

# 50 ans de programmation logique



Fred Mesnard

LIM, université de La Réunion

Juin 2023

## Résumé :

Nous profitons de l'anniversaire des 50 ans de Prolog pour exposer brièvement une vision personnelle des 50 ans de la programmation logique.

Au passage, nous rectifions quelques idées reçues.

Puis nous plaçons nos travaux dans ce cadre, en esquissant les directions que prennent nos recherches actuelles.

# La programmation logique $\neq$ Prolog

- ▶ La programmation logique est une façon de programmer :
  - ▶ on décrit le problème sous forme d'assertions logiques
  - ▶ on questionne cette description
  - ▶ le système répond
- ▶ Prolog est le langage de programmation basé sur cette idée le plus utilisé

# Algorithmes et calculs, logiques et preuves

Deux points de vue possibles :

- ▶ des preuves, on extrait des algorithmes
  - ▶ Une correspondance : Curry 58, Howard 69, ...
  - ▶ Coq : Huet & Coquand 84, Paulin 89, ...
  - ▶ Le théorème des 4 couleurs, Gonthier Werner 07
  - ▶ Le compilateur CompCert, Leroy 08
- ▶ des preuves, on extrait des calculs
  - ▶ Résolution, Robinson 65
  - ▶ Résolution SLD (*Selective Linear Definite clause resolution*), Kowalski & Kuehner 71
  - ▶ Prolog, Colmerauer & Roussel 72
  - ▶ *Predicate Logic as a Programming Language*, Kowalski 73

# Extraction d'algorithmes : un exemple

On se place dans  $(\mathbb{N}, \{0, 1\}, \{+/2\}, \{= /2, \leq /2\})$ , montrons :

$$\forall x \forall y [y \leq x \rightarrow \exists z | x = y + z]$$

Soit  $(x, y)$  un couple d'entiers naturels

- ▶ Si  $y > x$ , l'implication est vraie
- ▶ Sinon  $z = x - y$  convient

Mais la soustraction n'existe pas dans  $(\mathbb{N}, \{0, 1\}, \{+/2\}, \{= /2, \leq /2\})$  !

# Extraction d'algorithmes : un exemple

Montrons  $\forall x \forall y [y \leq x \rightarrow \exists z | x = y + z]$

Induction sur  $x$  :

- ▶ Pour  $x=0$ , montrons  $\forall y [y \leq 0 \rightarrow \exists z | 0 = y + z]$   
Distinguons 2 cas pour  $y$  :
  - ▶  $y = 0$  :  $z = 0$  convient
  - ▶  $y > 0$  : l'implication est trivialement vraie
- ▶ Supposons la propriété vraie pour un certain  $x$  :  
 $HR : \forall y [y \leq x \rightarrow \exists z | x = y + z]$   
Montrons qu'elle reste vraie pour  $x' = x + 1$ , i.e.,  
 $\forall y [y \leq x' \rightarrow \exists z | x' = y + z]$ .  
Distinguons 3 cas pour  $y$  :
  - ▶  $y < x + 1$  : en appliquant  $HR$ ,  $z' = z + 1$  convient
  - ▶  $y = x + 1$  :  $z = 0$  convient
  - ▶  $y > x + 1$  : l'implication est trivialement vraie

## Extraction d'algorithmes : un exemple

On a donc bien  $\forall x \forall y [y \leq x \rightarrow \exists z | x = y + z]$

Comment exprimer  $z$  en fonction de  $x$  et  $y$  ?

En *synthétisant* la soustraction à partir de la démonstration précédente !

# Extraction d'algorithmes : un exemple

Coq :

```
1 Require Import Lia.
2
3 Lemma minus:  $\forall x y : \mathbb{N}, y \leq x \rightarrow \{z \mid x=y+z\}$ .
4 Proof.
5   intro. induction x.
6   - intros. exists 0. lia.
7   - intros. destruct y.
8     + exists (S x). lia.
9     + destruct (IHx y).
10       * lia.
11       * exists x0. lia.
12 Defined.
13
14 Require Extraction.
15
16 Extraction minus.
17
18 (** val minus : nat -> nat -> nat **)
19 [(
20 let rec minus x y =
21   match x with
22   | 0 -> 0
23   | S x0 ->
24     (match y with
25     | 0 -> S x0
26     | S y0 -> minus x0 y0)
27 *)]
```

