Prolog, 50 ans d'histoire de l'intelligence artificielle

Laurent Cervoni et Julien Brasseur [1].

À l'occasion des 50 ans de la naissance du langage Prolog, nous proposons, dans cet article, d'évoquer certains des aspects qui, selon nous, en font un langage atypique, et nous discuterons de son apport dans les problèmes d'intelligence artificielle. Pour ce faire, quelques exemples concrets seront proposés et discutés.

Introduction : une brève histoire de Prolog

Le langage Prolog a été présenté pour la première fois à Marseille en 1972 par Alain Colmerauer et son équipe [3]. Prolog, contraction de programmation en logique, est un langage permettant de modéliser le raisonnement en s'appuyant sur la logique d'ordre 1. Il est l'aboutissement de réflexions commencées quelques années auparavant, notamment sur les systèmes-Q, qui sont des mécanismes de règles de transformations de graphes orientés, utilisés, par exemple, pour l'analyse syntaxique de phrases. On peut d'ailleurs noter qu'un des points de départ de ces travaux de recherche était la traduction (français-anglais) et le traitement du langage (analyse et génération de phrases, sur la base de la définition de leur structure grammaticale).

Ce travail, enrichi par les recherches de [7] de l'université d'Edimbourg (puis par les travaux de David H. D. Warren et bien d'autres), a jeté les bases de la programmation logique telle qu'elle existe encore aujourd'hui. La syntaxe des premiers Prolog a alors évolué vers la syntaxe dite d'Edimbourg puis a été normalisé par l'ISO [4]. Dans les années 1980, plusieurs variantes ont vu le jour et d'autres technologies

1. Directeur et chercheurs du Centre de recherche et innovation Talan

DOI:10.48556/SIF.1024.20.55 **1024** – Bulletin de la SiF, numéro 20, Novembre 2022, pp. 55–63

connexes ou dérivées (comme la programmation par contraintes) sont alors devenus des outils clés des approches symboliques de l'intelligence artificielle.

D'après l'index Tiobe (cf. table 1), 50 ans après sa création, Prolog reste toujours dans la vingtaine des langages utilisés (il était 3^{e} dans les années 1980) et semble même connaître un sensible regain d'intérêt. Son cinquantenaire pourrait être l'occasion de renforcer cette tendance, l'apport de la programmation logique étant particulièrement intéressant en spécifications de modèles, en raisonnement basé sur des règles et en résolution de problèmes avec contraintes.

La communauté Prolog est toujours active et le langage est proposé dans des versions modernisées comme Scryer Prolog (développé principalement en RUST), incluant CPL(B) et CLP(Z), Problog qui incorpore des éléments probabilistes, ou inspire de nouveaux outils tel Logica [2], conçu par Google.

Langage	Juin 2022	2017	2012	2007	2002	1997	1992	1987
Python	1	5	8	7	12	28	-	-
C	2	2	2	2	2	1	1	1
Java	3	1	1	1	1	13	-	-
C++	4	3	3	3	3	2	2	5
C#	5	4	4	8	18	-	-	-
Visual Basic	6	15	-	-	-	-	-	-
JavaScript	7	7	10	9	9	20	-	-
SQL	8	10	-	-	-	-	-	-
Assembleur	9	-	-	-	7	-	-	-
Swift	10	8	6	5	6	-	-	-
Prolog	**20**	**33**	**37**	**27**	**16**	**19**	**15**	**3**
Lisp	32	31	13	15	13	10	6	2
Pascal	270	105	16	20	99	9	3	6

TABLE 1: Extrait de l'index Tiobe répertoriant les langages de programmation les plus populaires (`https://www.tiobe.com/tiobe-index`, juin 2022).

La galaxie des outils et langages autour de la programmation logique (cf.figure 1) s'est enrichie au fil des années, Alain Colmerauer ayant lui-même contribué à la programmation par contraintes dans Prolog IV [2].

2. `https://logica.dev`.

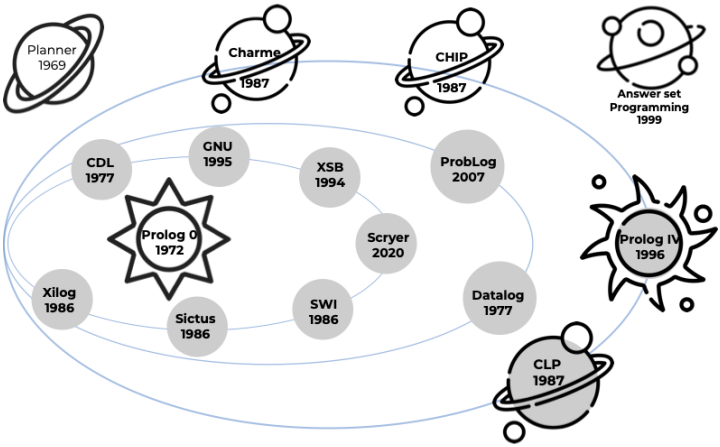


FIGURE 1. Représentation partielle et subjective d'une histoire de la galaxie « programmation logique ».

