50 ans de Prolog, et après?

Pierre Deransart

DR retraité INRIA,
membre honoraire de l'AFPC,
membre de l'Institut de Recherche de la FSU

PFIA, St Etienne, 27 juin 2022

Arpentage

-Quelques aspects de programmation en logique,Polog et programmation par contraintes

-50 ans de Prolog…

-Histoire plus personnelle: spécifications de programme, Prolog et débogage

-La saga de la standardisation

-50 ans de Prolog vus de la France

-Et après?

-Questionnements

Repères historiques

1962 les prémisses

1972 PROLOG

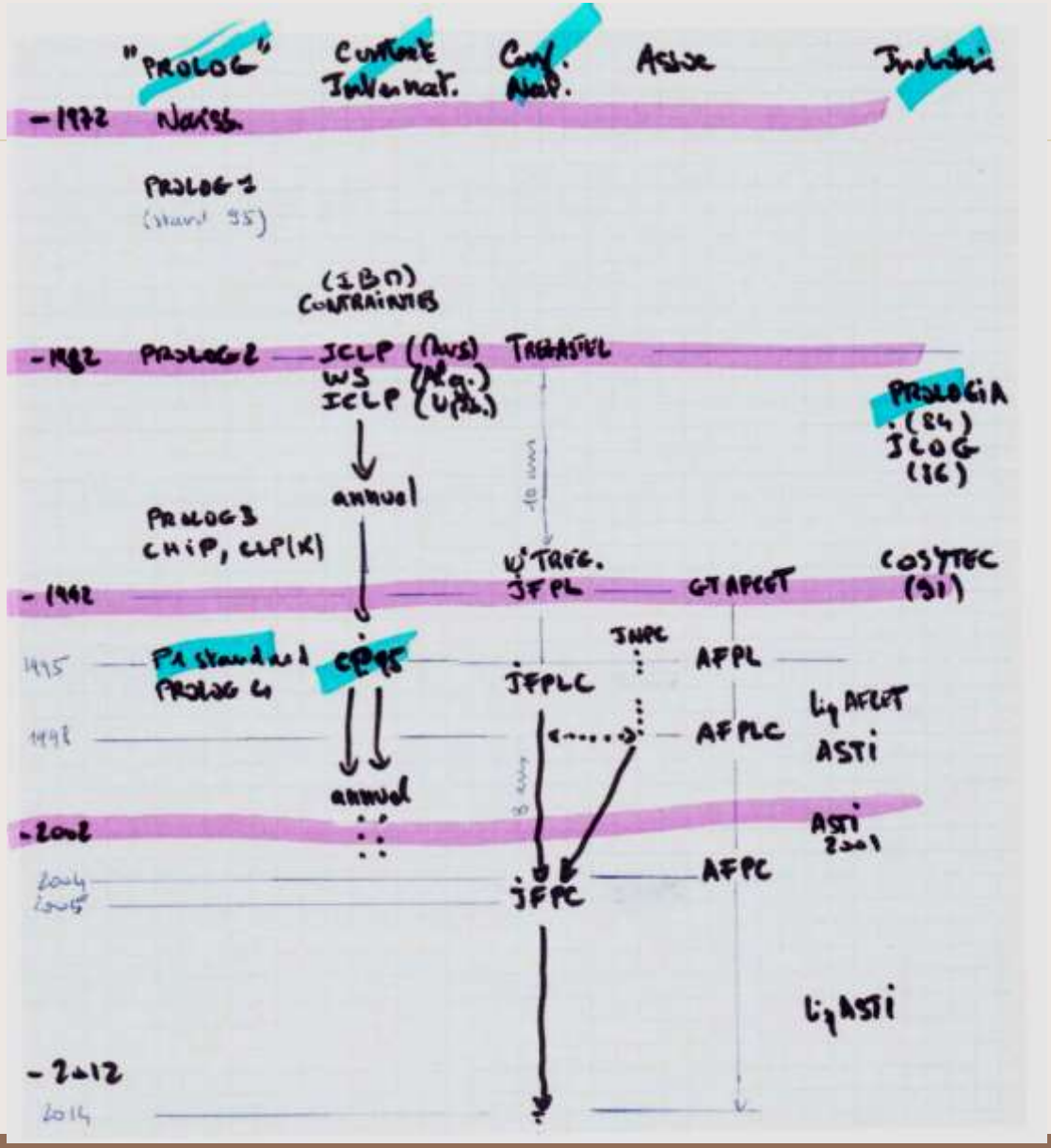
1982 IA 5ème génération computers

1992 Le tournant des contraintes

2002 …

2022 Futur?

Histoire, 50 ans de Prolog



Listes (Cervoni 2022)

L'objet de base en Prolog : la liste

[ ] = liste vide
[a, 123, « coucou »] = liste à 3 éléments

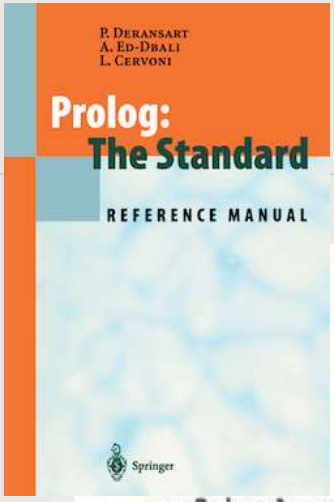
| = opérateur de construction de liste

[X | Y] = X tête, Y queue

Une liste est soit vide, soit un objet construit par | dont la queue est une liste.

```
liste([ ]).
liste([X | Y]) :- liste(Y).
```

Extrait de *Prolog, The Standard*




```
palindrome(L- L).
palindrome([A|L]- L).
palindrome([A|L]- M) :- palindrome(L- [A|M]).
```

Notice that with goals like `palindrom(l-m)`. where `l` is a list and `m` any term the program terminates and analyses or generates palindroms (i.e. `palindrom(l-m)` succeeds if and only if `l-m` denotes a list which is a palindrom), otherwise fails. If `l` is a variable or a partial list the program does not terminate, although it has some interesting behaviour.

However the two first clauses correspond to "risky" situations. If the goal

Extrait de *Prolog, The Standard*

```
trucmuch(L-L)
trucmuch([A|L]-L)
trucmuch([A|L]-M):- trucmuch(L-[A|M]).
```

```
palindrome(L) :- append(L1, L2, L), reverse(L1, L1).
append([], L, L).
append([A|M], L, [A|L']) :- append(M, L, L').

reverse([], []).
reverse([A|L], M) :- reverse(L, M'), append(L', [A], M).
```

Approche logique et déductive

John Alan Robinson
1930-2016

$P =$

```
palindrome(L- L).
palindrome([A|L]- L).
palindrome([A|L]- M) :- palindrome(L- [A|M])
```

$$P \models \exists g \iff P \vdash \exists g$$

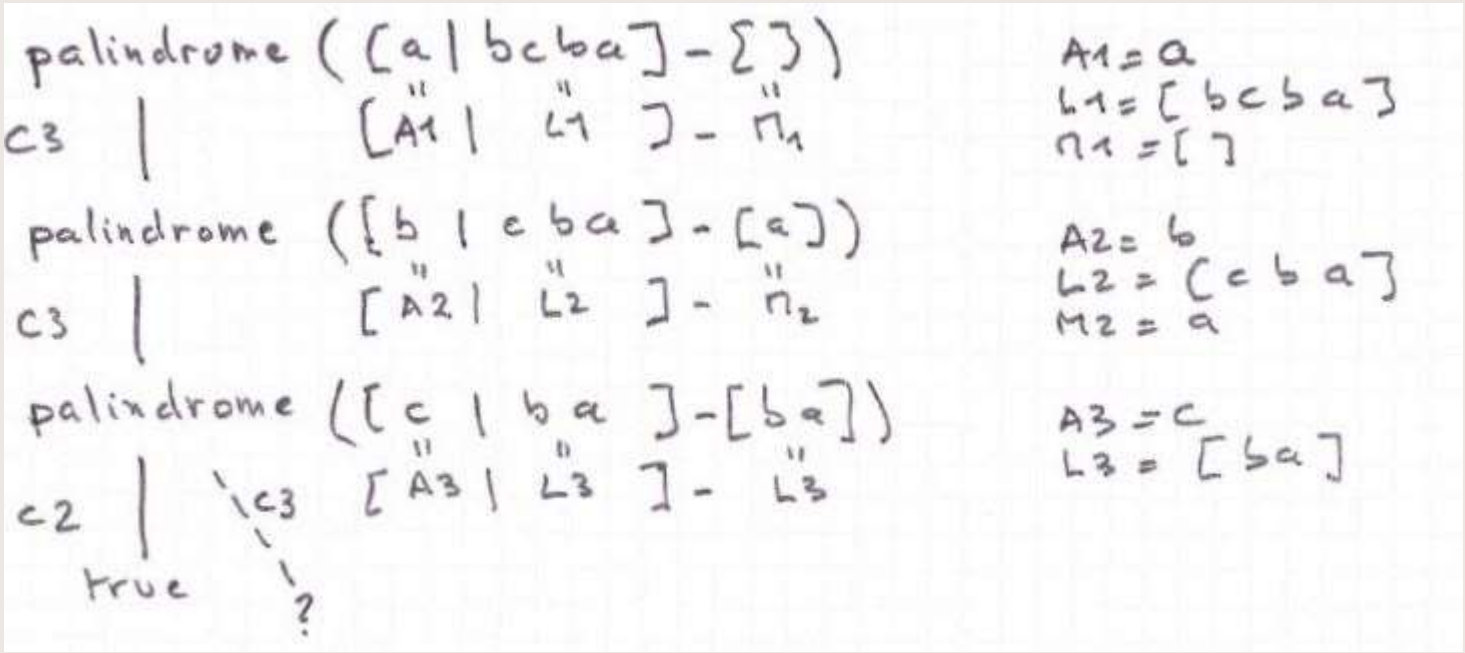
Palindrome ([a,b,c,b,a]-[])?

J.Alan Robinson

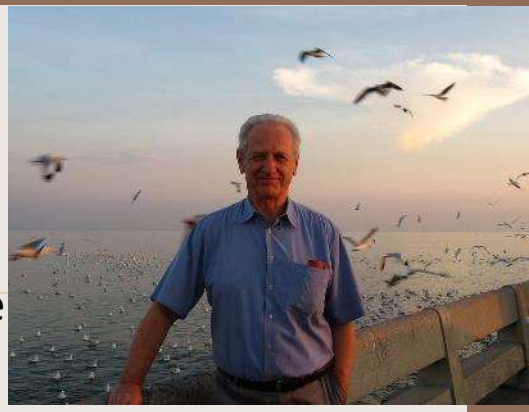
Preuve par inférences

Arbre de preuve + unification

"A Machine-Oriented Logic Based on the Resolution Principle" (**1965**), JACM 12



Kowalski: clauses de Horn
Arbre de preuve + unification



Kowalski, R. **1971**. Predicate Logic as Programming Language
IFIP 74

palindrome ([c | ba] - [ba])
c3 | [A3 | L3] - n3
A3 = c
L3 = [ba]
n3 = [ba]

palindrome ([b | a] - [cba])
c3 | [A4 | L4] - n4
A4 = b
L4 = [a]
n4 = [cba]

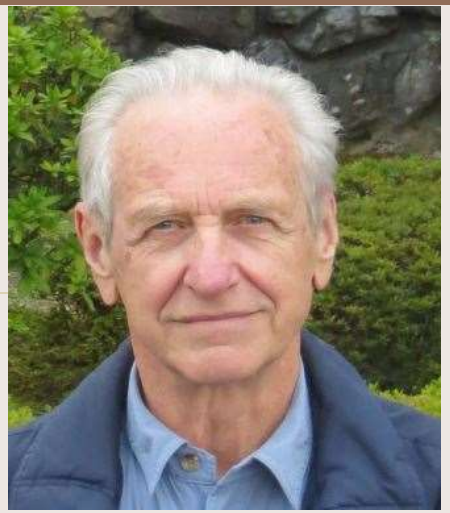
palindrome ([a | []] - [bcba])
c3 | [A5 | L5] - n5
A5 = a
L5 = []
n5 = [bcba]

palindrome ([] - [abcba])
| false
L6 = [] ≠ [abcba]
[A6 | L6] ≠ []

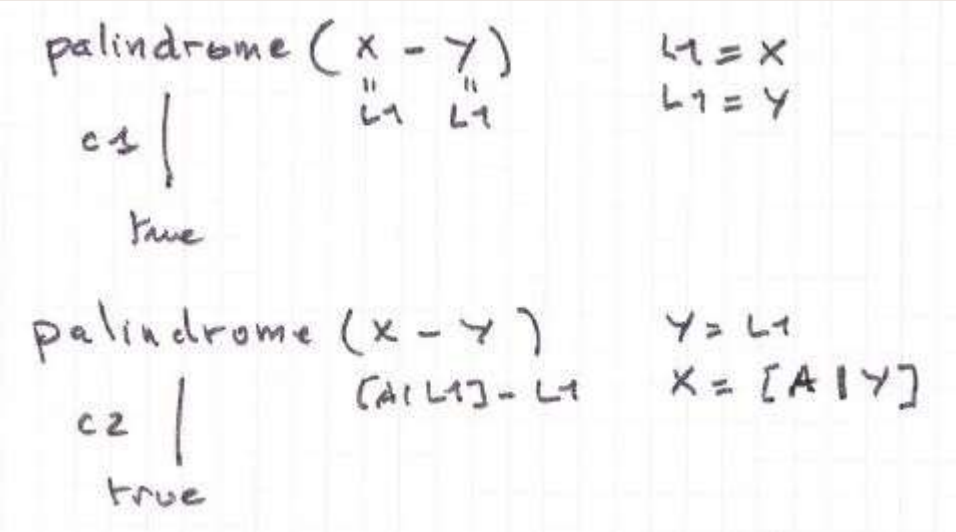
```
? Palindrome ( X – Y ).
```

Kowalski algorithm =

logic + **control**



Kowalski, R. **1979**. Algorithm = Logic + Control. Communications of the ACM 22, 7



Colmeraue

Prolog I

Arbre de

```
p(X, Y) :- q(X), r(X, Y).
p(X, Y) :- s(X).

s(d) :- true.

q(a) :- true.
q(b) :- true.
q(c) :- true.
```

Figure 4.2 shows the corresponding Prolog sear
predication underlined. Fresh renaming is denoted b
substitutions are represented before the node, besid

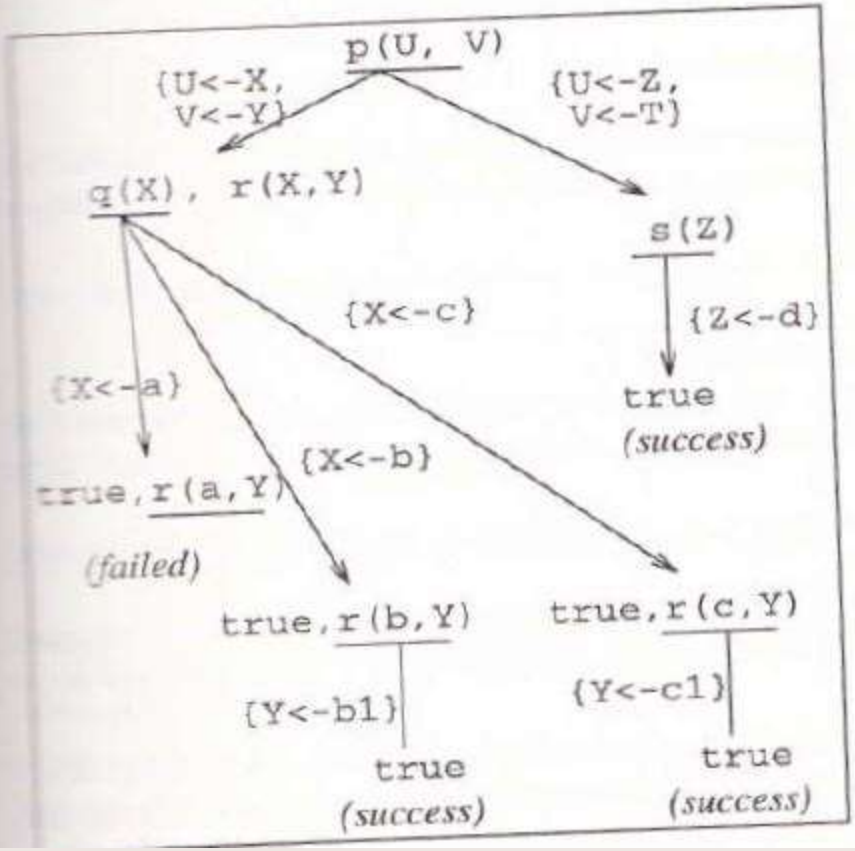


Fig. 4.2.
search-tree

Colmerauer

Prolog I (1972) avec coupure

```
p(X, Y) :- q(X), r(X, Y).   q(a) :- true.   r(b, b1) :- true.
p(X, Y) :- s(X).            q(b) :- true.   r(c, c1) :- true.
                            q(c) :- true.

s(d) :- true.
```

Figure 4.2 shows the corresponding Prolog search-tree with the predication underlined. Fresh renaming is denoted by new variables a substitutions are represented before the node, beside the incoming a

