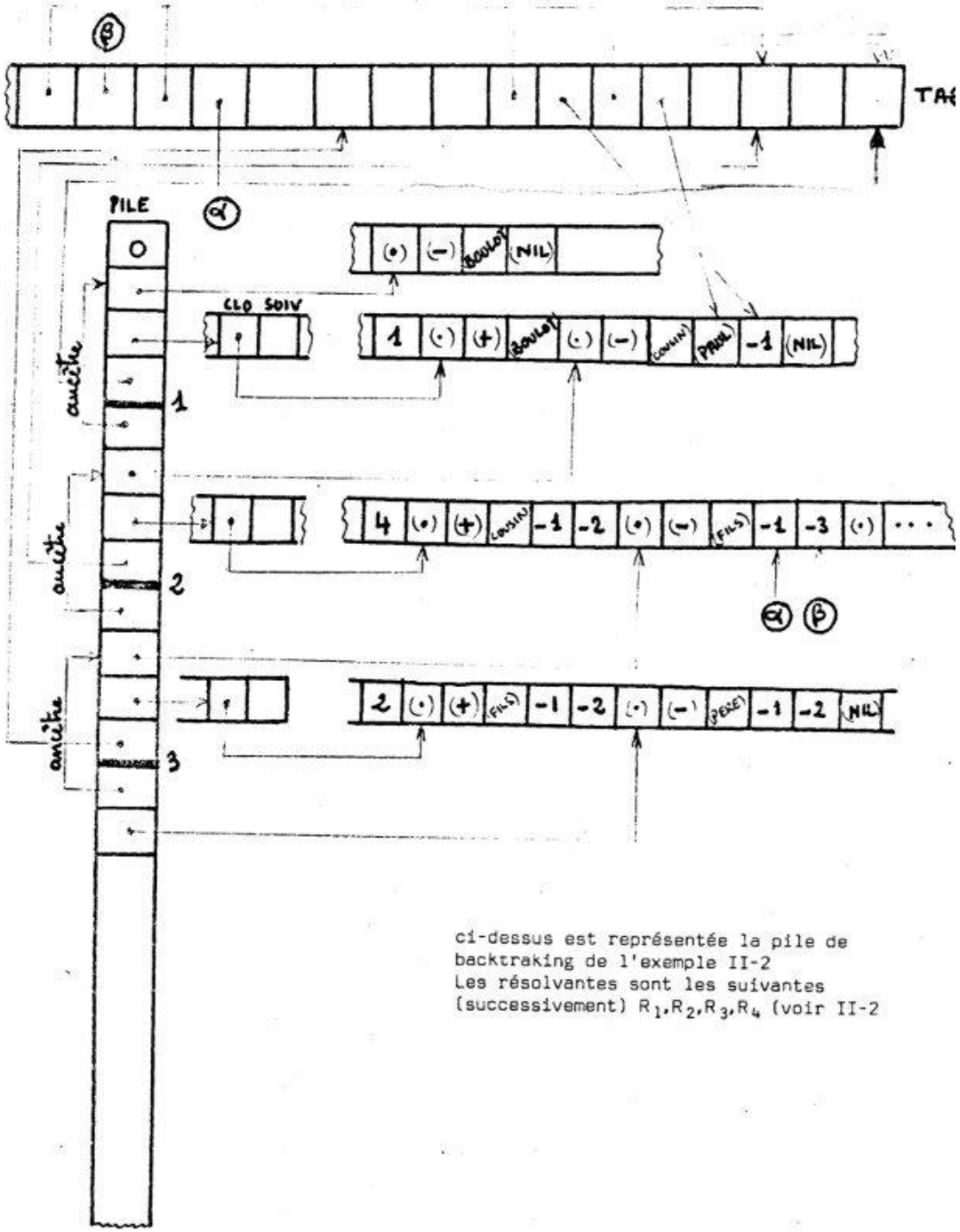


Anatomie d'un Langage de programmation

Prolog et les débuts de la recherche européenne en intelligence artificielle

Séminaire Codes Sources

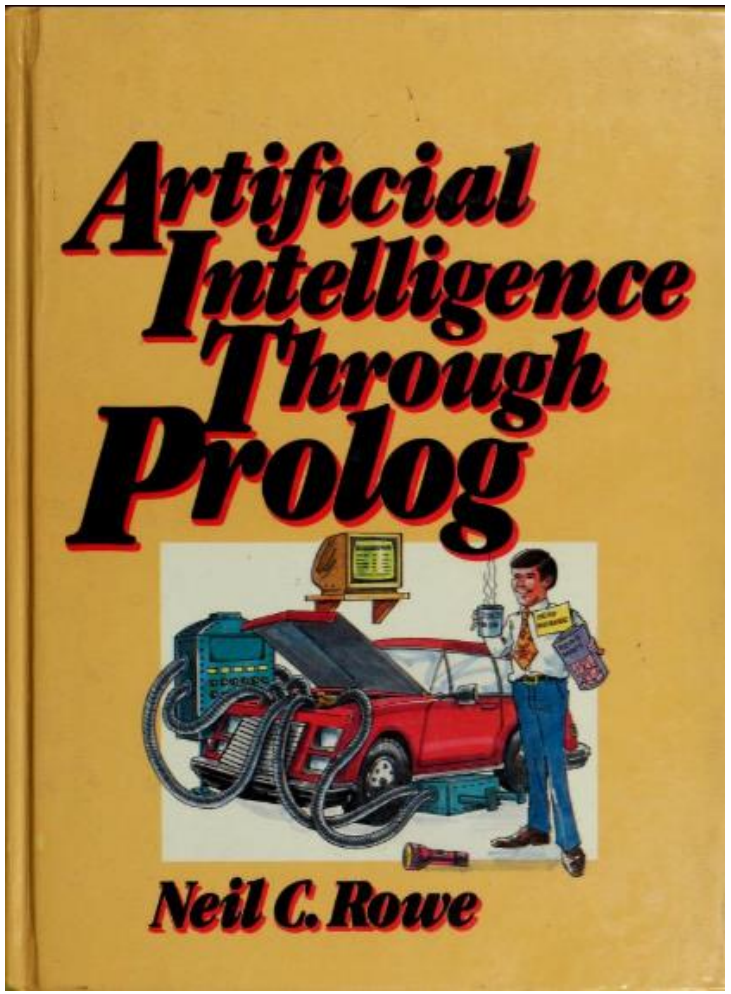
Mathilde Fichen, Ht2s-CNAM

Doctoral researcher

mathfichen.github.io

le cnam

Prolog, un langage de programmation associé au "printemps" de l'IA des années 1980