Le fonctionnement de Prolog et quelques exemples

Dans les langages de programmation « traditionnels » (impératifs), l'investissement pour représenter le raisonnement ou l'expression « humaine » est important puisqu'il faut pratiquement résoudre le problème en le programmant. En Prolog, on reste suffisamment proche de la logique mathématique pour que toute une série de problèmes puissent s'exprimer de façon assez proche de leur simple description.

Prolog utilise ainsi la logique du premier ordre pour permettre la description d'un « univers » en :

— établissant des faits sur des objets ;
— définissant des règles ou des relations sur ces objets ;
— interrogeant le système sur ces objets et ces relations.

Prolog peut aussi, dynamiquement, enrichir sa base de faits et de règles en fonction d'informations nouvelles ce qui rend le langage totalement différent de Python ou d'un autre langage plus courant.

En Prolog, les variables se définissent au sens des variables mathématiques que l'on trouve dans les expressions du type « Quel que soit X tel que... ». Elles ne sont pas typées au sens informatique usuel : une variable Prolog pourra s'unifier avec un entier, une chaîne, une liste ou tout autre objet Prolog. Par convention, toute variable Prolog débute par une majuscule. Ayant établi des relations et un univers, il est évidemment légitime de vouloir le questionner.

Considérons, à titre d'exemple, le programme Prolog de la table 2. Si on pose la question en Prolog de savoir si Socrate est humain (« `?- humain(socrate).` »),

Tout humain est mortel (pour tout X humain, X est mortel)	`mortel(X) :- humain(X).`
Socrate est humain	`humain(socrate).`
Mon voisin est humain	`humain(monvoisin).`

TABLE 2. Expression en français (à gauche) et écriture en Prolog (à droite).

alors la réponse sera logiquement « `true` ». De manière similaire, à la question « `?- humain(Qui).` », Prolog, qui est un langage non-déterministe, propose toutes les réponses valides à une question donnée et retournera donc « `Qui = socrate`, `Qui = monvoisin` ». Cependant, à la question « `?- humain(laurent).` », Prolog retournera « `false` », ce qui est logique puisque cette information ne figure pas dans son domaine de connaissance. On peut évidemment compléter ce « monde » Prolog en l'enrichissant de nouveaux faits et règles (cf.table 3).

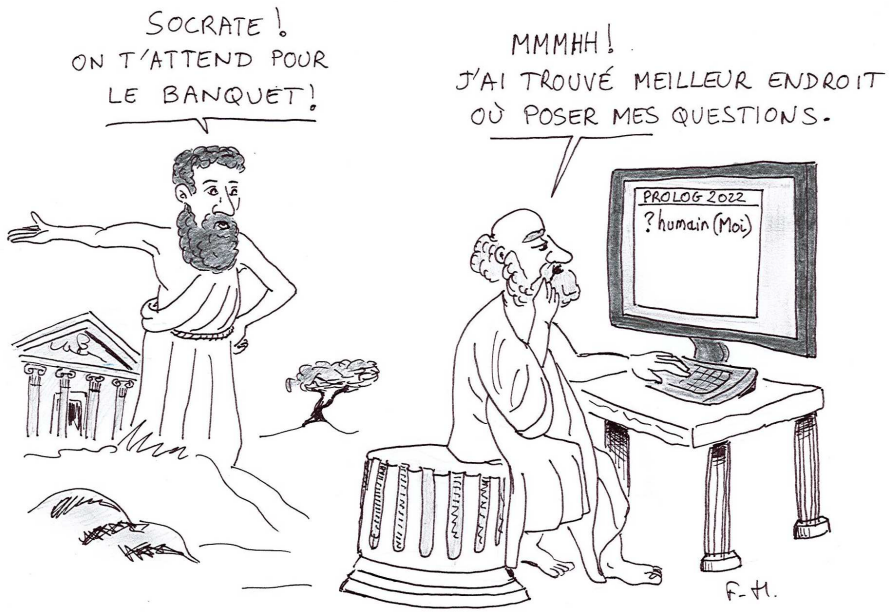
```
mortel(X) :- humain(X).
meurt(X) :- mortel(X), empoisonne(X).
empoisonne(X) :- boit(X,Y), poison(Y).
humain(socrate).
humain(monvoisin).
poison(cigue).
boit(socrate,cigue).
```

TABLE 3. Description plus complexe d'un monde.