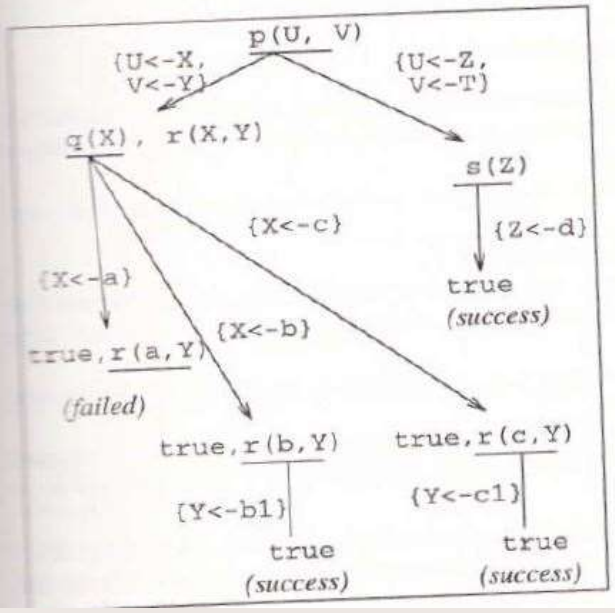
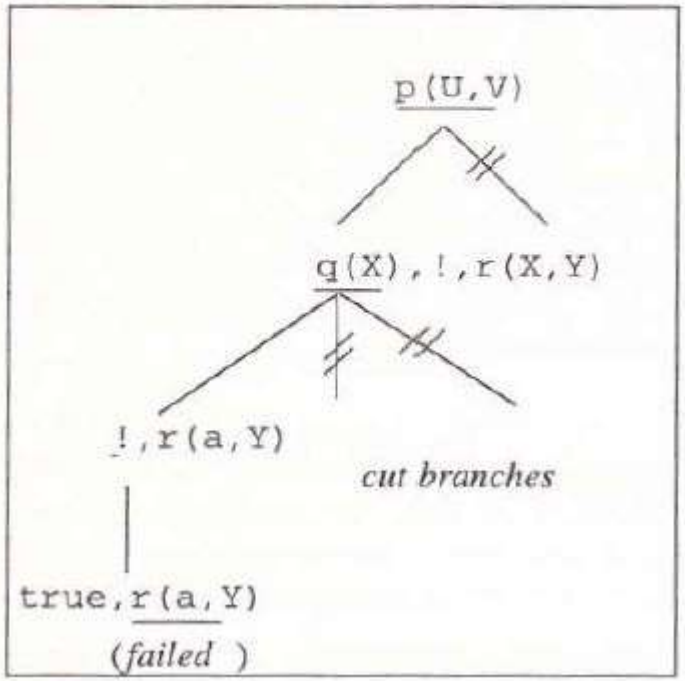


Fig. 4.2. search-tree

```
p(X, Y) :- q(X), !, r(X, Y).
p(X, Y) :- s(X).
```

Figure 4.6 shows that the search-tree cor depicted in the Figure 4.2, now has only one



Implantations

Philippe Roussel (Marseille)

Prolog I (**1972**)

Prolog, manuel de référence et d'utilisation,
Groupe Intelligence Artificielle,
Faculté des Sciences de Luminy, Université Aix-Marseille II,
France, septembre **1975**

David Warren (Edimburgh)

Warren, D. H. **1975**. **PROLOG to DEC 10** Machine Code Compiler.

David Warren (San Francisco)

Warren, D. H. **1983**. **An Abstract Prolog Instruction** Set. Tech. Rep. Technical Note, 309, Artificial Intelligence Center, Computer Science and Technology Division, SRI International.

Yves Bekkers, Bernard Canet, Olivier Ridoux,
Lucien Ungaro: A **Memory Management Machine**
for Prolog Interpreter. **ICLP 1984**: 343-353

1947-2020

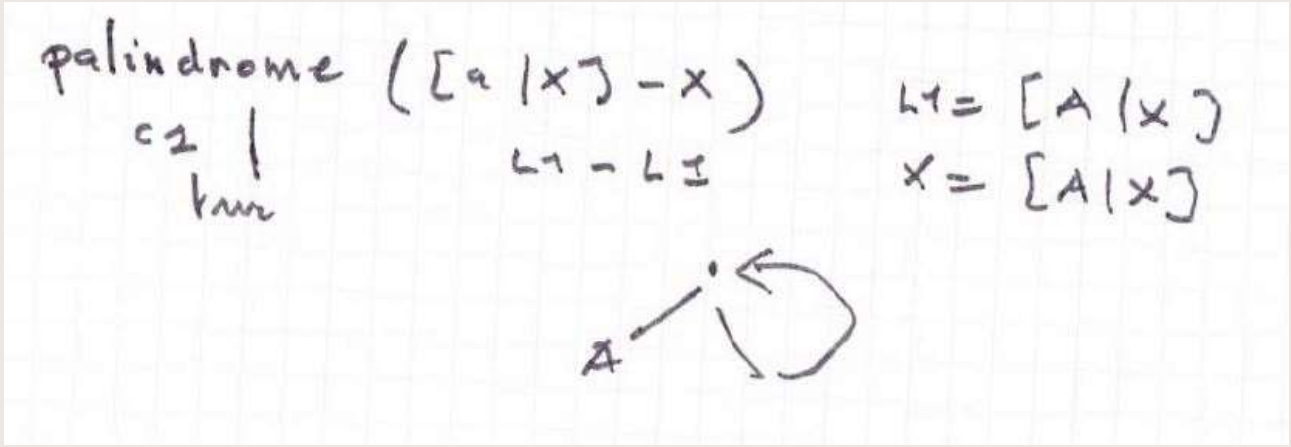
Palindrome ([a|X]- X) ?

A term is a tree

Infinite terms?

Colmerauer Prolog II

```
palindrome(L- L).
palindrome([A|L]- L).
palindrome([A|L]- M) :- palindrome(L- [A|M])
```



Le tournant des contraintes

Jean-Louis Lassez CLP(ℝ)

Jaffar, J. and Lassez, J. **1987**. Constraint Logic Programming.
In Proceedings POPL. ACM, 111–119.

CLP(ℝ) réels

1968

Catherine et
Jean-Louis Lassez

L'idée d'une généralisation à des langages de contraintes arbitraires commence à s'imposer à partir du milieu des années 80 (cf. les travaux théoriques de J-L. Lassez, J. Jaffar et M. Maher au centre d'IBM de Yorktown)

SOLVING NEW PROBLEMS

Mortgage example

which is the relation between : ?

C	Capital
D	Delay
J	Profit
B	Balance
M	Monthly paiement

Mortgage (C, 1, J, B, M) ⇐
{B + M = C∗(1 + J)}

Mortgage (C, D, J, B, M) ⇐
{D ≥ 2},
Mortgage(C∗(1+J) - M, D-1, J, B, M)

Q : Mortgage(120000, 120, 0.01, 0, M) ?
A : M = 1721.65

Q : Mortgage(C, 120, 0.01, 0, 1721.65) ?
A : C = 120000

Q : Mortgage(C, 120, 0.01, 0, M),
{ 100000 ≤ C ≤ 150000}?
A : M = 0.01434717∗C, 1434.71 ≤ M
≤ 2152.06

Q : Mortgage(C, 120, 0.01, B, M) ?
A : B = (3.30039∗C,) - (230.039∗M)

https://www.univ-montp3.fr/miap/~jq/OptionIA/cours/cspGoualard.pdf (2002)
Adapté de Eric Monfroy

CSP

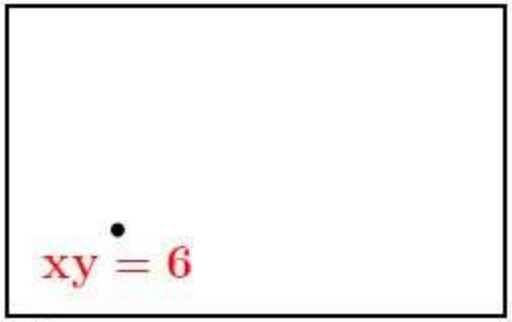
Un *problème de satisfaction de contraintes* (CSP) est défi ni par la donnée d'une contrainte n-aire $C(x_1, \ldots, x_n)$ sur une structure Σ et de n *domaines* $D_1, \ldots, D_n$ pour les variables. Le CSP représente implicitement la contrainte :

$$C \wedge x_1 \in D_1 \wedge \cdots \wedge x_n \in D_n$$

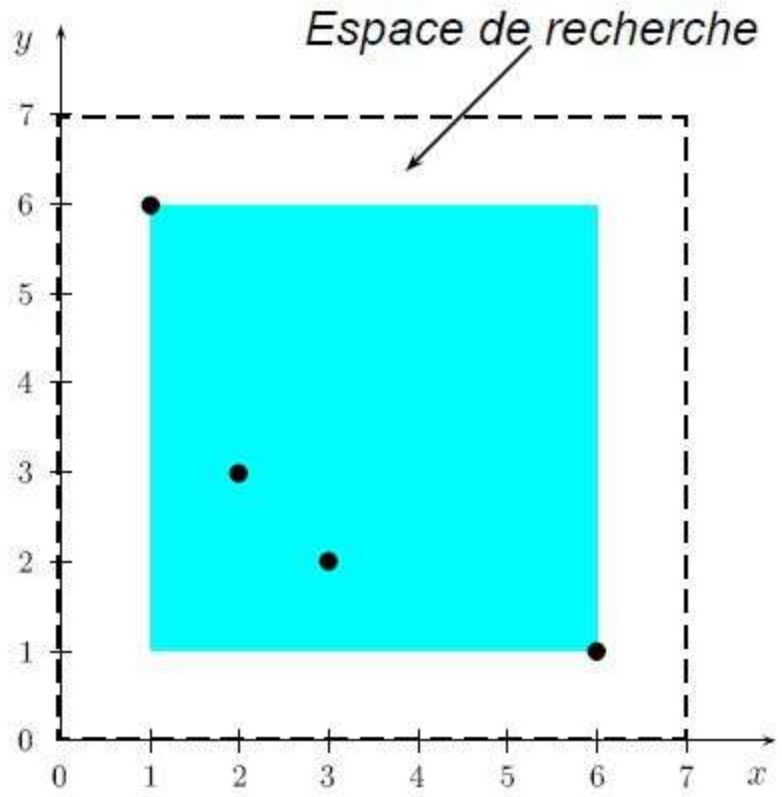
Une *solution du CSP* est un n-uplet $(a_1, \ldots, a_n)$ appartenant à $D_1 \times \cdots \times D_n$ et satisfaisant C.

Dualité de la PC (1)

```
1 :- x in 0..7,
2    y in 0..7,
3    x*y $= 6,
4    x+y $= 5,
5    x   $< y.
```



Store



Solveur de contraintes

Étant donné une contrainte c, on peut étudier les problèmes suivants :

- *satisfaction* : la contrainte c est-elle satisfiable ? (Existe t'il une valuation des variables de c telle que c est vraie)
- *solution* : si c est satisfiable, exhiber une ou plusieurs solutions
- *optimisation* : exhiber une solution optimale (concept à définir)
- *simplification* : transformer c en une *contrainte équivalente* (i.e. avec le même espace de solutions)

On ne s'intéressera ici qu'aux deux premiers problèmes

Programmation

Différents paradigmes de programmation

- Programmation structurée

  Pascal, C,. . .
- Programmation fonctionnelle

  Lisp, Caml,. . .
- Programmation orientée objets

  C++, Java,. . .
- Programmation logique

  Prolog, Mercury,. . .

Programmation par contraintes

$\Rightarrow$ orthogonale au paradigme

Contraintes « globales »

- *atmost*$(2, [X_1, X_2, X_3, X_4, X_5], 1)$
  Au plus 2 variables parmi $\{X_1, X_2, X_3, X_4, X_5\}$ sont égales à 1
- *alldiff*$([X_1, X_2, X_3, X_4, X_5])$
  Les variables $\{X_1, X_2, X_3, X_4, X_5\}$ ont des valeurs différentes deux à deux

Systèmes de PC (1)

- **CLP($\Re$)**. Système basé sur Prolog. Résolution de contraintes linéaires sur les réels (équations et inéquations). Utilisation de l'élimination de Gauss et du Simplexe. Contraintes non-linéaires mises en attente. Système incorrect et incomplet
- **Prolog IV**. Système basé sur Prolog. Résolution de contraintes (non-)linéaires sur les rationnels et les réels, les contraintes sur les arbres (équations, diséquations) et sur les chaînes de caractères (équations). Système fiable et complet (suivant algos)
- **CHIP**. Système basé sur Prolog. Contraintes linéaires sur les rationnels, contraintes booléennes.

Systèmes de PC (2)

- **ECLiPSe**. Système basé sur Prolog. Contraintes sur les domaines finis, contraintes linéaires sur les réels, *Constraint Handling Rules*
- **Ilog Solver**. Librairie C++. Contraintes sur les domaines finis et les réels (fiable et complet)
- **Numerica**. Système avec son propre langage. Contraintes (non-)linéaires sur les réels. Fiable et complet
- **GNU Prolog**. Prolog avec une extension pour la résolution de contraintes sur les domaines finis.

Négation

```
rase(barbier, X)
        :- not rase(X, X)

? rase(barbier, barbier)
```

$p \Leftarrow \sim p$

Négation
Familles de problèmes

Implantation, négation par échec:

not(p):- p, /, fail

not(p):- true.

Sémantique de PL avec négation

Raisonnement non monotone

Answer set programming (ASP)

…

Grammaires
Chomsky 1957

Grammaire de métamorphose
DCG

Véronica Dahl 1977

III.59. Une grammaire non-déterministe indépendante du contexte se traduit directement en un programme Prolog. Le premier interpréteur Prolog a été réalisé en 1972 au GIA de l'Université de Marseille dans ce but. Par exemple la grammaire :

```
sentence ::= nounphrase, verbphrase;
nounphrase ::= determiner, noun | noun;
verbphrase ::= verb | verb, nounphrase;
verb ::= [eats];
determiner ::= [the];
noun ::= [monkey] | [banana];
```

Fages 1996

Gross, Maurice (1934-2001)
Lentin, André (1913-2015)
Chomsky, Noam (1928-), préface

Grammaires

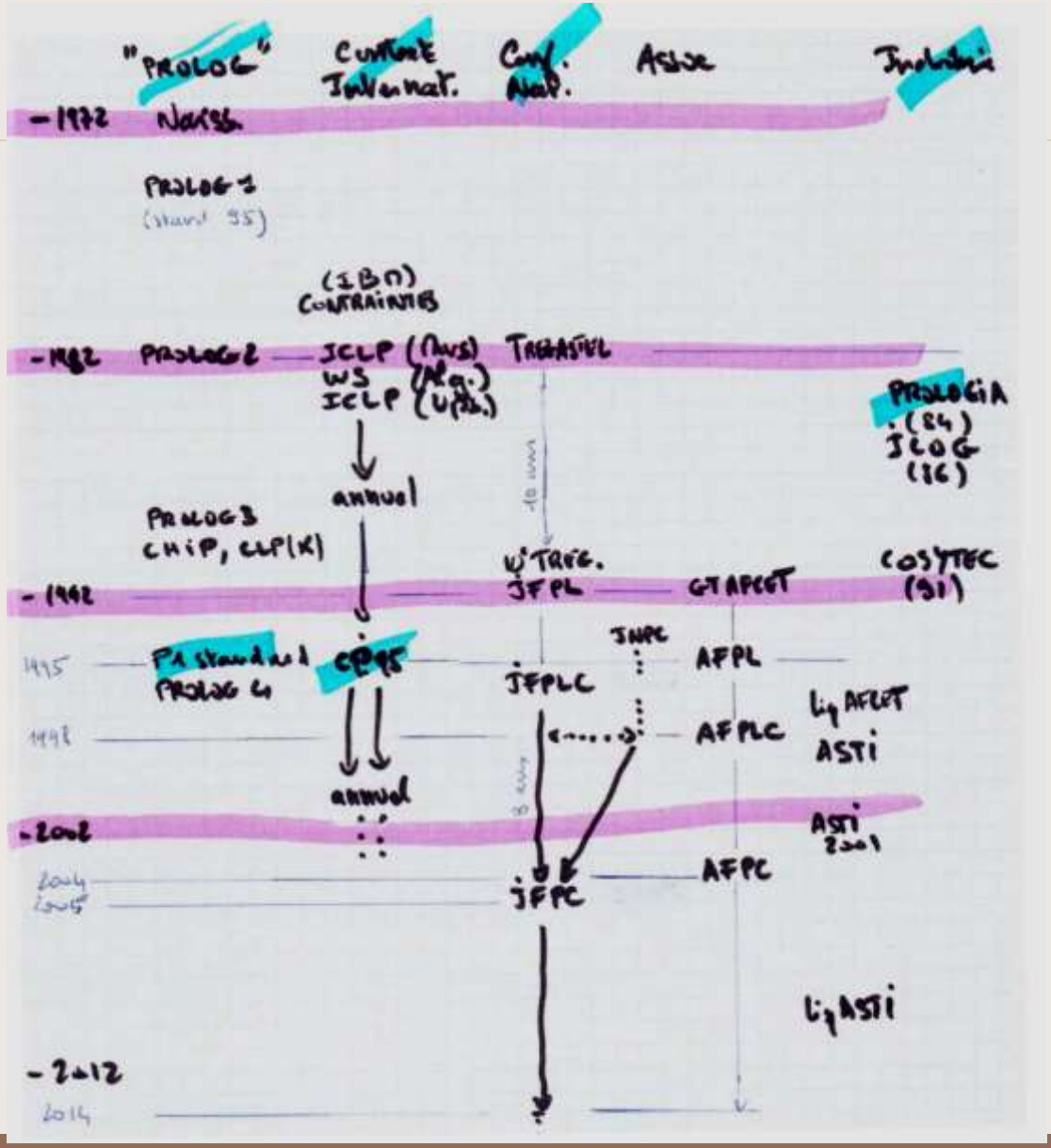
PROGRAMME III.60.— Analyse syntaxique et synthèse.

```
Sentence(L):-nounphrase(L1), verbphrase(L2), append(L1,L2,LX
nounphrase(L):- determiner(L1), noun(L2), append(L1,L2,L).
nounphrase(L):- noun(L).
verbphrase(L):- verb(L).
verbphrase(L):- verb(L1), nounphrase(L2), append(L1,L2,L).
verb([eats]).
determiner([the]).
noun([monkey]).
noun([banana]).
```

```
| ?- sentence([the,monkey,eats]).

yes

| ?- sentence([the,eats]).

no

| ?- sentence(L).

L = [the,monkey,eats] ? ;

L = [the,monkey,eats,the,monkey] ? ;

L = [the,monkey,eats,the,banana] ? ;

L = [the,monkey,eats,monkey] ?

yes
```

Fages 1996

Histoire, 50 ans de Prolog



Prolog:

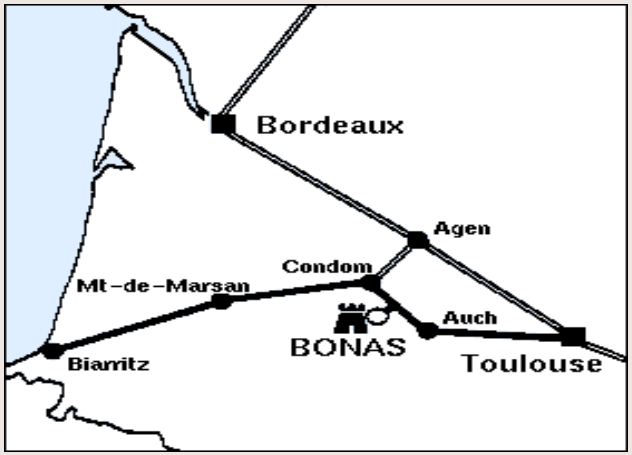
un pas majeur vers la programmation déclarative

(petite histoire plus personnelle)

Our story starts for me at Bonas in 1980

(a seminar center located in an old castle restored by Jean-Claude Simon in 1972)

(1923-2000)



International Workshop on Program Construction
A.Biermann, G.Guiho, Y.Kodratoff eds, 1980

A strong debate ?….