Artificial Intelligence Through Prolog
Neil C. Rowe, 1988

Prolog: PROgramming in LOGic

Axiomes:

1 +Frere(*y,*z)-Pere(*x,*y)-Pere(*x,*z)

2 +Pere(Paul, Jacques)

3 +Pere(Paul,Pierre)

Données:

4 -Frere(Jacques,*x)/+Reponse(*x)

1 se résoud sur 4:

5 -Pere(*x,Jacques)-Pere(*x,*z)/+Reponse(*z)

2 se résoud sur 5:

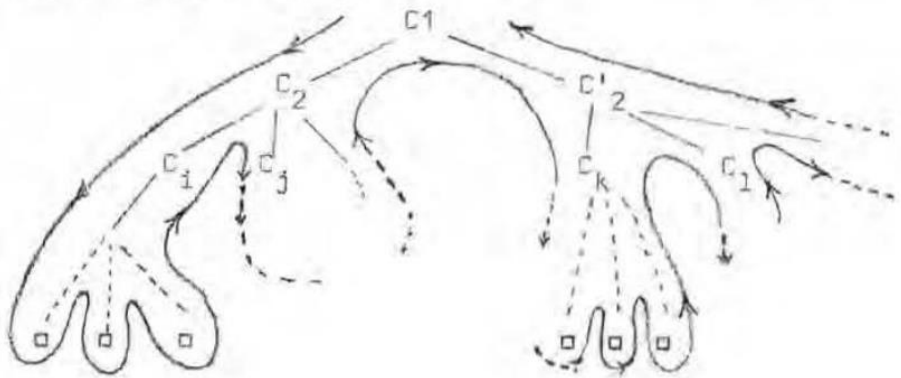
6 -Pere(Paul,*z)/+Reponse(*z)