Remarquez qu'il est possible de tester les éléments de programme proposés dans cet article dans SWI-Prolog, par exemple dans sa version en ligne [3]. Ces quelques exemples montrent la spécificité de Prolog, qui est donc un langage de programmation atypique. En effet, un programme Prolog est constitué de « paquets » de clauses de Horn. Une clause de Horn est un cas particulier de clause, composée d'une disjonction logique de littéraux, où au maximum un des littéraux est positif, et tous les autres, clos ou non clos (i.e. avec ou sans variable au sens logique), sont négatifs. La tête de la clause est le littéral positif, le corps de la clause contient les autres éléments.

Une clause de Horn « sans corps », c'est-à-dire avec un seul littéral positif est un fait. Un paquet contient les clauses ayant le même foncteur de prédicat (i.e. le même nom de clause). Une clause de Horn peut donc s'écrire sous la forme $(a \wedge b \wedge c \wedge \ldots) \Rightarrow$ Tête. L'exécution d'un programme en Prolog revient à vérifier si un but (ou une requête) est vrai au regard de l'ensemble des clauses contenues dans

3. `https://swish.swi-prolog.org`.



ce programme. Deux mécanismes essentiels sont mis en œuvre lors d'une requête Prolog : l'unification et le principe de résolution.

Unification Le principe de l'unification est le suivant : deux termes t_1 et t_2 sont dits unifiables s'il existe s, une substitution des variables, telle que $s(t_1) = s(t_2)$. Par exemple, si $t_1 = f(X,a)$ et $t_2 = f(b,Y)$, alors t_1 et t_2 sont unifiables pour $X = b$ et $Y = a$. En Prolog, l'algorithme de Herbrand calcule l'unificateur le plus général (*most general unifier* ou *mgu*, i.e. celui qui unifie le moins les variables). Si t_1 et t_2 sont unifiables, l'algorithme calcule le *mgu* des deux termes, sinon il signale un échec : en Prolog standard, il est convenu que t et X sont unifiables si t ne contient pas X.

Résolution Décrivons maintenant l'algorithme de résolution. On appelle « but » une clause de Horn sans littéral positif et « hypothèses » les autres. Pour résoudre un système de clauses de Horn par le principe de résolution, il est nécessaire d'avoir (1) un but (T) et (2) un ensemble d'hypopthèses (C_i).

Supposons que la clause T doive être prouvée à partir des clauses $C_1, \ldots, C_i, \ldots, C_n$, le fonctionnement du processus de résolution peut schématiquement se résumer en quatre étapes :

1. ajouter $\neg T$ à l'ensemble des C_i ;
2. calculer le résolvant de deux clauses ;

3. si le résolvant est vide (antilogie), la preuve est faite ;
4. sinon, l'ajouter à l'ensemble et retourner à l'étape 2.

Il est à noter que la stratégie de Prolog est dite en profondeur d'abord. Elle peut se perdre dans des branches infinies. L'ordre des éléments dans le but et l'ordre des clauses dans les paquets détermine la construction de l'arbre de recherche. L'utilisation de certains prédicats (comme l'opération `cut` qui supprime certaines branches de l'arbre des clauses à parcourir) viennent modifier la stratégie de parcours et réduire la lisibilité d'un programme.

Prolog aujourd'hui — Le cas Recov'Up

Ce panorama sur Prolog ne serait pas complet sans mention de ses applications les plus récentes, dont une liste non exhaustive peut être trouvée dans [6]. Considérons, à ce titre, le cas de l'application Recov'Up, application récente basée sur Prolog.

Recov'up est une application web qui prévient et aide les patients à la rééducation des troubles musculosquelettiques comme l'entorse de cheville, la lombalgie, la cervicalgie, etc. Cette application élabore une liste d'exercices en fonction des symptômes du patient.

La génération des exercices est obtenue par un algorithme d'intelligence artificielle symbolique, développé en Prolog, en collaboration avec les équipes du centre de recherche de Talan et de la startup Skeewai (éditrice de Recov'up). Cette approche symbolique s'appuie principalement sur des règles de rééducation exprimées par les médecins et kinésithérapeutes, spécifiant les exercices pertinents pour le patient en fonction de sa pathologie et son état physique. Le mécanisme mis en œuvre ne nécessite pas un apprentissage sur un important volume de données (ce qui est un avantage dans le secteur de la santé où les règles d'utilisation des données des patients sont strictement encadrées) mais intègre les données spécifiques de chaque patient et son évolution face aux séances qui lui sont proposées.

