PROgrammer en LOGique

Philippe Morignot
pmorignot@yahoo.fr

Plan du cours de PROLOG

- **<u>Mercredi 24 mai 2023 :</u>**
  - Cours magistral (1h)
  - Travaux Pratiques notés 1/2 (2h)

- **<u>Mercredi 31 mai 2023 :</u>**
  - Travaux Pratiques notés 2/2

Généralités

- Conçu pour représenter la partie intelligente d'un logiciel.
- Déclaratif et non procédural comme Python, C++, JAVA, etc.

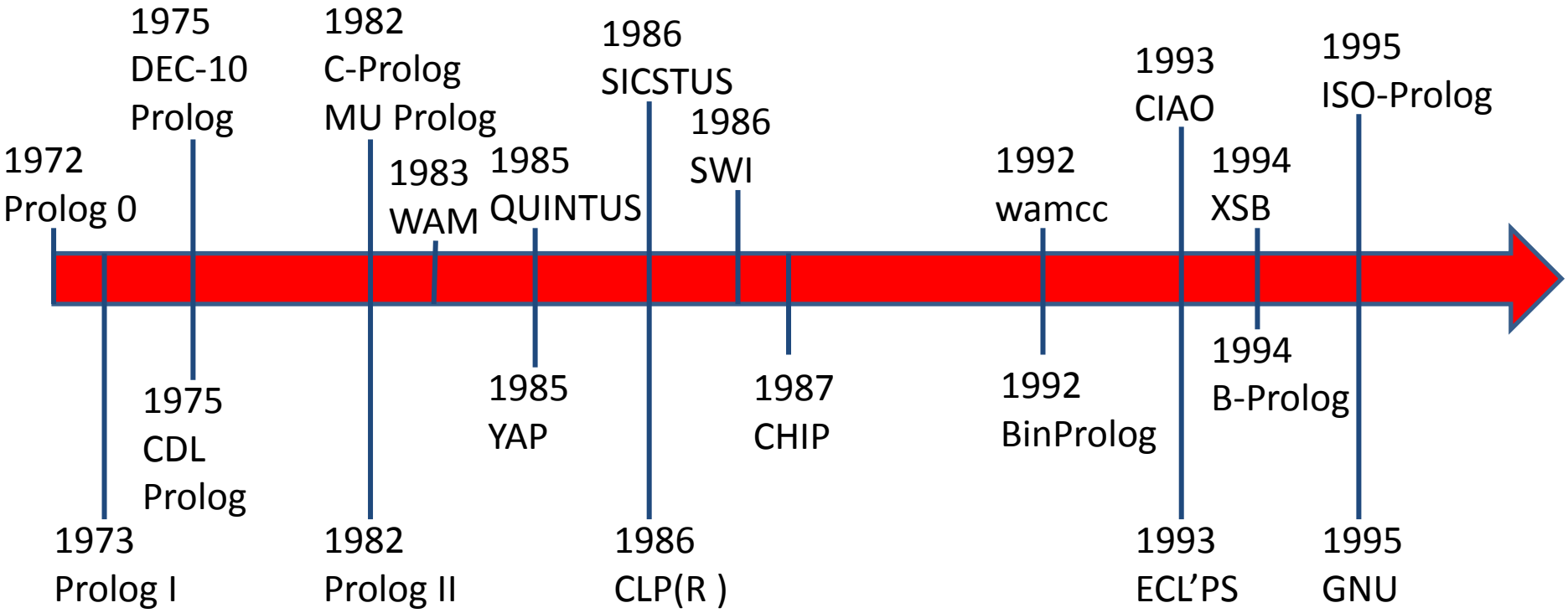
  On ne décrit pas par où doit passer le flux de contrôle,

  mais on donne les éléments en vrac.
- Basé sur des règles et des faits.

  $terme1 \wedge terme2 \wedge terme3 \rightarrow terme4 \wedge terme5$

  $terme6$
- Peut être considéré comme un démonstrateur ou comme un langage de programmation.