3 se résoud sur 6:

7 /+Reponse(Pierre)

A programming language born from automated theorem proving

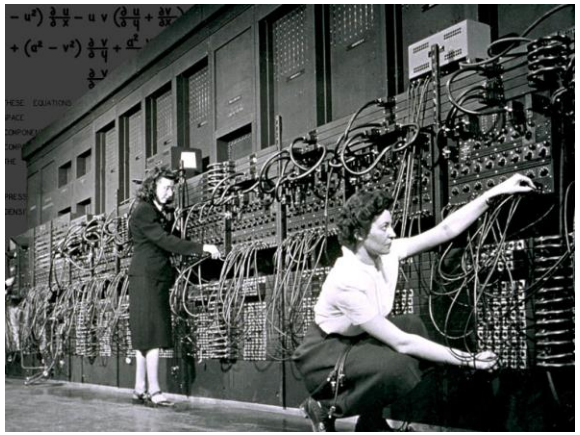
On peut représenter les descendants d'une clause C_1 de <données> sous forme d'un arbre:



Groupe d'Intelligence Artificielle, Université d'Aix-Marseille, 1973, Un système de communication homme-machine en Français, Rapport de recherche sur le contrat CRI n. 72-18

Généalogie techniques des langages préexistants

1946



"the ENIAC was a son-of-a-bitch to program"

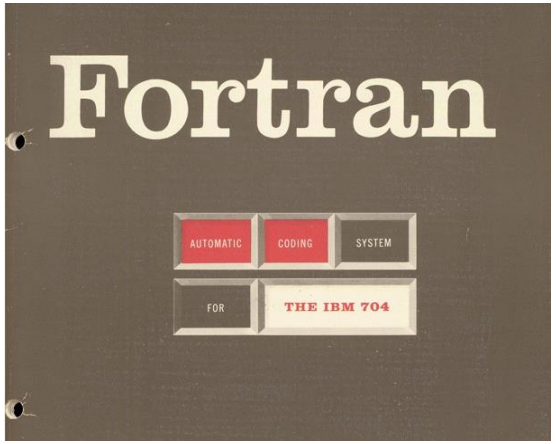
Jean Bartik

1951

The library of tapes on which subroutines are punched is contained in the steel cabinet shown on the left. The operator is punching a program tape on keyboard perforator. She can copy mechanically tapes taken from the library on to the tape she is preparing by placing them in the tapereader shown in the center of the photograph.

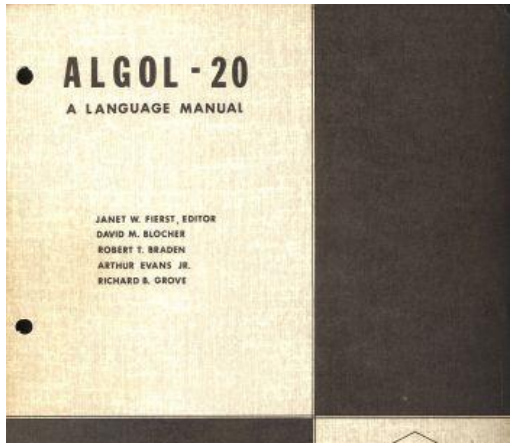
EDSAC, Programmation symbolique, rubans perforés, librairies de routines

Fin années 50



Portabilité (IBM), formulation mathématique, variables, efficacité de la compilation. Puis COBOL.