La génération d'exercices de rééducation s'appuie d'une part, sur un cœur en logique d'ordre 1 « standard » (la description des règles médicales et de kinésithérapie qui permettent de déterminer quels sont les exercices qui correspondent à la situation d'un patient) et, d'autre part, sur la capacité de Prolog à exécuter dynamiquement des prédicats et à enrichir sa base de clauses (via les prédicats d'ordre 2 que sont `findall`, `bagof` et `assert`).

Prolog et l'intelligence artificielle symbolique

L'intelligence artificielle peut être scindée en deux grands courants : l'intelligence artificielle symbolique et l'intelligence artificielle « connexionniste » (ou « numérique »). Cette dernière connaît, depuis 2012 environ, une popularité et une croissance vertigineuse due, entre autres, au développement des technologiques de l'information et à l'acquisition des larges quantités de données qu'elles ont permises.

Toutefois, l'IA numérique peut exiger, pour produire des résultats probants, des quantités de données parfois tellement importantes qu'elles nécessitent, pour être traitées (particulièrement en phase d'apprentissage), une puissance de calcul conséquente, ce qui peut être rédhibitoire, notamment pour les entreprises. En dehors des problèmes pratiques que cela pose, il est à noter que la quantité d'opérations qui doivent être effectuées peut avoir un impact carbone [13] ou une consommation énergétique qu'il est aujourd'hui nécessaire de limiter.

Il existe aussi de nombreux cas pratiques où les données peuvent être indisponibles, impossibles à acquérir ou ne pas pouvoir être conservées suffisamment longtemps (respect du RGPD, volumétrie...). Le cas se pose classiquement lorsque l'on souhaite mettre en place une IA statistique dans un cadre médical, où les données sont sensibles par nature et protégées par un cadre légal strict dès que des données personnelles sont en jeu.

L'IA symbolique et, plus particulièrement Prolog, peut ici être un atout majeur venant compléter la puissance de l'IA numérique. Le cas du projet Recov'Up, cité plus haut, en est un exemple : une pathologie est soignée quotidiennement par les médecins mais il n'existe pas de larges bases de données décrivant l'évolution du patient en fonction des différents soins pratiqués et des profils de patients.

Prolog présente également un intérêt pour les cas où il est important que l'IA soit explicable. Dans certains cas d'applications pratiques, civiles (voitures autonomes, banques, médecine, etc) comme militaires (drones, etc), cela peut devenir critique. En effet, les résultats donnés par Prolog sont, par nature, explicables : « coder » en Prolog revient à décrire un univers, aussi l'explication est-elle *contenue* dans la modélisation Prolog. On peut, en outre, interroger Prolog sur les unifications de faits réalisées : Prolog les parcourt en profondeur et peut expliquer son raisonnement via le système inductif qu'il utilise.

C'est ainsi, d'ailleurs, que Prolog a été utilisé pour concevoir des automates de détections de bugs dans le code de Tensorflow [12] ou encore pour réaliser des démonstrateurs de théorèmes, comme Muscadet [10] ou ARGOS [14]. Plus récemment, [11] ont montré que Prolog peut être utilisé pour représenter des théories scientifiques et permettre de nouvelles inférences (cf.table 4).

On voit bien là l'intérêt, sinon d'adopter des approches proposées par l'IA symbolique (lorsque cela est possible et pertinent), ou au moins de considérer des solutions hybrides, dites « neuro-symboliques ».

En ce sens, le langage Prolog, qui a été pensé dès sa conception comme un outil de l'intelligence artificielle symbolique, a toute sa place. On peut aussi noter que Prolog s'interface très facilement avec d'autres langages comme Python, actuellement très prisé par la communauté IA, et peut donc servir à la réalisation « d'IA neuro-symboliques ».

```
/* Si la dérivée de U par rapport à X est DU et dérivée
de V est DV, la dérivée de U + V est alors DU + DV, de
U−V est DU−DV, de U∗V est DU∗V+U∗DV, etc... */
d(U+V,X,DU+DV) :- d(U,X,DU), d(V,X,DV).
d(U-V,X,DU-DV) :- d(U,X,DU), d(V,X,DV).
d(U*V,X,DU*V+U*DV) :- d(U,X,DU), d(V,X,DV).
d(U/V,X,(DU*V-U*DV)/V^2) :- d(U,X,DU), d(V,X,DV).
```

TABLE 4. Comment formaliser les règles de dérivation de fonctions mathématiques pour, ensuite, demander à Prolog d'en déduire la dérivée d'une fonction.