# Extraction de calculs : un exemple

Même problème, cette fois en programmation logique

Représentons les entiers par les termes  $\{0, s(0), s(s(0)), \dots\}$  et modélisons l'addition :

$$\begin{array}{l} \forall y \quad [\top \rightarrow \text{add}(0, y, y)] \\ \forall x \ y \ z \quad [\text{add}(x, y, z) \rightarrow \text{add}(s(x), y, s(z))] \end{array}$$

Idem, syntaxe Prolog :

```
1 add(0,Y,Y).  
2 add(s(X),Y,s(Z)) :- add(X,Y,Z).
```

# Extraction de calculs : un exemple

```
1 add(0,Y,Y).  
2 add(s(X),Y,s(Z)) :- add(X,Y,Z).
```

- ▶ chaque implication est correcte : je lis et je suis ok  
⇒ le programme est correct

*La correction des programmes est triviale !*

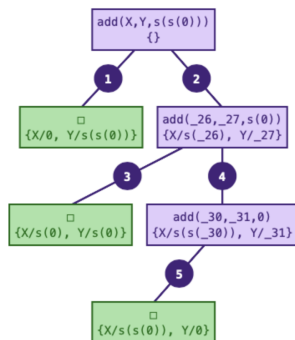
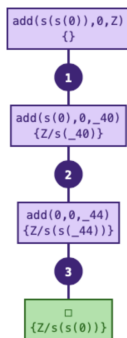
- ▶ par un calcul ascendant de la définition inductive :  
 $[ \text{add}(x, y, z) ] = \{ (s^n(0), y, s^n(y)) \mid n \in \mathbb{N}, y \text{ terme} \}$   
⇒ le programme est complet  
Quelques approches : Staerk 98, Drabent 22

# Extraction de calculs : un exemple

Via la résolution SLD, on extrait les valeurs des preuves :

$\exists ?Z \text{ add}(s(s(0)), 0, Z)$

$\exists ?X Y \text{ add}(X, Y, s(s(0)))$



# Extraction de calculs : en résumé

La programmation logique, c'est :

- ▶ spécifier en clauses de Horn définies — *quoi*
- ▶ prouver procéduralement des requêtes (conjonctions d'atomes existentiellement quantifiés) — *comment*
- ▶ extraire les valeurs des variables (toutes  $\exists$  quantifiées)

avec comme résultat central (van Emden & Kowalski 76) :

$$M_P = \text{lfp}(T_P) = SS(P)$$

La programmation logique est une instance du paradigme de programmation *déclarative* : l'utilisateur se concentre sur le *quoi*, la machine gère le *comment*

## Retour à notre exemple initial

Dans  $\mathbb{N}$ , on a :  $\forall x \forall y [y \leq x \rightarrow \exists z | x = y + z]$

Définitions Prolog :

```
leq(0,X).
```

```
leq(s(Y),s(X)) :- leq(Y,X).
```

```
add(0,X,X).
```

```
add(s(Y),Z,s(X)) :- add(Y,Z,X).
```

## Retour à notre exemple initial

Dans  $\mathbb{N}$ , on a :  $\forall x \forall y [y \leq x \rightarrow \exists z | x = y + z]$

Cette formule ne peut pas se réécrire comme une conjonction existentielle : elle n'est pas directement traitable par Prolog

Mais on constate la propriété pour  $X = s(s(0))$  et  $Y = s(0)$  :

```
[?- X=s(s(0)), Y=s(0), leq(Y,X).  
X = s(s(0)),  
Y = s(0).  
  
[?- X=s(s(0)), Y=s(0), leq(Y,X),add(Y,Z,X).  
X = s(s(0)),  
Y = Z, Z = s(0).  
  
?- ]
```

NB : la fonction qui calcule  $z$  pour  $x$  et  $y$  donnés est *implicite* dans la définition *relationnelle* de  $add(y, z, x)$ , il n'y a pas lieu de la synthétiser.

## Retour à notre exemple initial via LPTP

Dans  $\mathbb{N}$ , on a :  $\forall x \forall y [y \leq x \rightarrow \exists z | x = y + z]$

On constate via Prolog la propriété pour  $X = s(s(0))$  :

```
[?- X = s(s(0)), leq(Y,X).
```

```
X = s(s(0)),
```

```
Y = 0 ;
```

```
X = s(s(0)),
```

```
Y = s(0) ;
```

```
X = Y, Y = s(s(0)) ;
```

```
false.
```

```
[?- X = s(s(0)), leq(Y,X), add(Y,Z,X).
```

```
X = Z, Z = s(s(0)),
```

```
Y = 0 ;
```

```
X = s(s(0)),
```

```
Y = Z, Z = s(0) ;
```

```
X = Y, Y = s(s(0)),
```

```
Z = 0 ;
```

```
false.
```