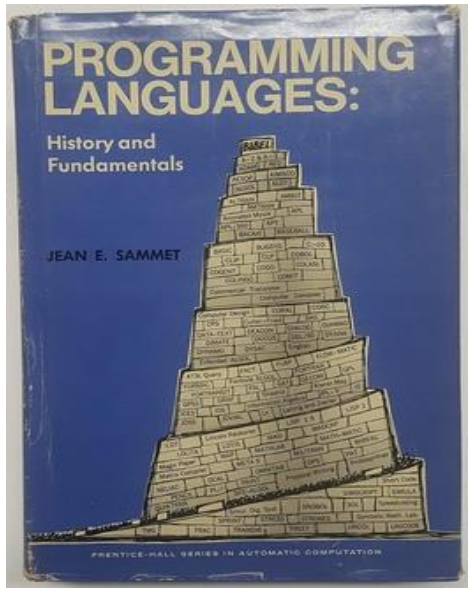
Années 1960



Languages de programmation comme objet académique, objet conceptuel, forme BNF.
Aussi LISP

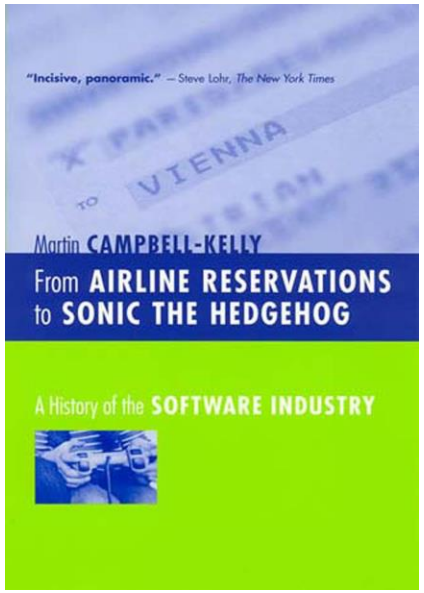
Historiographie des langages de programmation, un rapide état de l'art

Récits d'acteurs



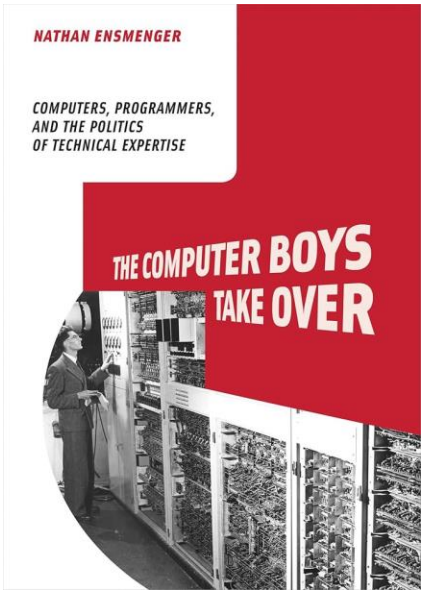
Programming languages: History and Fundamentals, J. Sammet, 1969