Thesis: « Prolog is purely declarative, since one has just to write axioms in FOL (just Horn clauses)… It works like an executable specification language, and there is nothing to prove about it»

Surprise: …. « Which kind of miracle is this …. !?!! »

Hervé Gallaire

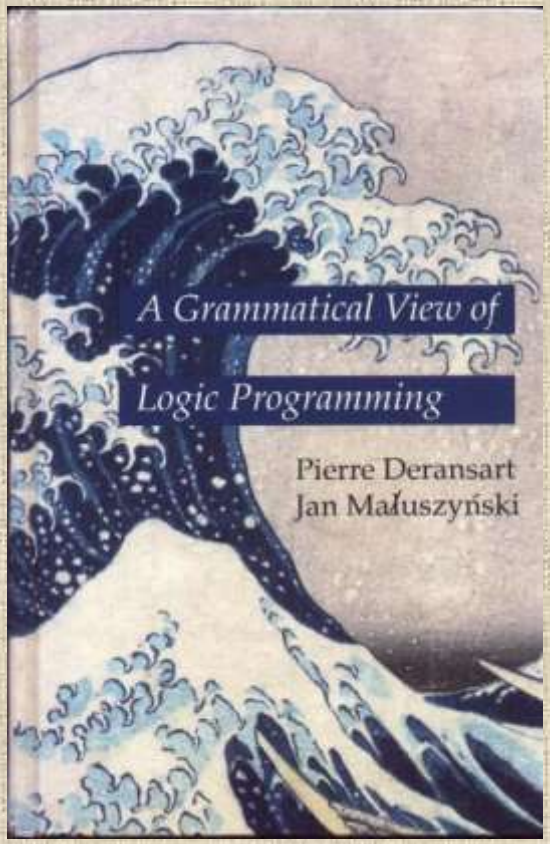
Michel Sintzoff

Travaux avec Jan Maluszynski (2020)

- **1982** First ICLP, Marseille, Pierre and Jan's first encounter
- **1983** ***Modeling Data Dependencies in LP by Attribute Schemata***, Perroz-Guirrec
- 1985 *Relating Logic Programs and Attribute Grammars*, JLP
- **1988** First PLILP, Orléans, with B. Lorho
- 1989 (Lauching WAGA)
- **1993** ***A Grammatical View of Logic Programming*** , The MIT Press
- 1996 *Une méthode de preuve pour programmes concurrents avec contraintes sous forme clausale*, with Pascal Chambre, JFPLC
- 2000 ***Analysis and Visualization Tools for Constraint Programming***, Springer

PLILP

(1988)



(1993)

Travaux avec Gérard Ferrand

- 1985 *Détection d'erreurs en programmation en logique* with G. Ferrand, SPLT
- 1987 *Formal Specification of Standard Prolog*, with GRichard and C.Moss,SPLT
- **1988** ***Proofs of Partial Correctness for Attribute Grammars with Applications to Recursive Procedures and LP***, with B.Courcelle, Inf &Control

- 1989 ***Methodological View of LP with Negation***, with G. Ferrand
- **1991** ***NSTO Programs***, with G. Ferrand and M. Téguia, ISLP, San Diego
- **1992** ***Proof Method of Partial Correctness and Weak Completeness for Normal LP***, with G. Ferrand, JICSLP, Washington, JLP 1993

- **1992** ***An Operational Formal Definition of Prolog: A specification Method and Its Application***, with G. Ferrand, New Generation Computing
- **1993** *Proof Methods of Declarative Properties of Definite Programs*, TCS

NSTO PROGRAMS

(Not Subject To Occur-check)

Pierre DERANSART
INRIA - Rocquencourt
Domaine de Voluceau
B.P. 105
78153 LE CHESNAY Cedex
FRANCE

with

Jan MALUSZYNSKI
LINKÖPING - SUEDE

Gérard FERRAND
ORLEANS - FRANCE

Michel TEGUIA
TOURS - FRANCE

Jean-Louis BOUQUARD
TOURS - FRANCE

and the members of the working group : ISO SC22-WG17,

in particular R.S. SCOWEN, C. BIERLE and

some remarks of H. SONDERGAARD, J. M. CORSINI and

of R.O'KEEFE and D. BOWEN

Travaux sur le débogage

State of the Art

- **Declarative debugging**

  Drabent Nadm-Tehrani, Małuszyǹski 89

  Lloyd 87, Bergère 92: adaptation to normal programs.

  Granvillier 94, Binks 96: implementation in Goedel. Many strategies implemented, graphic tools.

  Bergère, Ferrand, Le Berre, ...95: PLC revisited...

  Le Berre, Tessier 95: DD and CLP (Incorrectness).

Comments as Assertion

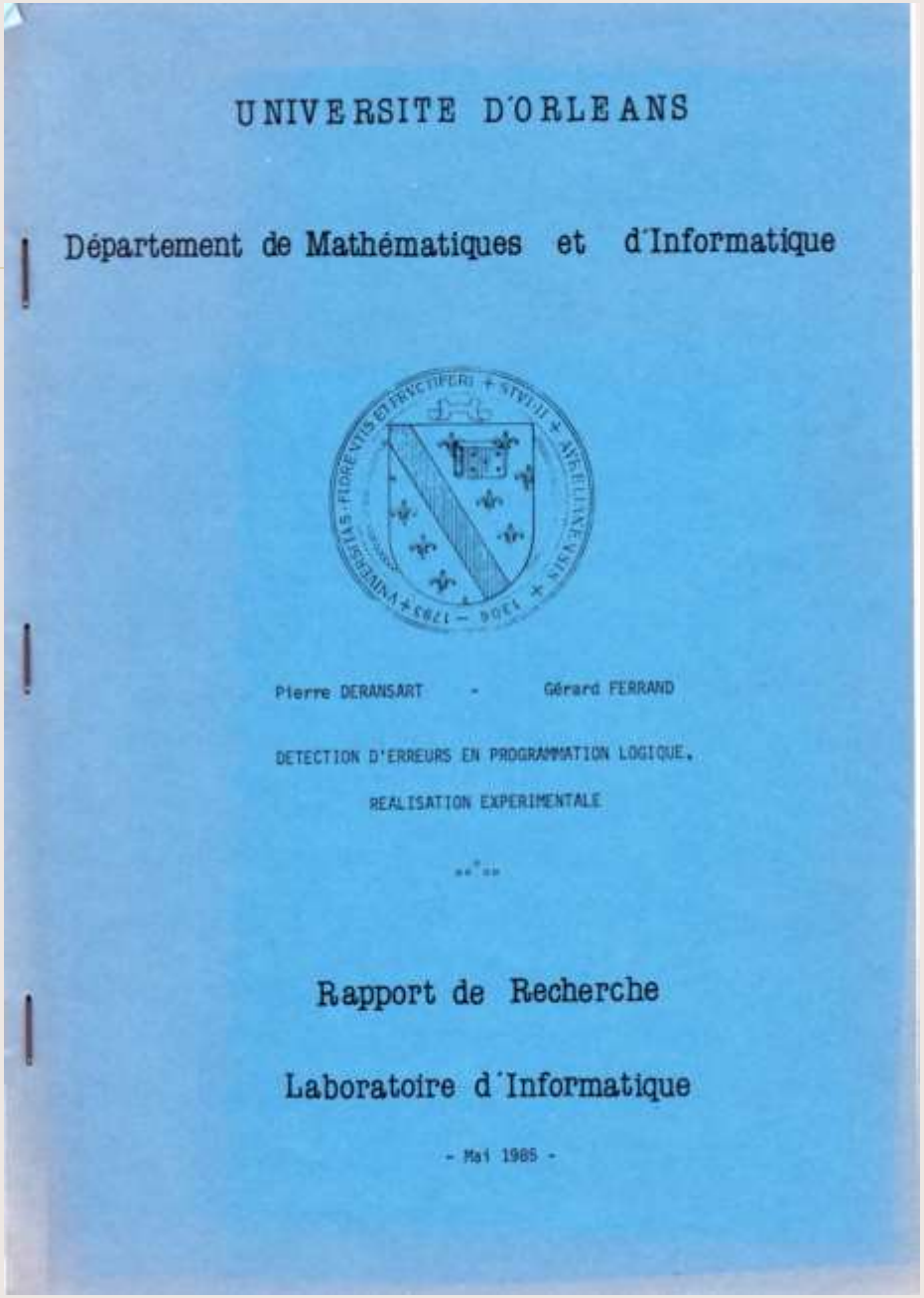
Pierre Deransart, Gérard Ferrand, An Operational Formal Definition of Prolog: A Specification Method and Its Application, New Generation Computing, 1992

inclus (L_1, L_2) ← not **ninclus** (L_1, L_2)
ninclus (L_1, L_2) ← **elem** (E, L_1), not **elem** (E, L_2)
elem (E, [E | L]) ←
elem (E, [H | L]) ← **elem** (E, L)

Assertions exprimant: Correction partielle Complétude

	S	C
inclus (L_1, L_2)	L_1, L_2 lists $\Rightarrow$ $L_1 \subseteq L_2$	L_1, L_2 lists and $L_1 \subseteq L_2$
ninclus (L_1, L_2)	L_1, L_2 lists $\Rightarrow$ $L_1 \not\subseteq L_2$	L_1, L_2 lists and $L_1 \not\subseteq L_2$
elem (E, L)	L list $\Rightarrow$ E $\in$ L	L list and E $\in$ L

Débogage déclaratif
(1985)



UNIVERSITE D'ORLEANS

Département de Mathématiques et d'Informatique

Pierre DERANSART - Gérard FERRAND

DETECTION D'ERREURS EN PROGRAMMATION LOGIQUE.

REALISATION EXPERIMENTALE

Rapport de Recherche

Laboratoire d'Informatique

- Mai 1985 -

Débogage déclaratif

Ayant l'intention d'écrire un programme pour inverser une liste, et utilisant des notations usuelles, on a écrit en fait le programme logique P suivant :

renv(nil,nil) ←

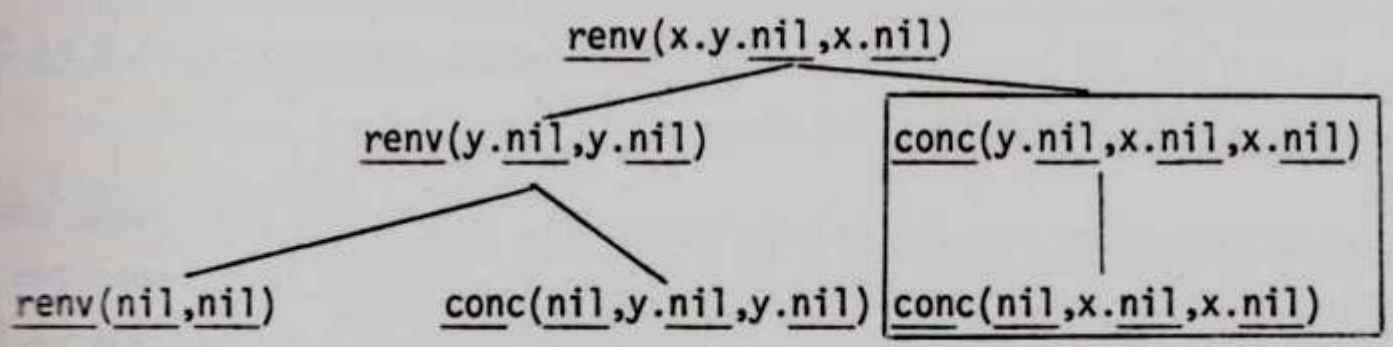
renv(x.y,z) ← renv(y,t) , conc(t,x.nil,z)

conc(nil,x,x) ←

conc(x.y,z,t) ← conc(y,z,t)

Dans la dénotation ("de fait") D(P) il y a l'atome renv(x.y.nil,x.nil), qui n'est pas dans la dénotation attendue I, c'est donc un symptome d'incorrection de P dans I.

Ce symptome d'incorrection est racine d'un arbre, que nous disons conforme à P car chaque noeud correspond à une instance d'une clause de P :



Ce qu'on a encadré met en évidence une incorrection de P dans I.

JFPLC 97, Orléans

The HyperPro Project (1994-2002)

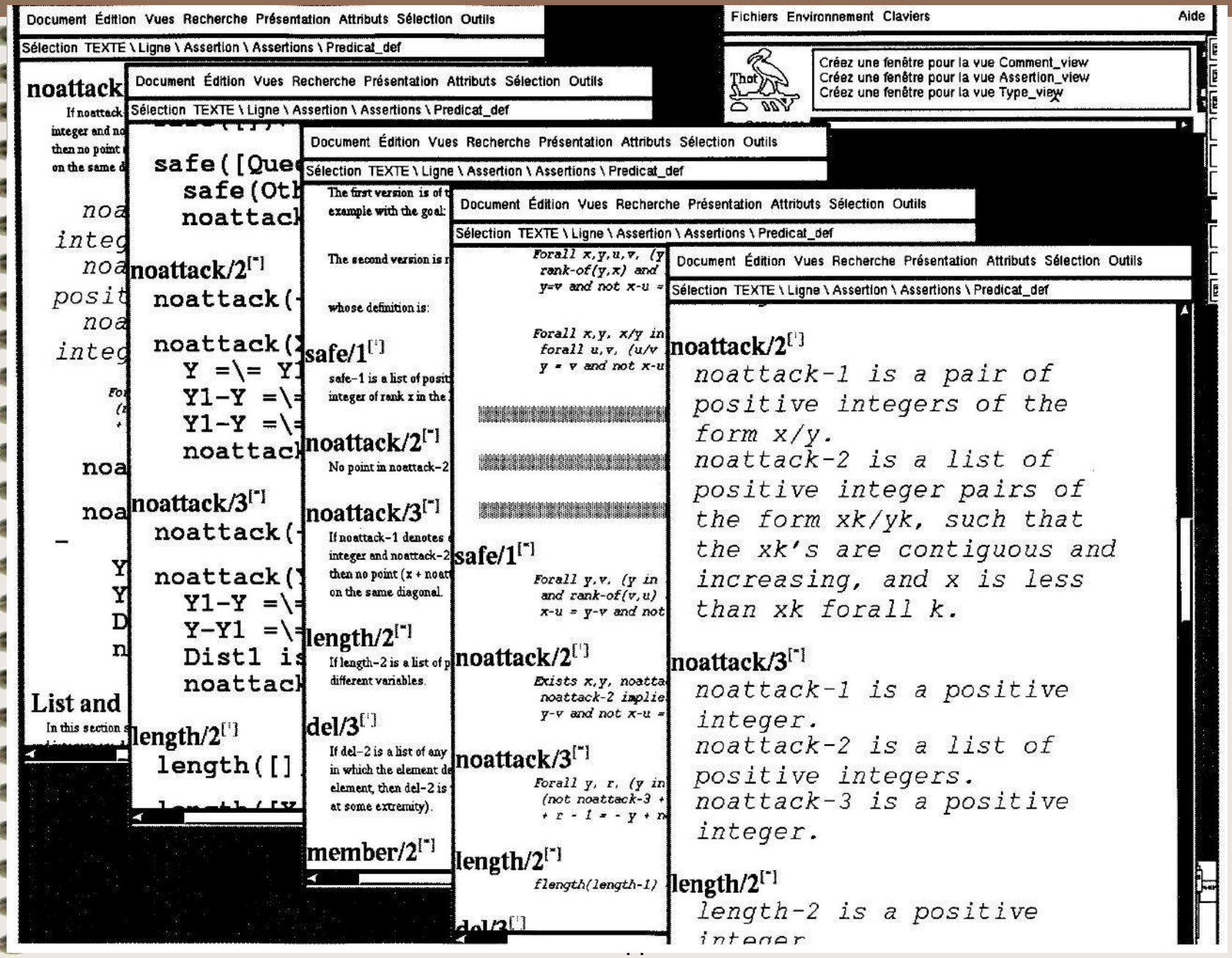
AbdelAli Ed-Dbali, Pierre Deransart, Mariza Andrade da Silva Bigonha, José de Siqueira, Roberto da Silva Bigonha:
HyperPro - An integrated documentation environment for CLP. WLPE 2001
Autres participants: Touraïvane (PrologIA, 1994), Patrick Parot (1995), Khalid El Khorchi (2002)