## Retour à notre exemple initial

Dans  $\mathbb{N}$ , on a :

$$\forall x \forall y [y \leq x \rightarrow \exists z | x = y + z]$$

*i.e.*

$$\forall y \forall x [y \leq x \rightarrow \exists z | x = y + z]$$

On constate via Prolog la propriété pour  $Y = s(0)$  :

```
[?- Y = s(0), leq(Y,X).  
Y = s(0),  
X = s(_).  
  
[?- Y = s(0), leq(Y,X), add(Y,Z,X).  
Y = s(0),  
X = s(Z).  
  
?- ]
```

## Retour à notre exemple initial

Dans  $\mathbb{N}$ , on a :  $\forall x \forall y [y \leq x \rightarrow \exists z | x = y + z]$

Preuve en LPTP :

**Lemma 1** [*leq\_add*]  $\forall x, y (\mathbf{S} \mathbf{leq}(y, x) \rightarrow \exists z \mathbf{S} \mathbf{add}(y, z, x)).$

**Proof.**

Induction<sub>0</sub>:  $\forall y, x (\mathbf{S} \mathbf{leq}(y, x) \rightarrow \exists z \mathbf{S} \mathbf{add}(y, z, x)).$

Hypothesis<sub>1</sub>: none.  $\mathbf{S} \mathbf{add}(0, y, y). \exists z \mathbf{S} \mathbf{add}(0, z, y).$

Conclusion<sub>1</sub>:  $\exists z \mathbf{S} \mathbf{add}(0, z, y).$

Hypothesis<sub>1</sub>:  $\exists z \mathbf{S} \mathbf{add}(x, z, y)$  and  $\mathbf{S} \mathbf{leq}(x, y).$

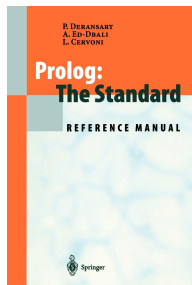
Let<sub>2</sub>  $z$  with  $\mathbf{S} \mathbf{add}(x, z, y). \mathbf{S} \mathbf{add}(s(x), z, s(y)). \exists z \mathbf{S} \mathbf{add}(s(x), z, s(y)).$

Thus<sub>2</sub>:  $\exists z \mathbf{S} \mathbf{add}(s(x), z, s(y)).$

Conclusion<sub>1</sub>:  $\exists z \mathbf{S} \mathbf{add}(s(x), z, s(y)). \quad \square$

# Idées reçues : autant de Prolog que d'implantations

- ▶ Nombreuses implantations stables et maintenues
  - ▶ Ciao-Prolog, Hermenegildo et al. 91
  - ▶ GNU-Prolog, Diaz 00
  - ▶ SICStus Prolog, Carlsson et al. 12
  - ▶ SWI-Prolog, Wielemaker et al. 12
  - ▶ XSB Prolog, Warren et al. 92
- ▶ Récemment : Ichiban/Prolog (Go), Tau (JS), Trealla (C/WASM), Stryer (Rust)
- ▶ ISO-Prolog, 95 – actuellement : Neumerkel



## Idées reçues : absence du test d'occurrence

- ▶ Test d'occurrence : l'équation  $x = T[x]$  n'a pas de solution dans les termes finis
- ▶ Historiquement omis dans Prolog pour des raisons d'efficacité
- ▶ En ISO-Prolog :  
`unify_with_occurs_check/2`
- ▶ En SWI-Prolog :  
`:- set_prolog_flag(occurs_check, true).`

# Idées reçues : arithmétique impérative

- ▶ `X is Y+Z` pour `X := Y+Z`
- ▶ son évaluation nécessite `Y` et `Z` connus
- ▶ sinon exception  
`ERROR: Arguments are not sufficiently instantiated`
- ▶ solution : les contraintes : `CLP(X)`, Jaffar & Lassez 87
- ▶ la contrainte `X=Y+Z` exprime une *relation* entre `X`, `Y` et `Z`

# Idées reçues : c'est lent

- ▶ LIPS : logical inferences per second
- ▶ quelques centaines mi 70
- ▶ SICStus Prolog : compilation JIT  
300 MLIPS sur Intel Core i7 3.60GHz  
(quelques centaines d'€)



- ▶ en 50 ans, un facteur  $10^6$

# Idées reçues : fermeture transitive

Définir la fermeture transitive de la relation binaire `arc/2` :