Conclusion

Les principes mis en œuvre dans Prolog — unification, SLD Résolution (i.e. *Selective Linear Definite clause resolution* [8], logique du premier ordre, gestion éventuelle de contraintes sur des domaines — en font un langage original. Il est ainsi particulièrement bien adapté pour la résolution de problèmes qui peuvent s'exprimer sous forme de règles, de bases de données ou de faits.

En outre, il permet la modélisation d'une forme de raisonnement avec une économie d'efforts qui le rend séduisant. Il est simple à apprendre et à mettre en œuvre. Ses concepts et les instructions à maîtriser sont réduits ; en premier langage d'apprentissage, il présente de nombreux avantages, notamment du fait qu'il ne nécessite pas d'imaginer d'algorithme pour résoudre un problème. De même, l'apprentissage de la récursivité est « historiquement » intuitif en Prolog (voir par exemple [5]). Paradoxalement, il devient plus difficile à utiliser pour un programmeur formé aux boucles, aux structures typées et ayant l'habitude de concevoir un algorithme plutôt que de décrire le problème posé.

Prolog peut s'articuler aisément avec d'autres mécanismes d'IA, et en particulier les processus d'apprentissage de type connexionniste. Ainsi, Prolog peut générer facilement des contenus syntaxiquement exacts à partir de grammaires avec une économie d'énergie incomparable. En revanche, les aspects sémantiques ou qui relèvent de la perception (images, sons, etc) y sont plus complexes à représenter.

Cette combinaison contribue à la construction de solutions « intelligentes ». Cette approche hybride nous paraît représenter certains fonctionnements du raisonnement humain et elle ne s'oppose pas aux techniques « statistiques », elle les enrichit.

Les extensions de la programmation logique et ses dérivées ont démontré leur efficacité comme, par exemple, dans NukkAI avec la programmation logique probabiliste [9], dans BitCoinLog ou encore la vérification de protocole cryptographique [1].

Pour toutes ces raisons, il nous semble que les spécificités de Prolog (et de ses descendants comme les langages de programmation par contraintes, par exemple) esquissées ici sont un atout pour de nombreux problèmes nécessitant un cheminement explicable, pouvant être décrit sous forme de règles ou de relations entre entités, où le raisonnement vient compléter des processus d'apprentissage automatique, ou bien encore quand les données sont en volume réduit.

Références

[1] B. Blanchet et al. An efficient cryptographic protocol verifier based on prolog rules. In *csfw*, volume 1, pages 82–96, 2001.

[2] A. Colmerauer. Les bases de Prolog IV. *Le Manuel de Prolog IV, PrologIA*, 1996.

[3] A. Colmerauer, H. Kanoui, R. Pasero, and P. Roussel. Un système de communication en français, rapport préliminaire de fin de contrat IRIA. *Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France*, 1972.

[4] P. Deransart, A. Ed-Dbali, and L. Cervoni. *Prolog : the standard : reference manual*. Springer Science & Business Media, 1996.

[5] B. S. Elenbogen and M. R. O'Kennon. Teaching recursion using fractals in Prolog. In *Proceedings of the nineteenth SIGCSE technical symposium on Computer science education*, pages 263–266, 1988.

[6] L. Gouzènes. New Prolog Application Census 2022, 2022.

[7] R. Kowalski. Predicate logic as programming language. In *IFIP congress*, volume 74, pages 569–544, 1974.

[8] R. Kowalski and D. Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2(3-4) :227–260, 1971.

[9] J. Li, S. Thepaut, and V. Ventos. StarAI : Reducing incompleteness in the game of Bridge using PLP. *arXiv preprint arXiv :2001.08193*, 2020.

[10] D. Pastre. MUSCADET : An automatic theorem proving system using knowledge and metaknowledge in mathematics. *Artificial Intelligence*, 38(3) :257–318, 1989.

[11] J.-C. Rohner and H. Kjellerstrand. Using logic programming for theory representation and scientific inference. *New Ideas in Psychology*, 61 :100838, 2021.

[12] R. Schumi and J. Sun. ExAIS : Executable AI Semantics. *arXiv preprint arXiv :2202.09868*, 2022.

[13] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni. Green AI. *Communications of the ACM*, 63(12) :54–63, 2020.

[14] J.-P. Spagnol. ARGOS, un demonstrateur de theoremes en geometrie. *Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation*, 8(1) :113–125, 2001.