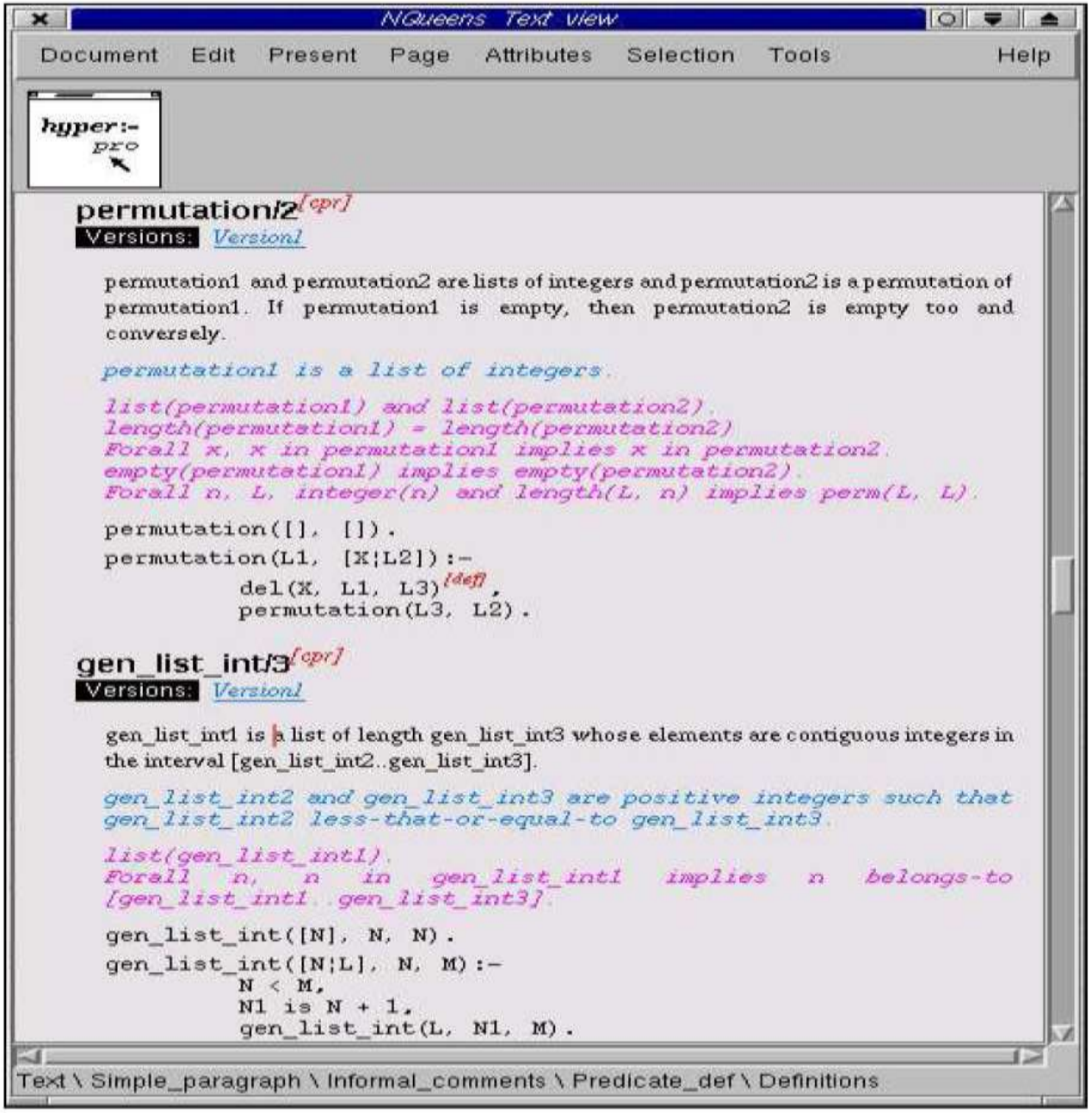
HyperPro: Literate CLP

Functionalities:

- Main view: exportable text (html, LaTeX, …)
- Document projections as synchronized views (including versions)
- TOC, Indexes
- Informal comments
- Formal comments (specifications, type assertions, modes)
- Codes (different versions)
- Applications (local evaluation, spell checker, theorem prover)

Idea: use existing document handling system


Document Édition Vues Recherche Présentation Attributs Sélection Outils
Sélection TEXTE \ Ligne \ Assertion \ Assertions \ Predicat_def
Fichiers Environnement Claviers Aide
Thot
Créez une fenêtre pour la vue Comment_view
Créez une fenêtre pour la vue Assertion_view
Créez une fenêtre pour la vue Type_view
noattack/2[*]
noattack-1 is a pair of positive integers of the form x/y.
noattack-2 is a list of positive integer pairs of the form xk/yk, such that the xk's are contiguous and increasing, and x is less than xk forall k.
noattack/3[*]
noattack-1 is a positive integer.
noattack-2 is a list of positive integers.
noattack-3 is a positive integer.
length/2[*]
length-2 is a positive

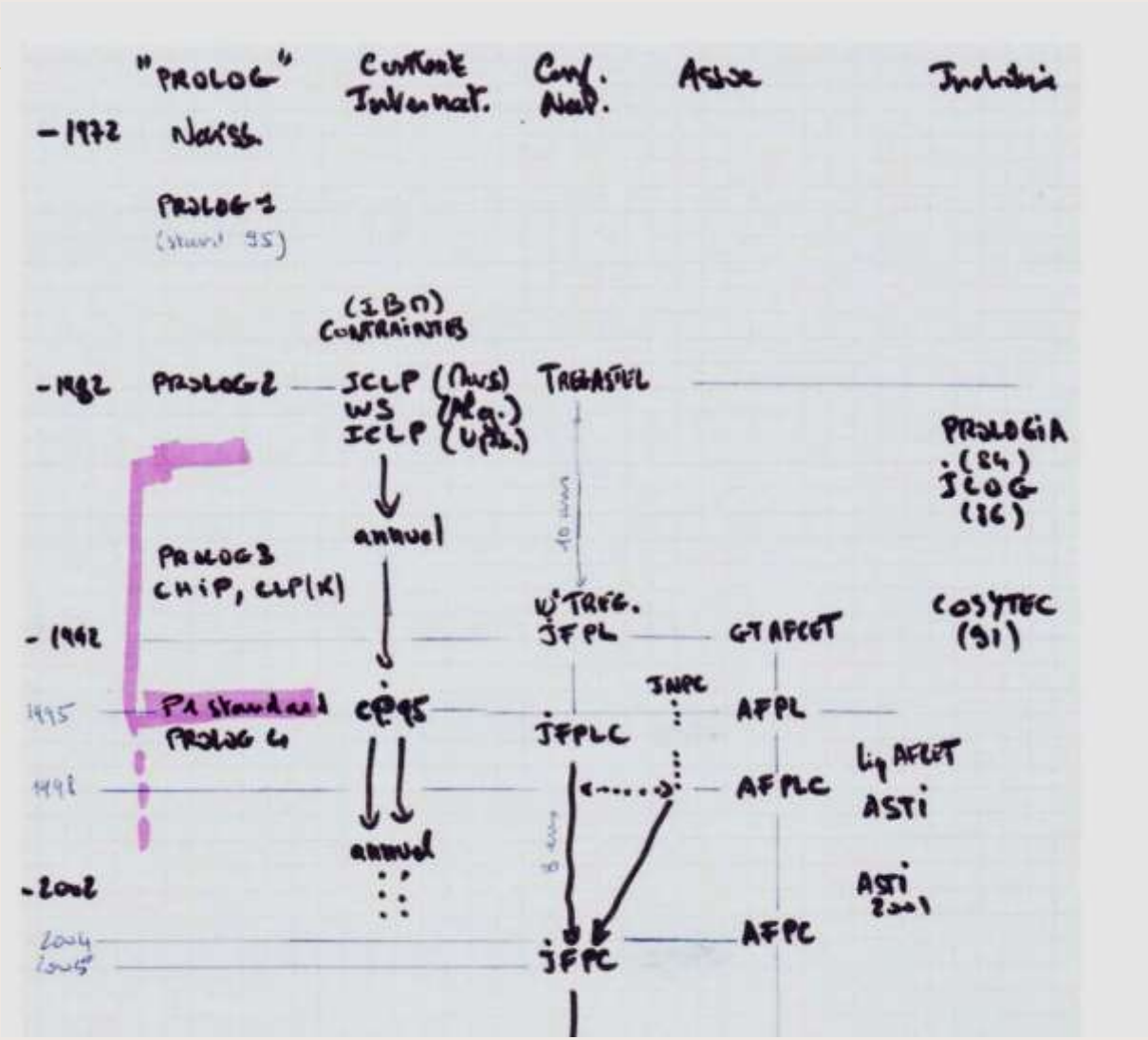

NQueens Text view
Document Edit Present Page Attributes Selection Tools Help
hyper:-
pro
permutation/2 [cpr]
Versions: Version1
permutation1 and permutation2 are lists of integers and permutation2 is a permutation of permutation1. If permutation1 is empty, then permutation2 is empty too and conversely.
permutation1 is a list of integers.
list(permutation1) and list(permutation2).
length(permutation1) = length(permutation2)
Forall x, x in permutation1 implies x in permutation2.
empty(permutation1) implies empty(permutation2).
Forall n, L, integer(n) and length(L, n) implies perm(L, L).
permutation([], []).
permutation(L1, [X|L2]):-
del(X, L1, L3) [def],
permutation(L3, L2).
gen_list_int/3 [cpr]
Versions: Version1
gen_list_int1 is a list of length gen_list_int3 whose elements are contiguous integers in the interval [gen_list_int2..gen_list_int3].
gen_list_int2 and gen_list_int3 are positive integers such that gen_list_int2 less-that-or-equal-to gen_list_int3.
list(gen_list_int1).
Forall n, n in gen_list_int1 implies n belongs-to [gen_list_int1..gen_list_int3].
gen_list_int([N], N, N).
gen_list_int([N|L], N, M):-
N < M,
N1 is N + 1,
gen_list_int(L, N1, M).
Text \ Simple_paragraph \ Informal_comments \ Predicate_def \ Definitions

Prolog:

La SAGA de la standardisation

Un standard qui tient bon…

Histoire, 50 ans de Prolog



	1985	London	BSI (UK)
	1987	Marseille, Paris	AFNOR (F)
May	1988	Oxford	ISO
Oct	1988	Grassau	DIN (G)
Apr	1989	Paris	
Nov	1989	Ottawa	
Apr	1990	Vienna	ANSI (US)
Dec	1990	Budapest	
Jun	1991	Paris	
Dec	1991	Leuven	
Jun	1992	Tokyo	
Nov	1992	Copenhagen	
Jul	1993	BALLOT	CD
Aug	1993	Boston	
Apr	1994	London	
Apr	1994	BALLOT	DIS
Dec	1994	Utrecht	
May	1995		IS

WG 17 produced more than 138 documents.

AKNOWLEDGMENTS

- Colleagues in the standardisation group, and particularly M. Van Caneghem, P. Chan, E. Fermaut, G. Narboni, J.-F. Pique in the French AFNOR panel

  and

  C. Bierle, E. Boerger, D. Bowen, M. Carlsson, B. Demoen, K. Dässler, A. Dodd, J. Hodgson, L. Jensen, C.D.S. Moss, K. Nakamura, T. Mansfield, U. Neumerkel, N. North, C. Pichler, S. Szpakowicz, G. Thwaites, A. Vellino in the ISO working group and R. Scowen, its convener.

  Only those who attended many meetings are cited here.

Le staff

Michel Van Caneghem

Le staff

Bataille pour une spécification formelle

WIM (W , [W | E]).

WIM (W , [V | U]) ⇐ WIM (W , U)

Do you pick up the idea ? ...

WIM (A , L) : if L is a list
then A is a member of L.

MEMBER (A , [A | L])

MEMBER (A , [B | L]) ⇐ MEMBER (A , L)

The axioms are easy to read if the proofs of partial correctness and completness of the idea are easy to perform.

La spécification formelle du standard

WG17,
Mai 1991

4.3. THE FORMAL SPECIFICATION

☞ **semantics**(P, G, T - N) then *if* P is a program and G a goal *then* T is a partial visited search-tree up to leaf node N.

semantics(P, G, T) ← **buildtree**(P,mk-tree(node(nil,root), node(zero.nil,G_1).nil) - node(zero.nil,G1), T),
flagcut (G, zero.nil, G_1)

☞ **buildtree**(P, T_1 - N_1, T_2 - N_2) then *if* P is a program and T_1 is a partial visited search-tree for P and the root goal of T_1 up to the leaf node N_1 *then* T_2 is the extension of T_1 up to the leaf node N_2 or up to the root if a finite visited search-tree exists.

buildtree(P, T - N, T - N) ←

buildtree(P, T_1 - N_1, T_2) ← not **root_node**(N_1),
treatment(P, T_1 - N_1, T_3 - N_3),
c_choice(T_3 - N_3, N_4),
buildtree(P, T_3 - N_4, T_2)

root_node(node(nil, root)) ←

☞ **c_choice**(T - N1, N2) then *if* T is a partial visited search-tree and N_1 a current node *then* N_2 is the next visited leaf following the visit order of the search-trees or the root if none exists.

NOTE: a "current node" has children leaves only or is a leaf and it has, as all its ancestors, just brother leaves.

c_choice(T, N) ← **first_child**(T, N)

c_choice(T, N) ← not **has_a_child**(T), **brother**(T, N)

c_choice(T, N) ← not **has_a_child**(T),
not **has_a_brother**(T),
re_c_choice(T, N)

re_c_choice(T, N) ← **brother**(T, N)

re_c_choice(T - N_1, N_2) ← not **has_a_brother**(T - N_1),
parent(T - N_1, N_3),
re_c_choice(T - N_3, N_2)

re_c_choice(T - N, N) ← not **has_a_brother**(T - N)

La spécification formelle du standard

WG17,
Mai 1991

5. Elements of validation of the specification

The validation task is a part of the specification work which is hidden to the readers. It is however a necessary step towards the correctness of the formal specification.

We present four aspects of the validation.

1) Verification of the stratification. (This gives the foundation of the actual semantics).

2) Partial proof of correctness and completeness of the specification w.r.t. the comments. (This helps to give a better form to the comments and to detect design errors).

3) Proof of some properties. (This helps to write the specification in a clearer way and to detect design errors).

4) Tests using the executable specification. (This helps to detect design errors).

Bataille pour une spécification formelle

ILPS 1991

THE FORMAL SPECIFICATION OF STANDARD PROLOG

-ISO SC22 WG17-

P. DERANSART	INRIA-Rocquencourt
G. FERRAND	INRIA, University of Orleans
A. ED.DBALI	INRIA, University of Orléans
S. RENAULT	INRIA-Rocquencourt
G. RICHARD	University of Orléans

Now :

- We have :
  - a full draft
  - 5 years of experience.
- Other candidates :
  - Pascal like (R.O'Keefe)
  - Denotational (N. North)
  - Evolving Algebras (E. Börger).

