d'assembler les morceaux dans un prédicat générique. Souvent, la démarche générale est rappelée pour que l'écriture du prédicat soit la plus simple possible.*

Ici, quelques outils supplémentaires, dont l'écriture ne présente pas d'intérêt pédagogique, sont fournis pour réaliser le prédicat final : `solution/1`.

### 5.5. Ouverture

*Dans la mesure du possible, les sujets de TP se concluent sur une partie complètement ouverte où l'élève propose ses propres solutions. Il s'agit souvent d'étendre l'outil réalisé.*

Dans le cadre des sorites, plusieurs extensions sont proposées : la prise en compte d'un univers de calcul explicite (*i.e.* les prémisses de la forme : *les U qui sont A sont B*) ; une gestion automatisée des termes antonymes permettant de ne pas gérer explicitement la modélisation de cette information ; une gestion automatique des équivalences lorsqu'elles se présentent (l'outil présenté ici ne fonctionne correctement que lorsque les prémisses s'expriment sous forme d'implication).

## 6. Historique

Avant la réforme de 1998, le cours de Programmation Logique était un module obligatoire de quinze heures dont le responsable était Patrice Boizumault. Il n'était alors pas raisonnable de consacrer trop de temps aux liens avec la logique formelle et le cours se focalisait sur le langage lui-même. De même, le volume horaire consacré aux TP interdisait la possibilité de réaliser des sujets plus avancés et l'on se contentait à l'époque de trois TP (les deux premiers actuels et un supplémentaire d'application du cours).

Suite à la réforme des enseignements à l'école des Mines en 1998, le module a vu son volume horaire doubler permettant alors de proposer le cours tel qu'il est aujourd'hui. Les premières années, le cours allait plus loin sur le langage : nous appelions ceci du PROLOG *avancé*. Nous présentions les structures incomplètes (listes de différences, ...), les manipulations de la base de faits, les prédicats *toutes solutions* et leur implantation, l'indexation de clauses, ... Au fil du temps, il est apparu que cette partie du cours s'éloignait un peu trop des objectifs du cours. En particulier, nous avons pensé qu'il n'est pas utile de montrer que PROLOG peut permettre de réaliser des choses très complexes de manière très efficace et nous avons préféré nous focaliser sur la démarche particulière induite par la programmation logique.

## 7. Conclusion et évolutions futures

Les enseignements sont évalués par les élèves à l'École des Mines de Nantes. Ainsi, chaque année, le cours est bien apprécié. L'aspect ludique de la première partie du cours permet de faire passer sans (trop) de souffrances les concepts de la logique. Le langage PROLOG est apprécié pour lui-même : la réversibilité, l'aspect déclaratif, ... sont autant d'atouts qui séduisent rapidement les élèves qui se destinent aux options informatiques proposées par l'école des Mines. Enfin, l'aspect *applicatif* des TP permettent une mise en situation du langage qui ne peut que convenir à des élèves-

ingénieurs. Bien sûr, il reste toujours un ou deux réfractaires au mode de pensée PROLOG par promotion mais il reste difficile à enseigner en lui-même.

Il reste quelques défauts. Le passage de la logique formelle à PROLOG est un peu trop rapide mais il est difficile de consacrer beaucoup plus de temps à ce lien. L'arithmétique (et ses pièges) est délicate à faire passer et représente le premier éloignement de PROLOG à une certaine logique. C'est pourquoi les prochaines promotions n'auront plus l'occasion d'affronter ces problèmes car nous utiliserons de manière totalement transparente une couche *contrainte* au dessus de PROLOG<sup>11</sup>.

Enfin, il est plus que temps de proposer aux élèves un environnement de développement adéquat. Il est clair que l'utilisation actuelle du mode *ligne de commande* de SICSTUS est loin d'être satisfaisant.

*Les documents mentionnés dans cet article sont disponibles sur une page dédiée : [www.emn.fr/jussien/prolog/](http://www.emn.fr/jussien/prolog/).*

#### Remerciements

Un grand merci à Patrice Boizumault, bien sûr, Yvon L'Hospitalier, Romuald Debruyne, Yahia Lebbah et ... les élèves qui se sont succédés ! Ils ont tous contribué à l'évolution de ce cours.

#### 8. Bibliographie

- [CAR 66] CARROLL L., *Logique sans peine*, Hermann, 1966.
- [FRI 86] FRIANT J., L'HOSPITALIER Y., *Jeux-problèmes : de la logique à l'intelligence artificielle*, Les éditions d'organisation, 1986.
- [GIA 85] GIANNESINI F., KANOUI H., PASERO R., CANEGHEM M. V., *Prolog*, Informatique Intelligence Artificielle, InterEditions, 1985.
- [JUS 94] JUSSIEN N., VERGER V., L'HOSPITALIER Y., AUGEREAU B., GILLET P., « Le système expert NEREIS des annélides polychètes de France (ordre des Phyllodocida, Amphinomida, Spintherida et Eunicida) », *Canadian Journal of Zoology – Revue Canadienne de Zoologie*, vol. 72, n° 12, 1994, p. 2255–2260.
- [L'H 98] L'HOSPITALIER Y., *Énigmes et Jeux logiques*, Eyrolles, 1998.
- [O'K 92] O'KEEFE R., *The Craft of Prolog*, Advanced Programming Techniques, MIT Press, 1992.
- [STE 86] STERLING L., SHAPIRO E., *The Art of Prolog*, Advanced Programming Techniques, MIT Press, 1986.

---

11. Cela facilitera d'autant le passage à la programmation par contraintes...