- ▶ *impossible* en logique du premier ordre !
- ▶ en Prolog, par exemple :  

```
tc_a(X, Y) :- arc(X, Y).  
tc_a(X, Z) :- arc(X, Y), tc_a(Y, Z).
```
- ▶ OK si `arc/2` acyclique mais boucle sinon
- ▶ solutions :
  - ▶ *modifier le code* en tenant à jour la liste des nœuds visités
  - ▶ ne pas modifier le code mais *modifier la stratégie* !  

```
:- table tc_a/2.
```
  - ▶ stratégie hybride descendante/ascendante

## Fait marquants et avancées significatives :

- ▶ La WAM, Warren 83
- ▶ Le projet *Fifth Generation Computer Systems*, 82-92
- ▶ CLP(X), Jaffar et Lassez 87  
Maher, Marriott, Naish, Stuckey
- ▶ *Abstract interpretation and application to logic programs*,  
Cousot et Cousot 92, Levi et al., Hermenegildo

## Fait marquants et avancées significatives :

- ▶ *The execution algorithm of Mercury, an efficient purely declarative LP language*, Somogyi et al. 96
  - ▶ complètement déclaratif
  - ▶ typé, modé, déterminisme, ordre supérieur
  - ▶ *programming in the large*, compilation vers C
- ▶ *Curry: A Truly Integrated Functional Logic Programming Language*, Hanus 97
- ▶ *The theoretical foundations of LPTP*, Stärk 98
- ▶ La négation : bien-fondée, stable, constructive  
Bien fondée = LPO + définitions inductives, Denecker 14

# Recherches en cours au niveau international

Des exemples récents :

- ▶ *Constrained Horn Clauses* ou CHC, inclus dans Z3  
Bjørner et al. 15
- ▶ *Soufflé*, basé sur Datalog et compilé en C++  
Scholz 16
- ▶ *Logica*, Google 21  
*Logic Programming Language for Data Analysis*  
Extension de Datalog, compile en SQL sur BigQuery
- ▶ *Verse* : un langage logico-fonctionnel d'Epic Games,  
Peyton-Jones et al. 22

# Recherches en cours au niveau international

Des exemples récents orientés IA symbolique :

- ▶ *answer set programming*, représentation des connaissances, pré-processeur SAT  
Marek, Niemelä, Truszczuński 99
- ▶ *s(CASP)*, version orientée par les requêtes  
Automating Commonsense Reasoning with ASP and *s(CASP)*, Gupta et al. 22

# Recherches en cours au niveau international

- ▶ *Logical English* – Kowalski 20

- ▶ un langage de programmation *lisible* par les non-informaticiens (~ Prolog édulcoré)  
mais plus difficile à programmer efficacement

## Applications :

- ▶ *Computational Law*

The British Nationality Act as a Logic Program  
Kowalski et al. 86

- ▶ *Explainable AI* – Pourquoi ?

- ▶ *Smart Contract* – Blockchain

- ▶ *Rules as Code* – Réglementations et textes administratifs

- ▶ directement compréhensible par les non-informaticiens
  - ▶ sous une forme informatiquement exécutable  
tester, raisonner, appliquer les techniques de génie logiciel

# Recherches en cours au niveau international

Un exemple orienté IA neuro-symbolique :  
ChatGPT est un agent conversationnel basé sur le modèle de langage GPT-3 (Generative Pre-training Transformer v3) développé par OpenAI.

FR

que sais-tu de Mahé de La Bourdonnais ?



Mahé de La Bourdonnais, de son nom complet Bertrand-François Mahé de La Bourdonnais, était un marin et un administrateur colonial français. Né le 11 février 1699 à Saint-Malo, en Bretagne, il est principalement connu pour son rôle en tant que gouverneur des îles de France (aujourd'hui l'île Maurice) et de Bourbon (aujourd'hui La Réunion) au XVIIIe siècle.

La Bourdonnais a commencé sa carrière maritime à l'âge de 17 ans et est rapidement devenu un marin expérimenté. Il a acquis une grande réputation grâce à ses talents de navigateur et de stratège militaire. En 1735, il a été nommé gouverneur des îles de France et de Bourbon, des postes qu'il a occupés pendant plusieurs années.

# Recherches en cours au niveau international

## Un exemple orienté IA neuro-symbolique :

FR

Exprime cette réponse en Prolog avec des faits et des règles.