Histoire du logiciel

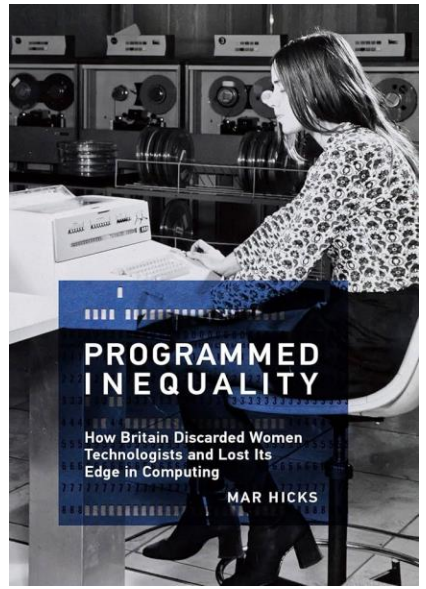


From airline reservations to sonic the hedgehog, M. Campbell-Kelly, 2004

Histoire de la programmation



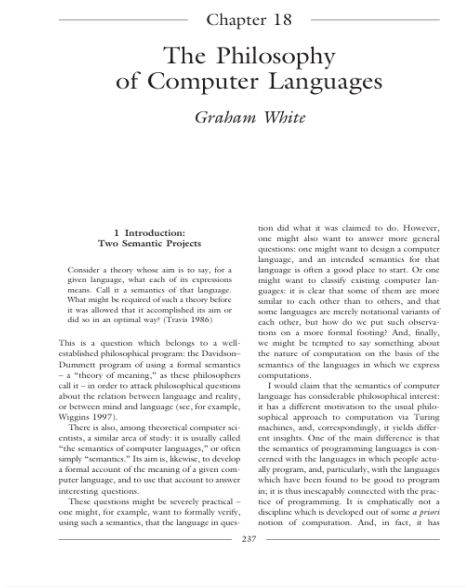
The Computers Boys Take Over, N. Ensmenger, 2010



Programmed Inequality, M. Hicks, 2017

Historiographie des langages de programmation, un rapide état de l'art

Philosophie



Chapter 18

The Philosophy of Computer Languages

Graham White

1 Introduction: Two Semantic Projects

Consider a theory whose aim is to say, for a given language, what each of its expressions means. Call it a semantics of that language. What might be required of such a theory before it was allowed that it accomplished its aim or did so in an optimal way? (Travis 1986)

This is a question which belongs to a well-established philosophical program: the Davidson-Dummett program of using a formal semantics – a "theory of meaning," as these philosophers call it – in order to attack philosophical questions about the relation between language and reality, or between mind and language (see, for example, Wiggins 1997).

There is also, among theoretical computer scientists, a similar area of study: it is usually called "the semantics of computer languages," or often simply "semantics." Its aim is, likewise, to develop a formal account of the meaning of a given computer language, and to use that account to answer interesting questions.

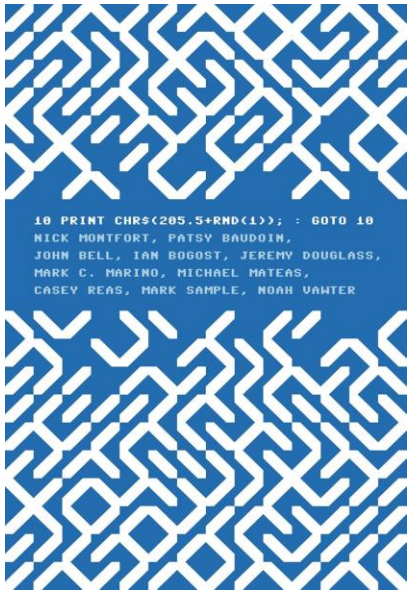
These questions might be severely practical – one might, for example, want to formally verify, using such a semantics, that the language in question did what it was claimed to do. However, one might also want to answer more general questions: one might want to design a computer language, and an intended semantics for that language is often a good place to start. Or one might want to classify existing computer languages: it is clear that some of them are more similar to each other than to others, and that some languages are merely notational variants of each other, but how do we put such observations on a more formal footing? And, finally, we might be tempted to say something about the nature of computation on the basis of the semantics of the languages in which we express computations.

I would claim that the semantics of computer language has considerable philosophical interest: it has a different motivation to the usual philosophical approach to computation via Turing machines, and, correspondingly, it yields different insights. One of the main difference is that the semantics of programming languages is concerned with the languages in which people actually program, and, particularly, with the languages which have been found to be good to program in; it is thus inescapably connected with the practice of programming. It is emphatically not a discipline which is developed out of some *a priori* notion of computation. And, in fact, it has