Bataille pour une spécification formelle

THE PROPOSALS

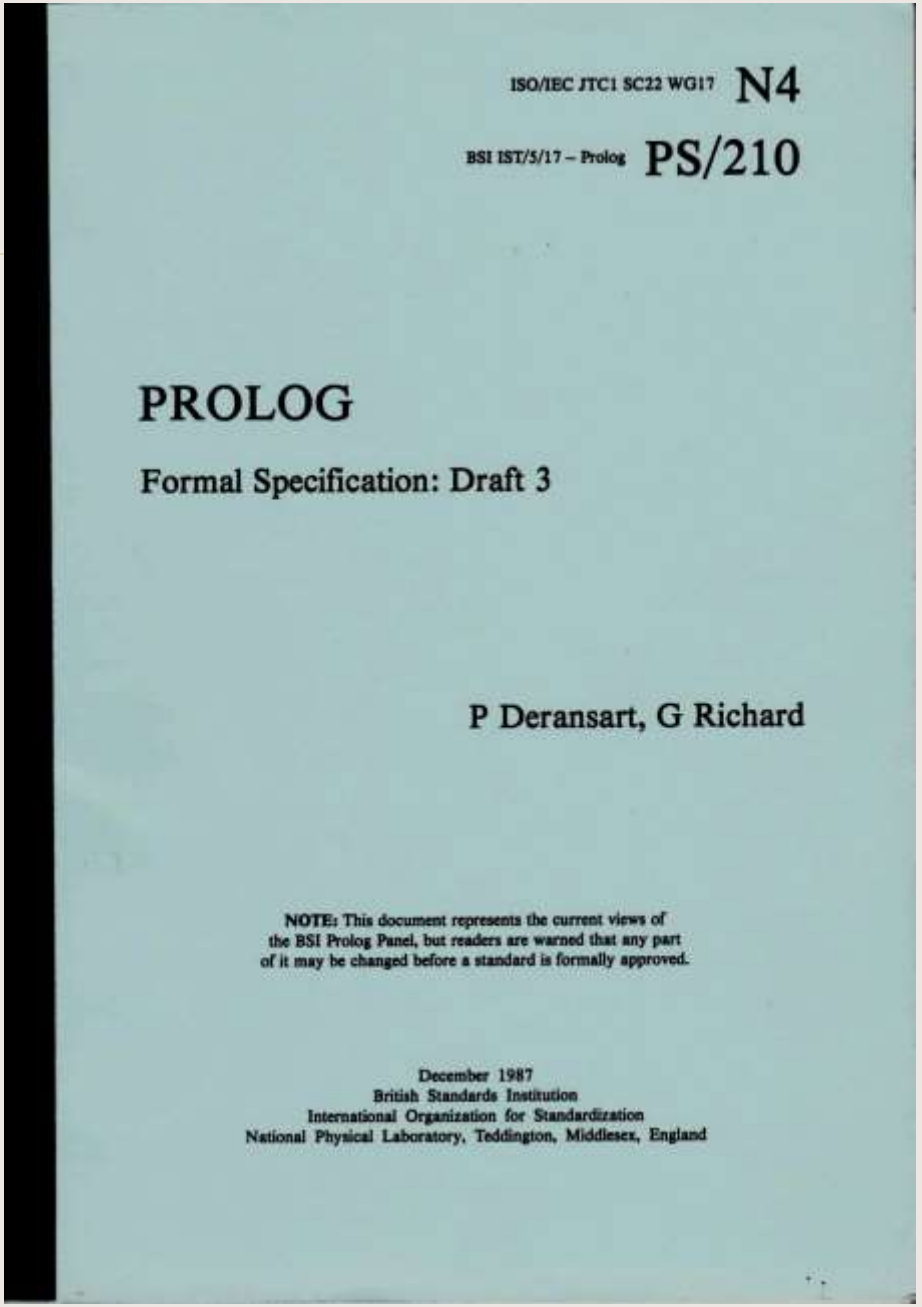
O'Keefe's description :
- low level (Pascal like)
- operational

Denotational semantics (N. North) :
- synthetic, declarative
- many comments
- need to know D.S.
- exec. spec in MIRANDA

Evolving Algebras (E. Börger) :
- simple algebraic formalism
- operational

Logic Programming (P. Deransart, G. Ferrand, A. Ed-Dbali) :
- declarative
- form. spec in a sublanguage
- exec. spec in the same language

Prolog
Formal Specification
(1987)



ISO/IEC JTC1 SC22 WG17 N4

BSI IST/5/17 – Prolog PS/210

PROLOG

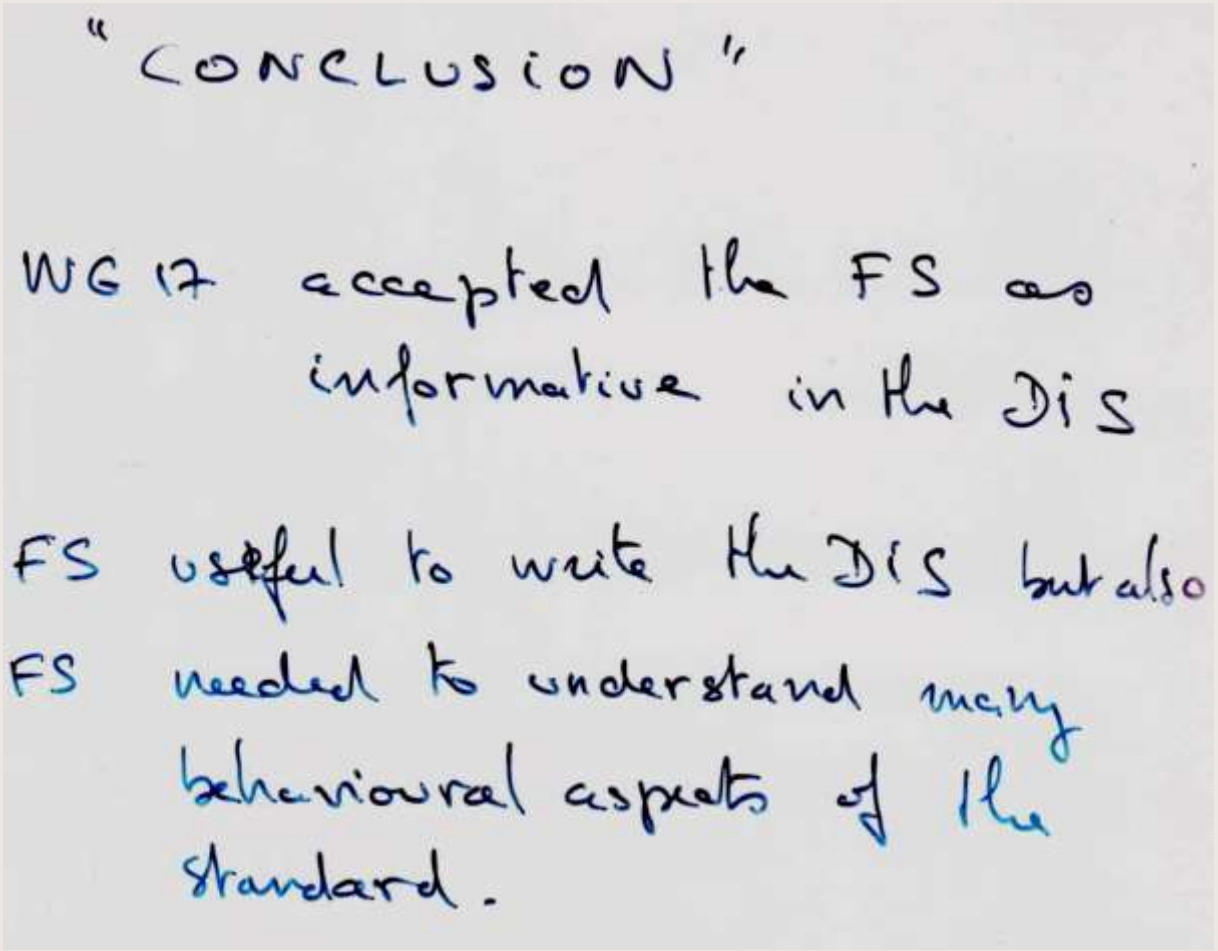
Formal Specification: Draft 3

P Deransart, G Richard

NOTE: This document represents the current views of the BSI Prolog Panel, but readers are warned that any part of it may be changed before a standard is formally approved.

December 1987
British Standards Institution
International Organization for Standardization
National Physical Laboratory, Teddington, Middlesex, England

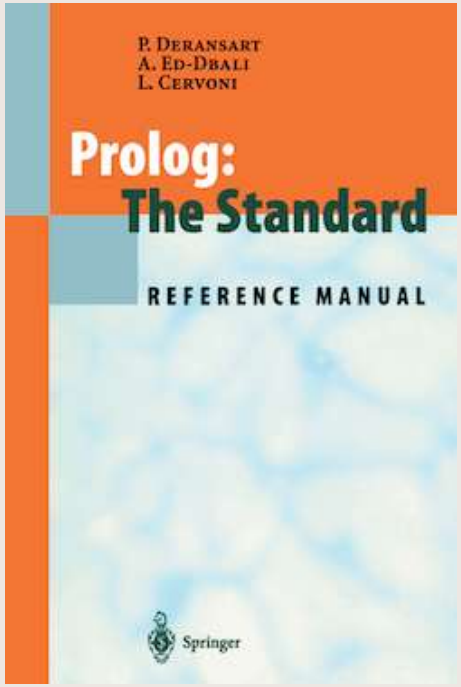
Bataille pour une spécification formelle DIS mars 1993



Conformity Testing I: Syntax

#	Query	13211-1:1995 ²: Cor.2, ³: Cor.3	IF V5.1B	SWI 6.3.18-3*	YAP 6.3.4	B 7.8#5	GNU 1.4.5	SICStus 4.3.3	Minerva 2.4	XSB 3.7 rev.8750	Ciao 1.20.0	IV 1.4.2	ECLiPSe
	number of conforming queries	306/309	243	219	222	225	306	282	217	196	232	261	307
	recent improvements	8+16		28	58	62	44	52		59	21		new
	recent regressions			1	2	1				11	2		
	misinterpretations	1	27	42	28	23	0	0	26	40	27	18	0
	rejections	2	15	14	15	24	0	0	28	21	4	10	0
	write syntax deviations	0	3	3	7	6	1	0	19	15	4	2	0
	extensions		21	31	37	31	2	27	20	37	42	18	2
	read/1 syntax OK	#286	#203	#56	#49	#111	yes	yes new	#151	#251	#197	#35	yes
	write_canonical/1 OK	yes	#18	#227	#227	#172	yes	yes new	#18	#1, #172	yes new	yes	yes
	writeq/1 OK	yes ²	#133	#156	#150	#18	#260	yes new	#135	#1	223	#135	yes
1	writeq('\n').	'\n'	OK	OK	OK	OK	OK	OK	OK	'	OK new	OK	OK
2	'	syntax err.	OK	waits	waits	waits	OK	waits	OK	waits	OK new	waits	OK
3	)	waits	sy._e.(..)	OK	OK	OK	OK	OK	OK	OK	OK	OK	sy._e.(..)
261	) '.	syntax err.	OK	waits	waits	waits	waits	waits	OK	waits	OK new	waits	OK
4	.	syntax err.	OK	OK	OK	OK	OK new	OK	OK	OK	OK	OK	OK
5	writeq(' '). % horiz. tab	syntax err.	'\t'	'\t'	'\t'	'\t'	OK new	OK	'\t'	' '	'\t' new	'\t'	OK
177	0'\t=0' . % horiz. tab	syntax err.	succeeds	succeeds	succeeds	succeeds	OK new	OK	succeeds	succeeds new	succeeds	succeeds	OK
6	writeq(' ').	syntax err.	OK	'\n'	'\n'	'\n'	OK	OK	OK	'	OK new	OK	OK
7	writeq('\ '). % "\\n"	''	OK	OK new	OK new	OK new	OK	OK	sy._e.(..)	OK	OK	OK	OK
8	writeq('\ a'). % "\\na"	a	OK	OK	OK new	OK new	OK	OK	sy._e.(..)	OK	OK	OK	OK
9	writeq('a\ b'). % "a\\nb"	ab	OK	OK	OK	OK new	OK	OK	sy._e.(..)	OK	OK	OK	OK
10	writeq('a\ b'). % "a\\n b"	'a b'	OK	ab	OK new	OK new	OK	OK	sy._e.(..)	OK new	OK	OK	OK
11	writeq('\ ').	syntax err.	OK	OK new	OK new	OK new	OK	OK	OK	'\\ ' new	''	'\\ '	OK
193	writeq('\ '). % "\\ \n"	syntax err.	OK	OK new	OK new	OK new	OK	OK	OK	'\\ '	OK new	OK	OK
12	writeq('\ '). % "\\\t"	syntax err.	OK	OK new	OK new	OK new	OK	OK	OK	'\\ '	''	'\\\t'	OK
13	writeq('\t').	'\t'	OK	OK	OK	OK	OK	OK	OK	' '	OK new	OK	OK
14	writeq('\a').	e.g. '\a'	OK	OK	OK	OK	OK	OK	'\u0007'	'^G'	OK new	OK	OK
15	writeq('\7').	e.g. '\a'	OK	OK	OK new	OK	OK	OK	'\u0007'	'^G'	OK new	OK	OK
16	writeq('\ca').	syntax err.	OK	a	a new	OK new	OK	OK	OK	'\\ca' new	a	'\\ca'	OK
241	writeq('\d').	syntax err.	OK	OK	'\177\' new	OK	OK	'\x7F\' new	OK	'\\d' new	'\177\' new	'\\d'	OK
17	writeq('\e').	syntax err.	'\e'	'\033\'	'\033\' new	OK new	OK	'\x1B\' new	'\u001b'	'\\e' new	'\033\' new	'\\e'	OK
18	writeq('\033\').	e.g. '\33\' or repr. err.	'\e'	'\033\'	'\033\' new	'^['	'\x1b\'	'\x1B\' new	'\u001b'	'^['	'\033\'	'\x1b\'	'\33\'
301	writeq('\0\').	e.g. '\0\' or repr. err.	syn. e.	'\000\'	''	'	err.	'\x0\'	'\u0000'	''	''	''	'\0\'
19	char_code('\e',C).	syntax err.	C = 27	C = 27	C = 27	OK new	OK	C = 27	C = 27	t._e.(c,..) new	C = 27	t._e.(c,..)	OK
21	char_code('\d',C).	syntax err.	OK	OK new	C = 127	OK new	OK	C = 127	OK	t._e.(c,..) new	C = 127	t._e.(c,..)	OK
22	writeq('\u1').	syntax err.	OK	OK	OK new	OK new	OK	OK	OK	'' new	'\001\'	'\\u1'	OK
23	X = 0'\u1.	syntax err.	OK	OK new	OK new	OK	OK	OK	OK	X = 1 reg	X = 1 reg	OK	OK
24	writeq('	syntax err.	OK	waits	waits	waits	OK new	waits	OK	waits	OK new	waits	OK
25	writeq(.	syntax err.	OK	OK	OK	OK	OK new	OK	OK	OK	OK	OK	OK

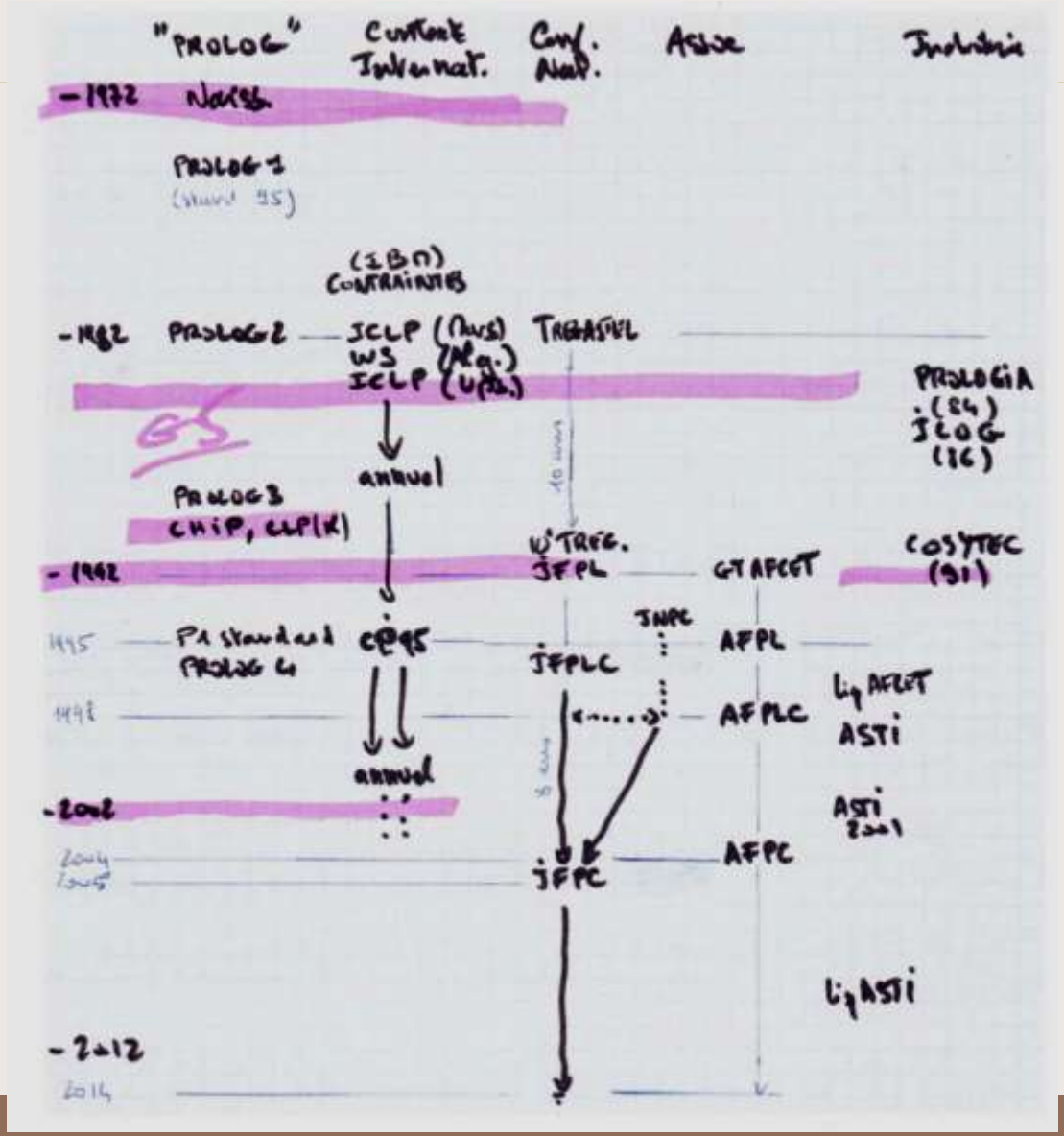
https://eddbali.net/errata.html



Pierre Deransart
AbdelAli Ed-Dbali

Laurent Cervoni

Histoire, 50 ans de Prolog
Grandes étapes



Repères historiques

1962 les prémisses

1972 PROLOG

Prolog I
Prolog II
Prolog III

1982 IA 5ème génération computers

1992 Le tournant des contraintes

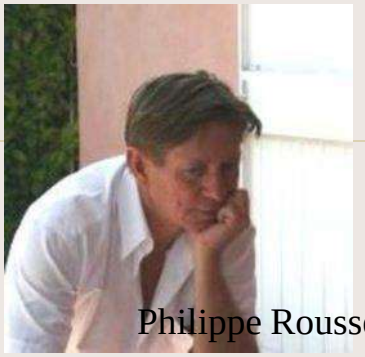
Prolog IV

2002 …

2022 Future?

Histoire, 50 ans de Prolog
Equipe de Marseille

A. Colmerauer (2000)



Philippe Roussel

Henry Kanoui

1972 « naissance »

1975 Prolog I → standard

1982 Prolog II (unif equationnelle)

1984 Prologia — Geneviève Bossu, Henri Garetta, Henri Kanoui, Robert Pasero, **Jean-Francois Pique** and **Michel van Caneghem**

1989 Prolog III, concept de « PL avec Contraintes » seront Prolog III, CLP(R) et CHIP

1996 Prolog IV approximation des contraintes non-linéaires

Equipe de Marseille

1972 **Naissance** de « Prolog » selon A. Colmerauer

1975 **Prolog I** (programmation avec clauses de Horn et coupure, celui qui sera standardisé plus tard)

1982 **Prolog II** : unification équationnelle, termes infinis (premières contraintes)

1989 **Prolog III** : concept de « Programmation Logique avec Contraintes » (PLC). Aux US : CLP(R), en Europe : CHIP.