Histoire



- Proposé en 1972 par Alain Colmerauer et ses collègues à Marseille.
- <u>Dialectes en 2023 :</u> GNU Prolog, SICSTUS Prolog, SWI Prolog, etc.

Constituants

- PROLOG considère des clauses de Horn

  $tete \leftarrow corps1 \wedge corps2 \wedge corps3$

  Règle dont la tête est composée d'un terme unique.

- Base de faits : ensemble des clauses de Horn **sans** corps.

  $pere(paul, philippe) \leftarrow$

- Base de règles : ensemble des clauses de Horn **avec** corps.

  $grandpere(x,y) \leftarrow homme(x) \wedge parent(x,z) \wedge parent(z,y)$

- Question : $pere(x, philippe)$?

Moteur d'inférences

- Logique d'ordre 1
- Chaînage arrière : résolution guidée par les buts.
- Résolution en profondeur d'abord :

  fratrie(x,y) ← *frere(x, y)*
  fratrie(x,y) ← *soeur(x, y)*

  Les clauses d'un prédicat sont considérées dans l'ordre d'écriture.
  Risque de bouclage …
- Résolution par tentatives : retour-arrière (en anglais : *backtrack*) si échec.
- Négation par l'échec : *not(p)* réussit si le moteur d'inférences n'arrive pas à démontrer *p*.

Syntaxe

- **Constantes :** nombres entiers, nombres flottants, chaînes de caractères ne commençant pas par une majuscule.

  2 *3.14159* *parent* *philippe*

- **Variables :** chaînes de caractères commençant par une majuscule.

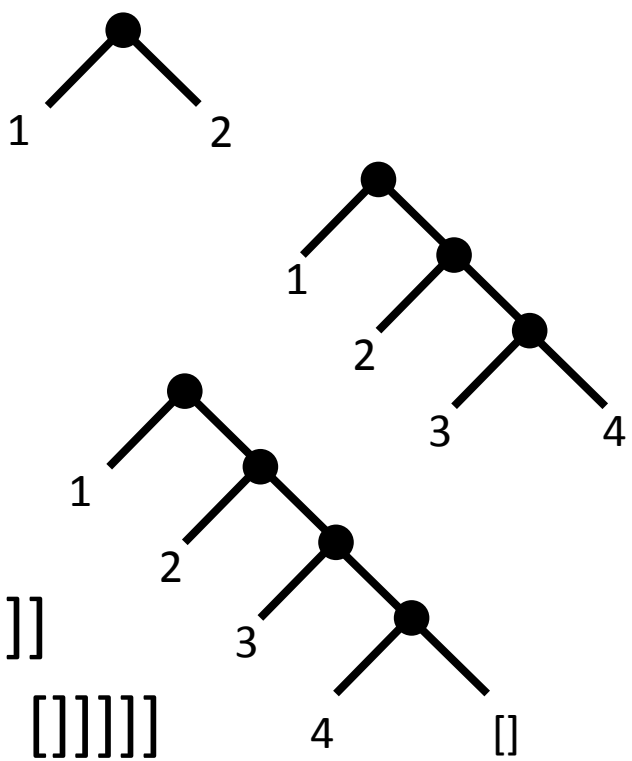
  UneVariable *X*

- **Prédicats :** nom commençant par une minuscule ; arguments pouvant être des constantes, des variables ou des prédicats.
- **Listes** *[1, 2, 3]*
- **Faits** *pere(paul, philippe).*
- **Règles** *parents(paul, odile, philippe) :-*

  pere(paul, philippe), mere(odile, philippe).