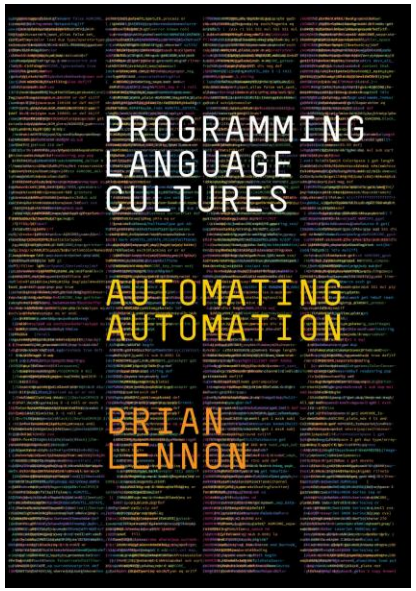
237

The Philosophy of Computer Languages, G. White, 2004

Cultural studies / Media Studies / Code Studies

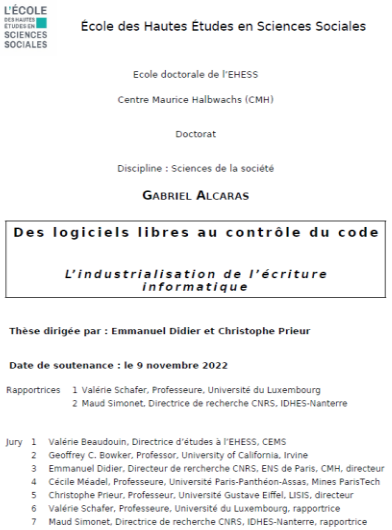


10 PRINT, N. Monfort et al., 2013



Programming language cultures, B. Lennon, 2024

Sociologie



L'ÉCOLE DES HAUTES ÉTUDES EN SCIENCES SOCIALES

École des Hautes Études en Sciences Sociales

Ecole doctorale de l'EHESS

Centre Maurice Halbwachs (CMH)

Doctorat

Discipline : Sciences de la société

GABRIEL ALCARAS

Des logiciels libres au contrôle du code

L'industrialisation de l'écriture informatique

Thèse dirigée par : Emmanuel Didier et Christophe Prieur

Date de soutenance : le 9 novembre 2022

Rapportrices 1 Valérie Schafer, Professeure, Université du Luxembourg
2 Maud Simonet, Directrice de recherche CNRS, IDHES-Nanterre

Jury 1 Valérie Beaudouin, Directrice d'études à l'EHESS, CEMS
2 Geoffrey C. Bowker, Professor, University of California, Irvine
3 Emmanuel Didier, Directeur de rercherche CNRS, ENS de Paris, CMH, directeur
4 Cécile Méadel, Professeure, Université Paris-Panthéon-Assas, Mines ParisTech
5 Christophe Prieur, Professeur, Université Gustave Eiffel, LISIS, directeur
6 Valérie Schafer, Professeure, Université du Luxembourg, rapportrice
7 Maud Simonet, Directrice de recherche CNRS, IDHES-Nanterre, rapportrice

Des logiciels libres au contrôle du code, G. Alcaras, 2022

Problématique et approche

Etude diachronique d'un langage de programmation sur le temps long

Non seulement de sa conception, mais aussi de sa diffusion, de ses usages, de ses représentations

Un dialogue entre la technicité et la matérialité concrète du langage et l'environnement scientifiaue, culturel et social dans lequel il évolue

Prolog, s'inscrit dans une généalogie technique différente des langages préexistants et n'est pas initialement conçu comme un langage de programmation.

Comment qualifier son appartenance à la catégorie technico-scientifique de "langage deprogrammation" ?