1995 Publication d'un **standard** pour Prolog I après 10 ans d'élaboration au sein de l'AFNOR puis de l'ISO.

1996 **Prolog IV** : introduction traitement de contraintes non-linéaires par approximations

« The Birth of Prolog », Alain Colmerauer & Philippe Roussel, GIA, Luminy, Nov 1992

PL, PLC, PC la francophonie

1982-1991 Perros-Guirrec (2 fois) puis **Trégastel** (8 fois): « Séminaires annuels de Programmation en Logique de Trégastel », **Mehmet Dincbas, Jean-Pierre Le Pape, Serge Bourgault** (CNET Lannion).

1992-1994 **JFPL** période AFCET (cf colonne « Assoc »)

1995 **JFPLC** modification du nom de la conférence (le « **C** » pour « Contraintes » est ajouté)

2000 aux JFPLC de Marseille tentative de fusion des communautés JFPLC et JNPC qui échoue

2004 fusion JFPLC et JNPC et renommage en **JFPC**, (l'AFPLC devient AFPC et accueille toute la communauté (21 juin 2004))

2014 : 10èmes JFPC …

Histoire, 50 ans de Prolog

1982-1991 Perros-Guirrec (2 ans) puis Trégastel :
« Séminaires annuels de Programmation en Logique de Trégastel »,
M. Dincbas, J.-P. Le Pape, S. Bourgault (CNET Lannion).

Trégastel 1985

A l'international…

1980 IBM – Contraintes

1982 1ère International Conference on L ogic Programming (ICLP) à Marseille

1983 Logic Programming Workshop '83 : Praia da Falesia, Algarve(P)

1984 2ème ICLP, Upsala (Se). Les conférences ICLP deviennent alors annuelles

1990 1rst North American Conference on Logic Programming (NACLP)

1991 9ème ICLP à Paris (cocktail à a tour Eiffel). Plus de 600 participants et exposition avec plus de 15 « industriels ».

1993 NACLP devient ILPS (International Logic Programming Conference)

1995 1rst Principles and Practice of Constraint Programming Conference (CP '95), Cassis (F)

…Les deux conférences (ICLP et CP) évoluent annuellement en parallèle.

1998 ILPS se « fond » dans ICLP : JICSLP, puis une seule conférence internationale : ICLP

2004 ICLP revient en France à St Malo

ICLP et CP continuent à être organisées chaque année.

2006 12th International Conference Principles and Practice of Constraint Programming - CP 2006 à Nantes

ICLP 82

PROCEEDINGS

of the

FIRST INTERNATIONAL

LOGIC

PROGRAMMING

CONFERENCE

Organized by : Groupe Intelligence Artificielle
et Association pour la Diffusion
et le Développement de Prolog

Sponsored by : Agence de l'Informatique
Faculté des Sciences de Luminy

Edited by : Michel Van Caneghem

September, 14-17th, 1982
Faculté des Sciences de Luminy
Marseille, France

ICLP 82

GENERAL CHAIRMAN

Alain COLMERAUER
Groupe d'intelligence artificielle
Case 901
Faculte des Sciences de Luminy
13288 MARSEILLE Cedex 9
FRANCE

PROGRAM COMITTEE
AND
REFEREES

Kenneth BOWEN
School of Computer and
Information Science
Syracuse University
SYRACUSE NY 13210
USA

Maurice BRUYNOOGHE
Afdeling Toegepaste
Wiskunde en Programmatie
Katholieke Universiteit Leuven
Celestijnenlaan 200 A
B - 3030 HAVERLEE BELGIQUE

Keith CLARK
Department of Computing
Imperial College of Science & technology
180 Queen's gate
LONDON SW7 2BZ
ENGLAND

Maarten van EMDEN
Department of Computer Science
University of Waterloo
WATERLOO ONTARIO N2L 3G1
CANADA

Herve GALLAIRE
Laboratoire de Marcoussis
Centre de recherche de la CGE
Route de Nozay
91460 MARCOUSSIS

Robert KOWALSKI
Department of Computing
Imperial College of Science & technology
180 Queen's gate
LONDON SW7 2BZ
ENGLAND

Luis Moniz PEREIRA
Departamento de Informatica
Universidade Nova de Lisboa
1899 LISBOA Codex
PORTUGAL

J.A. ROBINSON
School of Computer and
Information Science
Syracuse University
SYRACUSE NY 13210
USA

Philippe ROUSSEL
Departamento de Matematicas
Universidad Simon Bolivar
Apartado Postal 80659
CARACAS
VENEZUELA

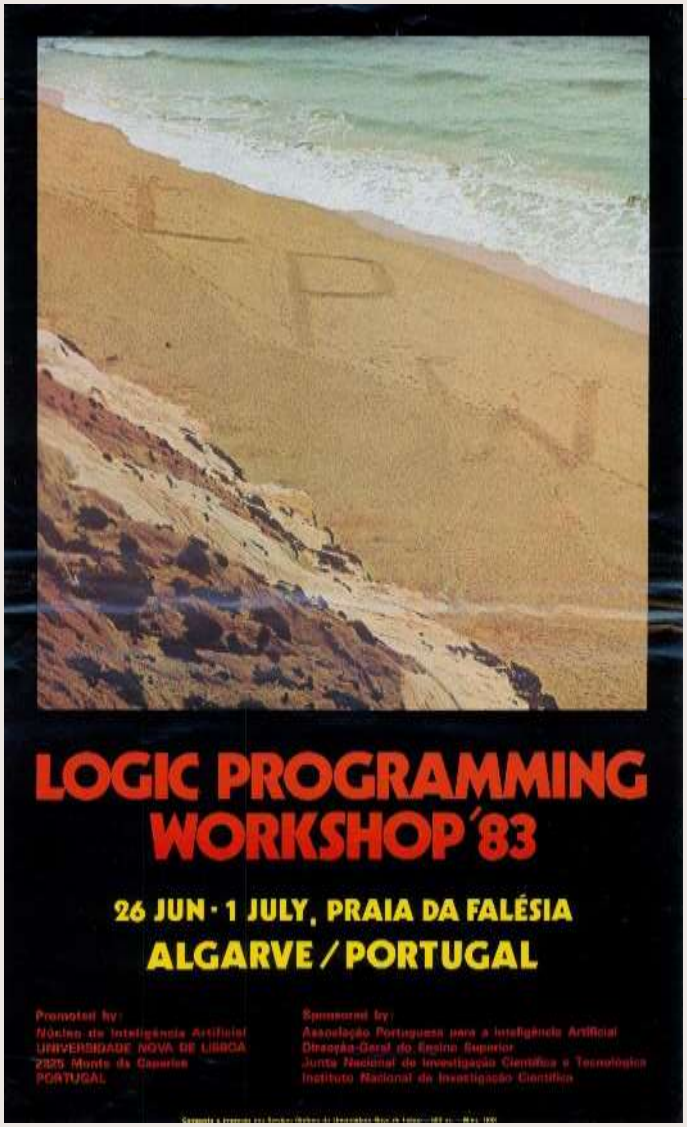
Peter SZEREDI
SZKI
P.O.B. 224
BUDAPEST H-1368
HONGRIE

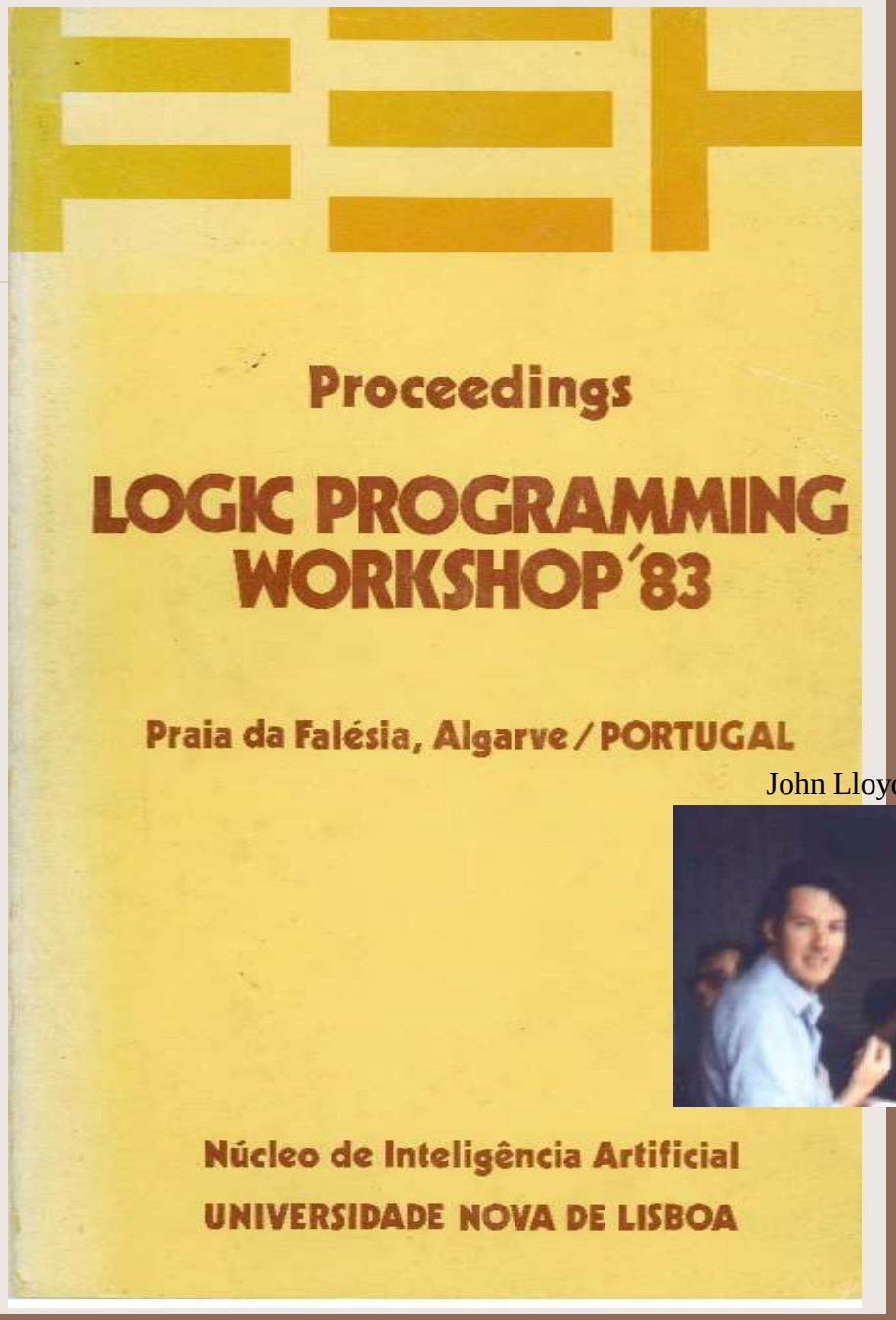
Sten-Ake TARNLUND
UPMAIL
Uppsala University
Computing Science Department POB 2059
S-750 02 UPPSALA
SUEDE

Copies are available for $30, payment
in advance, bookrate; add postal tax.
Make checks payable to ADDP.
Order from:

ADDP-GIA
Case 901
Faculte des Sciences de Luminy
13288 MARSEILLE Cedex 9
FRANCE

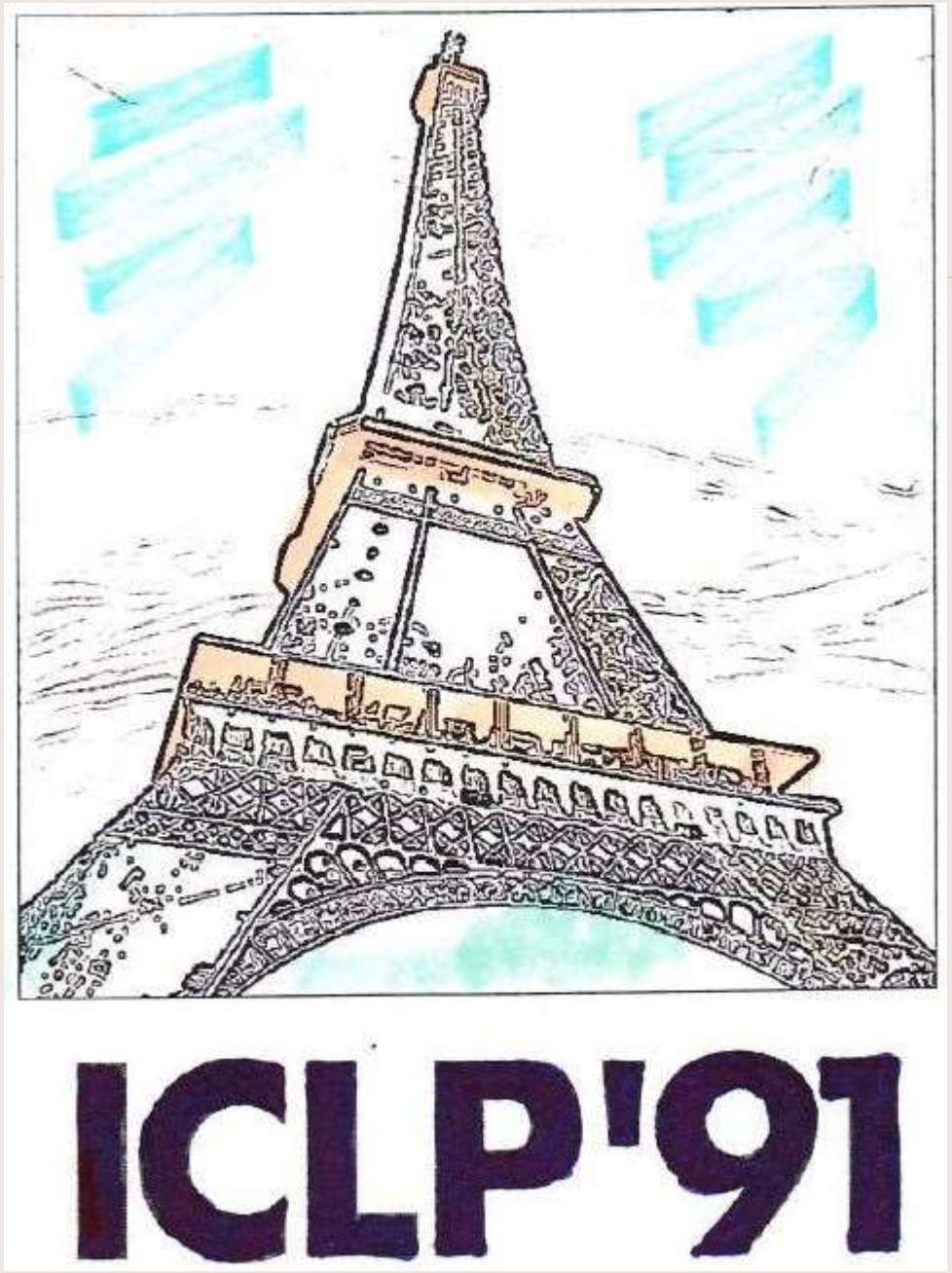
LPW 83





John Lloyd

ICLIP 91
Paris



ICLP 91
Avenir de la programmation en logique

Conferences on Logic Programming (ICLP)