Listes

- [1, 2, 3, 4] (*crochets ouvrants et fermants*)
- [1, [2, 3], 4]
- [1 | 2] (*barre verticale*)
- [1, 2, 3 | 4]
- []
- [1, 2, 3, 4] = [1 | [2, 3, 4]]
  = [1 | [2 | [3, 4]]]
  = [1 | [2 | [3 | [4]]]]
  = [1 | [2 | [3 | [4 | []]]]]



Unification

- Définition :

  un terme *A* s'unifie avec un terme *B* ssi

  il existe une substitution de variables *S* telle que

  S(A) = S(B)

- Exemples :
  - A = [1 [2 3] 4 5] B = [1 X |Y]

    S = X / [2 3] ; Y / [4 5]
  - A = [1 X 2] B = [1 2 X]

    S = X / 2
  - A = [1 X 3 X] B = [1 Y Y 2]

    Pas de substitution !
- « _ » (underscore) : l'unification avec « _ » réussit toujours.

« cut »

- « *!* » : coupe tous les points de choix avant lui.

  Il s'agit d'une aberration, ne pas s'en servir trop souvent …

  cutl : *cut* local aux clauses d'un prédicat.

- *fail* : échoue toujours.
- *true* : réussit toujours.

- Exemple :

```
not(X) :- X, !, fail.
not(X).
```

Kit de survie

- *X is 1 + 2* Réussit. Evalue le terme de droite et l'unifie à la variable de gauche.
- *T1 ; T2* (Ou). Réussit si le terme *T1* réussit, ou sinon si le terme *T2* réussit.
- *X = Y* Réussit en unifiant la variable X à la variable Y
- *X == Y* Réussit si la variable *X* est équivalente à la variable *Y*
- *T =.. L* *T* est un prédicat *pred(arg1,arg2, arg3)* et *L* est une liste *[pred arg1 arg2 arg3]*
- *\+ T* (Not). Réussit si le moteur d'inférences n'arrive pas à prouver *T*.
- *read(X) write(X) writeln(X)* Réussit. Lecture et écriture à l'écran.

Exemples

% **Base de données familiale**

```
homme(paul).                    pere(X,Y) :- homme(X), parent(X,Y).
homme(philippe).                mere(X,Y) :- femme(X), parent(X,Y).
homme(andre).                   sœur(X,Y) :- femme(X), parent(Z,X), parent(Z,Y), \=(X,Y).
femme(dominique).               frere(X,Y) :- homme(X), parent(Z,X), parent(Z,Y), \=(X,Y).
femme(odile).                   fratrie(X,Y) :- sœur(X,Y) ; frere(X,Y).
parent(paul, philippe).         ancetre(X,Y) :- parent(X,Y).
parent(odile, philippe).        ancetre(X,Y) :- parent(X,Z), ancetre(Z,Y).
parent(paul, dominique).
parent(odile, dominique).       :- sœur(X, philippe) ?
parent(andre, paul).            :- fratrie(X, philippe) ?
                                :- ancetre(X, philippe) ?
```

% **Un terme est il membre d'une liste ?**

membre(X, [X | _]).
membre(X, [_ | Y]) :- membre(X,Y).
:- membre(1, [2 3 1 4 5]) ?

Conclusion

- PROLOG est un langage déclaratif permettant de représenter la partie intelligente d'un logiciel.
- PROLOG est basé sur des clauses de Horn, organisées en base de faits et base de règles.
- PROLOG se base sur la logique d'ordre 1, et considère les clauses en profondeur d'abord dans l'ordre d'écriture, avec retour-arrière si échec.
- Extension : programmation par contraintes.

Références

- Philippe Beaune, Gauthier Picard, Laurent Vercouter. *Initiation à l'intelligence artificielle*. Ecole Nationale Supérieure des Mines de Saint-Etienne, Pôle XXI 2010-2011.
- Journée *Les cinquante ans de PROLOG*. Université Paris 5, 10 novembre 2022. (Jean Rohmer)
- L. Sterling, E. Shapiro. *The art of Prolog*, 1994. Traduit en français chez Masson, 1990.
- Ulf Nilsson, Jan Maluszynski. *Logic, programming and Prolog*, 2nd edition, 2000. https://www.ida.liu.se/~ulfni53/lpp/