Voici une formulation de la réponse en Prolog, utilisant des faits et des règles pour représenter les informations sur Mahé de La Bourdonnais :

prolog

Copy code

```
% Faits
marin(mahe_de_la_bourdonnais).
naissance(mahe_de_la_bourdonnais, 1699, saint_malo).
gouverneur(mahe_de_la_bourdonnais, iles_de_france).
gouverneur(mahe_de_la_bourdonnais, iles_de_bourbon).
poste(gouverneur).
reforme(mahe_de_la_bourdonnais, developpement_economique).
reforme(mahe_de_la_bourdonnais, defenses).
exploits(mahe_de_la_bourdonnais, guerre_de_succession).
accusation(mahe_de_la_bourdonnais, corruption).
emprisonnement(mahe_de_la_bourdonnais, 2).
mort(mahe_de_la_bourdonnais, paris, 1753).

% Règles
navigateur(X) :- marin(X).
stratège_militaire(X) :- marin(X).
promouvoir(X, developpement_economique) :- gouverneur(X, iles_de_france).
renforcer(X, defenses) :- gouverneur(X, iles_de_france).
victoire(X) :- exploits(X, guerre_de_succession).
soumis_enquete(X) :- accusation(X, corruption).
libere_sans_charges(X) :- emprisonnement(X, 2), \+ accusation(X, _).

% Questions
?- navigateur(mahe_de_la_bourdonnais).
?- naissance(mahe_de_la_bourdonnais, Annee, Lieu).
?- gouverneur(mahe_de_la_bourdonnais, iles_de_france).
?- promouvoir(mahe_de_la_bourdonnais, developpement_economique).
```

## En local :

- ▶ analyse de terminaison
- ▶ inférence de conditions de terminaison
- ▶ analyse de non-terminaison
- ▶ optimalité de conditions de terminaison
- ▶ PL, PLC, Java Bytecode, Android
- ▶ typage descriptif / prescriptif
- ▶ les flottants
- ▶ génération concolique de tests

avec Roberto Bagnara (Parme), Sébastien Hoarau (UR), Wim van Hoof (Namur), Andy King (Kent), Vitaly Lagoon (Melbourne), Fonenantsoa Maurica (UR), Ulrich Neumerkel (Vienne), Etienne Payet (UR), Salvatore Ruggieri (Pise), Alexander Serebrenik (Louvain), Fausto Spoto (Vérone), Peter Stuckey (Melbourne), Germàn Vidal (Valence) ...

En local :

En cours :

- ▶ non-termination
- ▶ termination pour les stratégies équitables
- ▶ compétition de termination (1er 2022 catégorie LP)
- ▶ intégration de solveurs SMT, Nicolas Lhost (Namur)
- ▶ preuves interactives de programmes logiques en LPO avec Thierry Marianne (UR)

Et voilà !

Questions, remarques ?

[frederic.mesnard@univ-reunion.fr](mailto:frederic.mesnard@univ-reunion.fr)

# Quelques applications notables

Où des parties logicielles ont été écrites en Prolog :

- ▶ **Watson** IBM, USA

Fodor 08

Système de réponse à des questions en langage naturel

Bancaire, juridique, médecine, ...

Prolog pour la partie analyse du langage naturel

- ▶ **NexSIS 18-112** Sécurité civile, FR

Narboni 22

Unification des systèmes d'information et de

commandement des services d'incendie et de secours

PLC pour la planification des mesures et moyens à prendre



# Un exemple de PLC : la spécification du tri d'entiers

En SWI-Prolog :

```
:- use_module(library(clpfd)).
```

```
sort(L,S) :-  
    permutation(L,S),  
    sorted(S).
```

```
sorted([]).  
sorted([_]).  
sorted([X,Y|Zs]) :-  
    X #=<= Y,  
    sorted([Y|Zs]).
```

```
permutation([],[]).  
permutation([X|Xs],Ys) :-  
    permutation(Xs,Zs),  
    select(X,Ys,Zs).
```

```
select(X,[X|Xs],Xs).  
select(X,[Y|Ys],[Y|Zs]) :-  
    select(X,Ys,Zs).
```

```
?- sort([3,2,0,1],L).  
L = [0, 1, 2, 3] ;  
false.
```

```
?- sort([A,B,C,D],[3,2,0,1]).  
false.
```

```
?- sort([A,B],[B,A]).  
A = B, B in inf..sup ;  
A#>=B ;  
false.
```

```
?- sort([A,B,C,D],[1,2,3]).  
false.
```

```
?- sort([A,B,C],[1,2,3]).  
A = 1, B = 2, C = 3 ;  
A = 2, B = 1, C = 3 ;  
A = 3, B = 1, C = 2 ;  
A = 1, B = 3, C = 2 ;  
A = 2, B = 3, C = 1 ;  
A = 3, B = 2, C = 1 ;  
false.
```