J. Cohen, A. Robinson, K. Furukawa (1996)

Conferences on (PP of) Constraints Programming (CP)

5ième génération

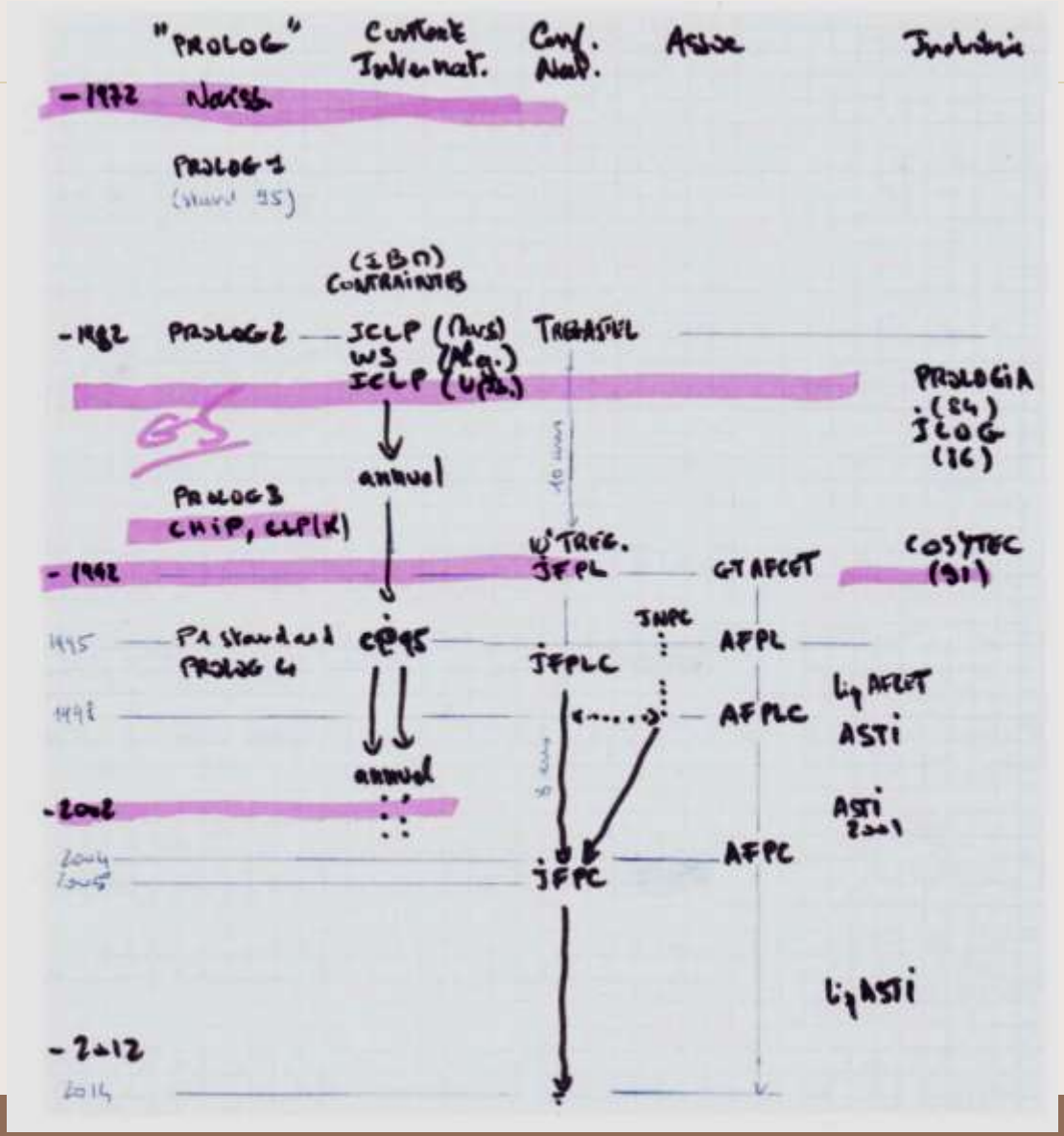
MARSEILLE-FRANCE

L'inventeur du Prolog trouve une suite au Japon

Nul n'est prophète en son pays. Pas même Alain Colmerauer, inventeur de « Prolog », un langage destiné à l'« Intelligence Artificielle ». Professeur à l'université d'Aix-Marseille, il a dû attendre dix ans avant de voir le fruit de ses recherches massivement exploité... par les Japonais.

«P

Histoire, 50 ans de Prolog
Grandes étapes



ECRC

1984-1999?

ECRC GmbH, Arabellastrasse 17, D-81925 Munich, Germany

L'ECRC a été fondé en 1984 en tant qu'institut de recherche par les sociétés Bull (France), ICL (UK) and Siemens (Germany).

ECRC Network Services GmbH a ensuite émergé de cet institut et a été racheté par Cable & Wireless le 4 janvier 1999.[1]

La société a été initialement intégrée au groupe en tant que filiale, Cable & Wireless ECRC GmbH, et a été complètement absorbée par le groupe Cable & Wireless en 2004 (date inconnue)

H. Gallaire (2004)

Hervé Gallaire, directeur de la recherche et de la technologie de Xerox (bureau du futur)

PROLOG
DINOSAUR OR MAMMALIAN?

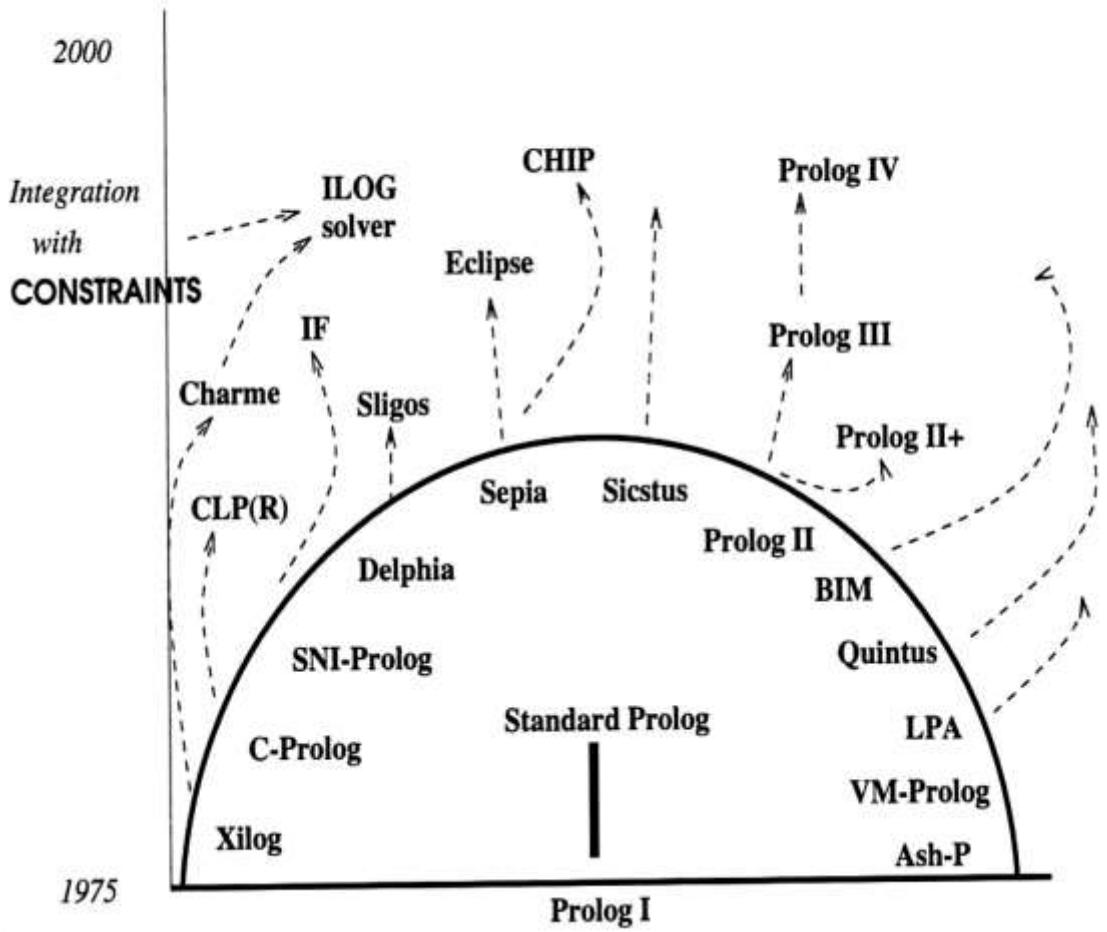
PAP'96, Tutorial

Pierre DERANSART
INRIA-Rocquencourt
Pierre.Deransar@inria.fr

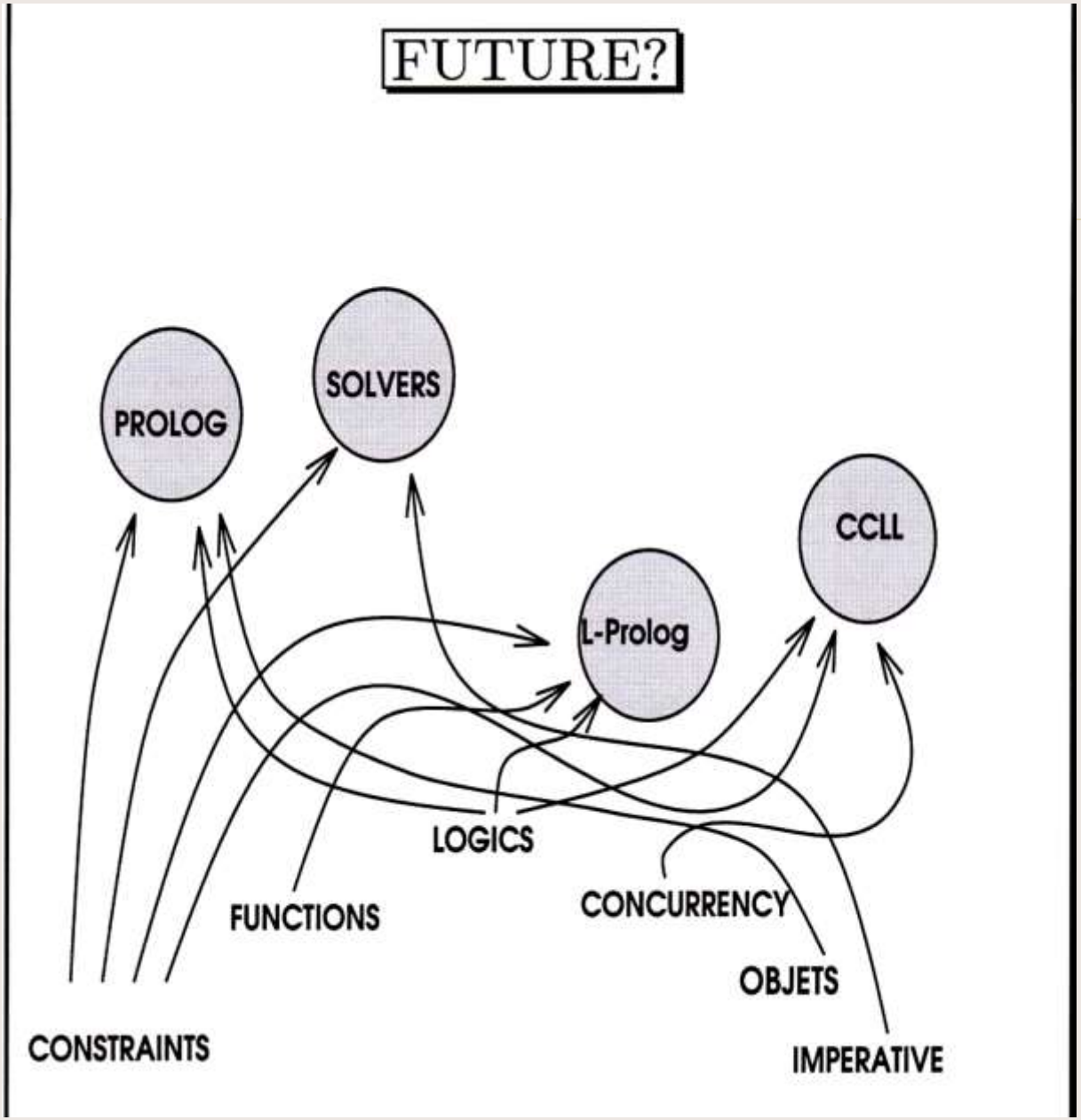
AbdelAli ED-DBALI
University of Orléans

April 23, 1996

Genealogy


2000
Integration
with
CONSTRAINTS
ILOG
solver
CHIP
Prolog IV
Eclipse
IF
Prolog III
Charme
Sligos
Prolog II+
CLP(R)
Sepia
Sicstus
Prolog II
Delphia
BIM
SNI-Prolog
Quintus
Standard Prolog
LPA
C-Prolog
VM-Prolog
Xilog
Ash-P
1975
Prolog I

FUTURE?


PROLOG
SOLVERS
CCLL
L-Prolog
LOGICS
FUNCTIONS
CONCURRENCY
OBJETS
CONSTRAINTS
IMPERATIVE

ISO Prolog

Agenda WG17 meeting 2022

1. Greeting and introduction of participants, administrativa
1.1 Minutes of 2021 meeting N284
1.2 Recent progress in Prolog systems
1.3 Prolog 50
1.4 Prolog archive

2. Further corrigenda

3. Prolog prologue

4. DCG progress

5. Interest for real meeting

Basic sources

Fifty Years of Prolog and Beyond *

PHILIPP KO¨ RNER, MICHAEL LEUSCHEL
Institut für Informatik, Universität Du¨sseldorf, Universit¨atsstr. 1, D-40225 Du¨sseldorf (*e-mail:* {p.koerner,leuschel} @uni-duesseldorf.de)
JOA˜O BARBOSA, V´ITOR SANTOS COSTA
Department of Computer Science, Faculty of Science of the University of Porto (*e-mail:* {joao.barbosa,vscosta} @fc.up.pt)
VERO´ NICA DAHL
Computing Sciences Department, Simon Fraser University (*e-mail:* veronica dahl@sfu.ca)
MANUEL V. HERMENEGILDO, JOSE F. MORALES
IMDEA Software Institute and Universidad Polit´ecnica de Madrid (UPM) (*e-mail:* {manuel.hermenegildo,josef.morales} @imdea.org)
JAN WIELEMAKER
Centrum voor Wiskunder en Informatica (CWI), Amsterdam (*e-mail:* J.Wielemaker@cwi.nl)
DANIEL DIAZ
Centre de Recherche en Informatique, University Paris-1 (*e-mail:* daniel.diaz@univ-paris1.fr)
SALVADOR ABREU
NOVA-LINCS, University of E´vora (*e-mail:* spa@uevora.pt)
GIOVANNI CIATTO
Dept. of Computer Science and Engineering, Alma Mater Studiorum—Univerit`a di Bologna (*e-mail:* giovanni.ciatto@unibo.it)

Strengths, Weaknesses, Opportunities, and Threats – **SWOT**

Forces, vos Faiblesses, ainsi que les Opportunités et Menaces du marché

Strengths (Section 4.1)

• clean, simple language syntax and semantics
• immutable persistent data structures, with "declarative" pointers (logic variables)
• arbitrary precision arithmetic
• safety (garbage collection, no NullPointer exceptions, ...)
• tail-recursion and last-call optimization
• efficient inference, pattern matching, and unification, DCGs
• meta-programming, programs as data
• constraint solving (3.3.2), independence of the selection rule (coroutines (3.3.3))
• indexing (3.3.3), efficient tabling (3.3.3)
• fast development, REPL (Read, Execute, Print, Loop), debugging (3.3.4)

• commercial (2.10.1) and open-source systems
• active developer community with constant new implementations, features, etc.
• sophisticated tools: analyzers, partial evaluators, parallelizers, ...
• successful applications
— program analysis,
— domain-specific languages
— heterogeneous data integration
— natural language processing
— efficient inference (expert systems, theorem provers), symbolic AI
• many books, courses and learning materials

Strengths, Weaknesses, Opportunities, and Threats – **SWOT**

Forces, vos Faiblesses, ainsi que les Opportunités et Menaces du marché

Forces (section 4.1)

• syntaxe et sémantique de langage propres et simples

• des structures de données persistantes immuables, avec des pointeurs « déclaratifs » (variables logiques)

• arithmétique de précision arbitraire

• sécurité (garbage collection, pas d'exceptions NullPointer, ...)

• optimisation de la récursivité terminale et du dernier appel

• inférence efficace, correspondance de modèles et unification, DCG

• méta-programmation, programmes en tant que données

• résolution de contraintes (3.3.2), indépendance de la règle de sélection (coroutines (3.3.3))

• indexation (3.3.3), classement efficace (3.3.3)

• développement rapide, REPL (Read, Execute, Print, Loop), débogage (3.3.4)

• systèmes commerciaux (2.10.1) et open source

• communauté de développeurs active avec de nouvelles implémentations, fonctionnalités, etc.

• des outils sophistiqués : analyseurs, évaluateurs partiels, paralléliseurs, ...

• les candidatures retenues

— analyse de programme,

— langues spécifiques à un domaine

— intégration de données hétérogènes

— traitement du langage naturel

— inférence efficace (systèmes experts, démonstrateurs de théorèmes), IA symbolique

• de nombreux livres, cours et supports d'apprentissage

Strengths, Weaknesses, Opportunities, and Threats – **SWOT**

Forces, vos Faiblesses, ainsi que les **Opportunités** et Menaces du marché

Opportunities (Section 4.2)

• new application areas, addressing societal challenges

4.2:

— neuro-Symbolic AI

— explainable AI, verifiable AI

— Big Data

• new features and developments

— probabilistic reasoning (3.5.1)

— embedding ASP (3.5.1) and SAT or SMT solving

— parallelism (2.7, 3.3.3) (resurrecting 80s, 90s re-search)

— full-fledged JIT compiler

Strengths, Weaknesses, Opportunities, and Threats – **SWOT**

Forces, vos Faiblesses, ainsi que les **Opportunités** et Menaces du marché

Opportunités (Section 4.2)

- de nouveaux domaines d'application, répondant à des enjeux sociétaux 4.2 :
  - — IA neuro-symbolique
  - — IA explicable, IA vérifiable
  - — Big Data
- nouvelles fonctionnalités et développements
  - — raisonnement probabiliste (3.5.1)
  - — intégration de la résolution ASP (3.5.1) et SAT ou SMT
  - — parallélisme (2.7, 3.3.3) (ressuscite les années 80, 90 chercher)
  - — compilateur JIT à part entière

Strengths, **Weaknesses**, Opportunities, and Threats – **SWOT**
Forces, vos **Faiblesses**, ainsi que les Opportunités et Menaces du marché

Weaknesses (Section 4.3)

- syntactically different from “traditional” programming languages, not a mainstream language
- learning curve, beginners can easily write programs that loop or consume a huge amount of resources
- static typing (see, however, 3.3.3)
- data hiding (see, however, 3.3.1)
- object orientation (see, however, 4.5.4)
- limited portability (see 4.5.1)
- packages: availability and management
- IDEs and development tools: limited capabilities in some areas (e.g., refactoring; 4.5.2)
- UI development (usually conducted in a foreign language via FLI (3.3.1))
- limited support for embedded or app development

Strengths, Weaknesses, Opportunities, and **Threats** – **SWOT**

Forces, vos Faiblesses, ainsi que les Opportunités et **Menaces du marché**

Faiblesses (Section 4.3)

- syntaxiquement différent de langages de programmation « traditionnels », pas un langage courant
- courbe d'apprentissage, les débutants peuvent facilement écrire des programmes qui bouclent ou consomment une énorme quantité de ressources
- typage statique (voir cependant 3.3.3)
- masquage des données (voir cependant 3.3.1)
- orientation objet (voir cependant 4.5.4)
- portabilité limitée (voir 4.5.1)
- modules : disponibilité et gestion
- IDEs et outils de développement : capacités limitées dans certains domaines (par exemple, refactoring ; 4.5.2)
- Développement d'interface utilisateur (généralement effectué dans une langue étrangère via FLI (3.3.1))
- prise en charge limitée du développement intégré ou d'applications

Strengths, Weaknesses, Opportunities, and **Threats** – **SWOT**
Forces, vos Faiblesses, ainsi que les Opportunités et **Menaces du marché**

Threats (Section 4.4)

- comparatively small user base
- fragmented community with limited interactions (e.g., on StackOverflow, reddit), see 4.4.1
- active developer community with constant new implementations, features, etc.
- further fragmentation of Prolog implementations, see 4.4.1
- new programming languages
- post-desktop world of JavaScript web-applications
- the perception that it is an “old” language
- wrong image due to “shallow” teaching of the language

Strengths, Weaknesses, Opportunities, and **Threats** – **SWOT**

Forces, vos Faiblesses, ainsi que les Opportunités et **Menaces du marché**

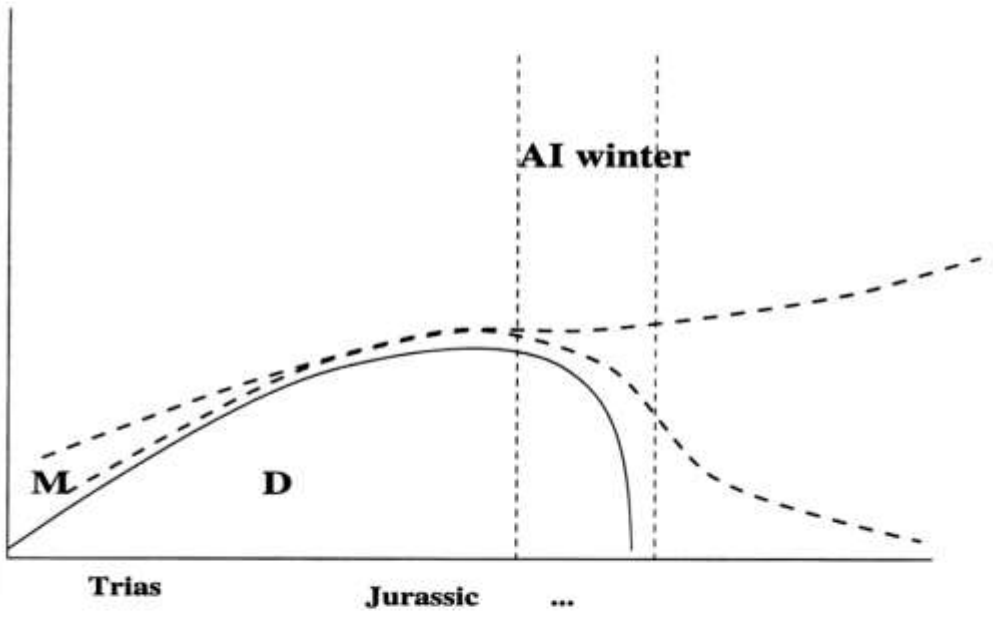
Menaces (Section 4.4)

- base d'utilisateurs relativement petite
- communauté fragmentée avec des interactions limitées (par exemple, sur StackOverflow, reddit), voir 4.4.1
- communauté de développeurs active avec de nouvelles implémentations, fonctionnalités, etc.
- fragmentation supplémentaire des implémentations Prolog, voir 4.4.1
- nouveaux langages de programmation
- monde post-bureau des applications Web JavaScript
- la perception qu'il s'agit d'une « ancienne » langue
- image erronée due à un enseignement "superficiel" de la langue

Prolog
Dinosaure
ou
Mamifère?

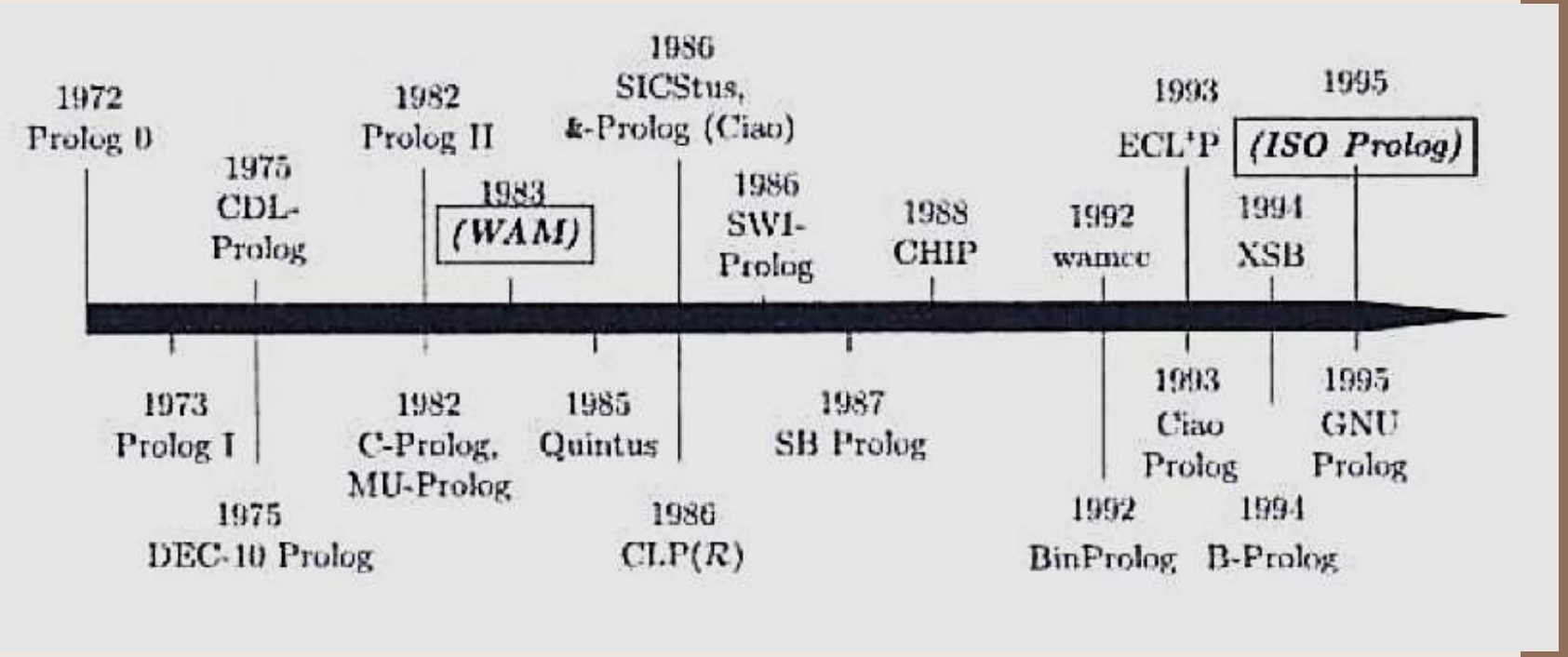
EVOLUTION

Paleo-languagology



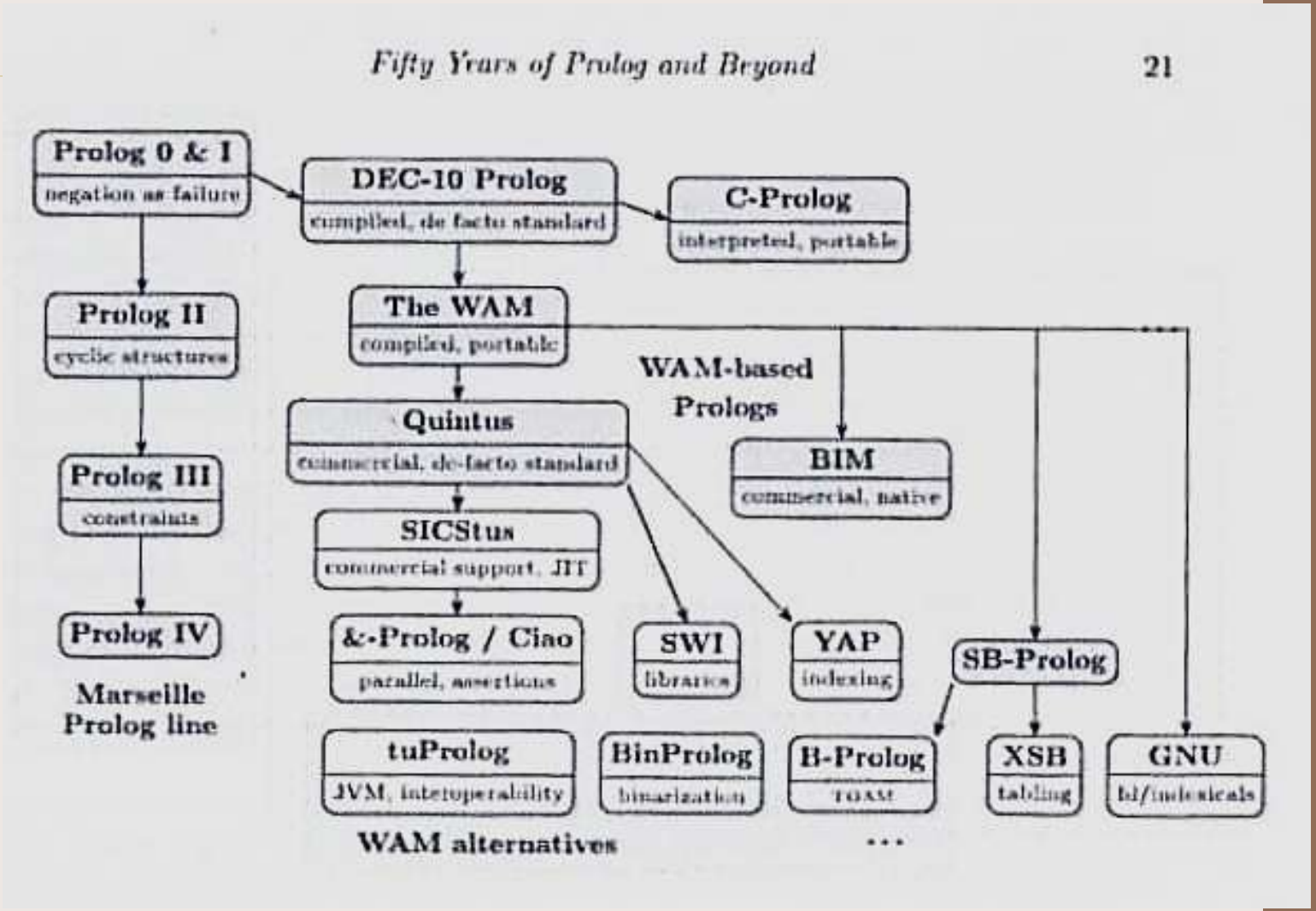
Will Prolog survive after the AI winter?

Early prolog systems



Prolog III **Prolog IV**

Early prolog systems



ISO/IEC JTC1 SC22 WG17 Minutes via Zoom
Date of meeting: 2022-06-06
Time of meeting: 17:00 - 18:09 UTC

1. Greeting and introduction of participants, administrativa

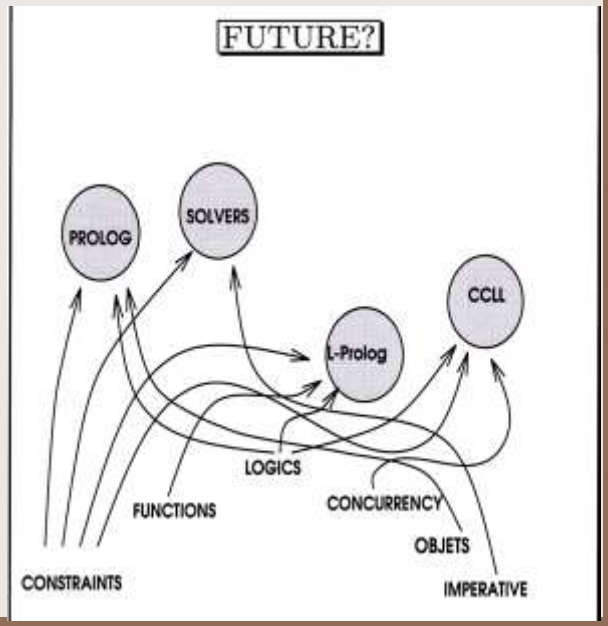
WG17 members:
Klaus Däßler (Germany)
Stefan Kral (Austria)
Markus Triska (Austria)
Manuel Hermenegildo (Spain)
Mark Thom (Canada)
Ulrich Neumerkel (convener)

Invited non-member experts:
Adrian Arroyo Calle (Spain)
Jose A. Riaza (Spain)
Yukata Ichibangase (Japan)
Jose Morales (Spain)
Neng-Fa Zhou (US)

…
Ad 1.4 Prolog archive.
The archive by Paul McJones collects documents and artefacts related to Prolog history. Currently, some systems are not represented. Convener encourages submission.
…

Ad 2 Further corrigenda.
N289 (stc) was briefly reviewed, no items requiring an urgent reaction were identified.
…

Ad 4 DCG progress.
N292 uses better example syntax.
…



Software preservation

TIOBE

About us | Join TIOBE | News | Coding Standards | TIOBE Index | Contact

Products | Quality Models | Markets | Request a demo

Very Long Term History

To see the bigger picture, please find below the positions of the top 10 programming languages of many years back. Please note that these are *average* positions for a period of 12 months.

Programming Language	2022	2017	2012	2007	2002	1997	1992	1987
Python	1	5	8	7	12	-	-	-
C	2	2	2	2	2	1	1	1
Java	3	1	1	1	1	16	-	-
C++	4	3	3	3	3	2	2	5
C#	5	4	4	8	18	-	-	-
Visual Basic	6	14	-	-	-	-	-	-
JavaScript	7	7	10	9	9	21	-	-
Assembly language	8	10	-	-	-	-	-	-
PHP	9	6	5	5	6	-	-	-
SQL	10	-	-	-	7	-	-	-
Fortran	20	27	27	21	11	15	3	7
Prolog	24	33	39	27	16	19	19	3
Lisp	32	31	13	15	13	7	8	2
(Visual) Basic	-	-	7	4	4	3	4	4

Références

Fifty years of Prolog and Beyond,

Philip Körner, Michel Leusschel, João Barbosa, Vitor Santos Barbosa, Veronica Dahl…. (11 autors)

TCLP, 2022

(History of) Logic Programming

Robert Kowalski, In: Volume 9, Computational Logic (Joerg Siekmann, editor). In the History of Logic series, edited by Dov Gabbay and John Woods, Elsevier, 2014, pp 523-569. ScienceDirect / PDF at www.doc.ic.ac.uk/~rak

La plupart des photos sont de P. Deransart, quelques-unes sont extraites du film *ALAIN COLMERAUER L'AVENTURE PROLOG*

Ma profonde reconnaissance à toutes les actrices et acteurs de cette épopée…