Sources

Fonds d'archives

- *Archives personelles d'Alain Colmerauer*
- *Archives du C.N.R.S*
- *Archives Université d'Edimbourg*
- *Archives de l'entreprise Bull (Archives du monde du travail, Roubaix)*

Archives numériques

- *Prolog and Logic Programming Historical Sources Archive, SoftwarePreservation Group*
- *Internet Archive (Manuels, Magazines, Livres)*
- *ACM digital library*

Archives Orales

- *De nombreux acteurs historiques encore présents*

Plan

Prolog comme programme de recherche, 1970-1974

Prolog comme artefact circulant, 1975-1979

Prolog devient un logiciel, 1980-1985

Prolog comme programme de recherche

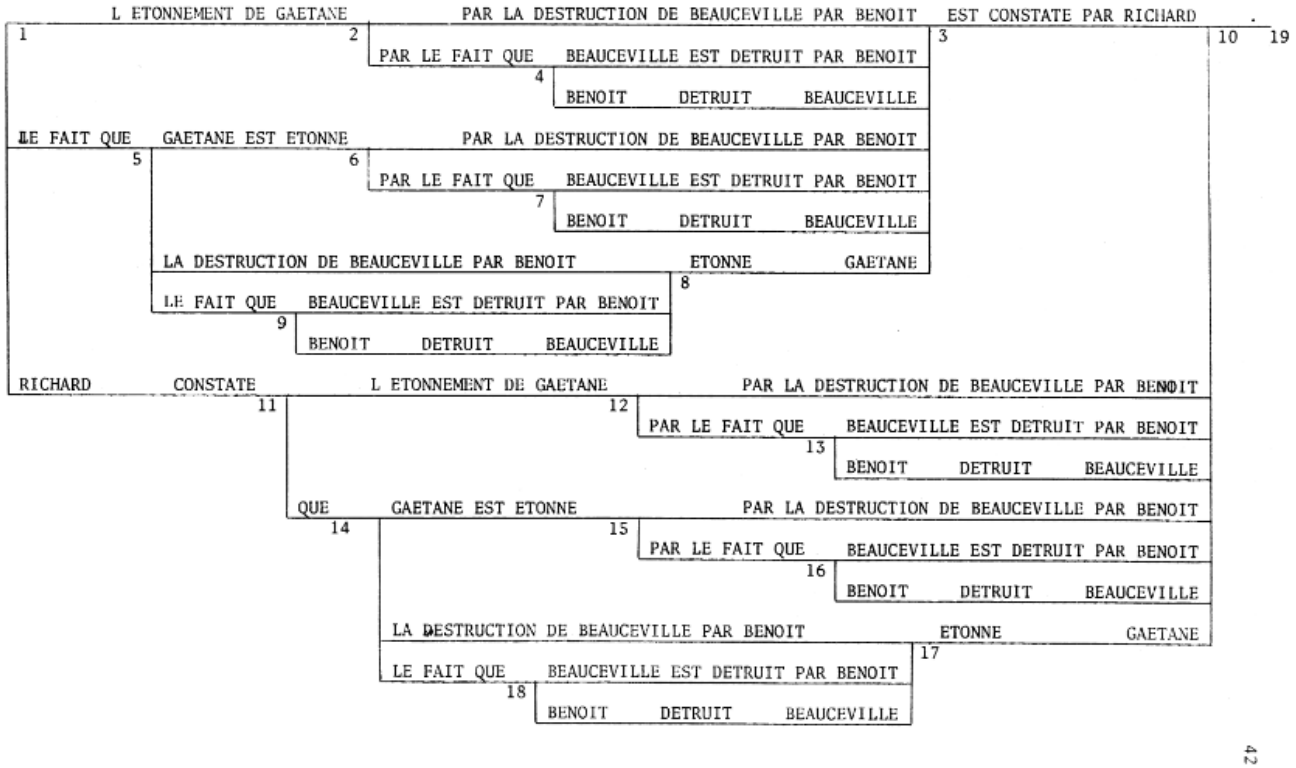
1970-1974

Prolog à Marseille, de la traduction à la démonstration automatique


the Japanese Government (applica-

Figure 4 shows a recent picture of The Mark 1.5 robot device with (left to right) Dr Harry Barrow, Mr Steve Salter, Mr Gregan Crawford and Mr Robin Popplestone.

Traduction automatique, Alain Colmerauer à TAUM Montréal

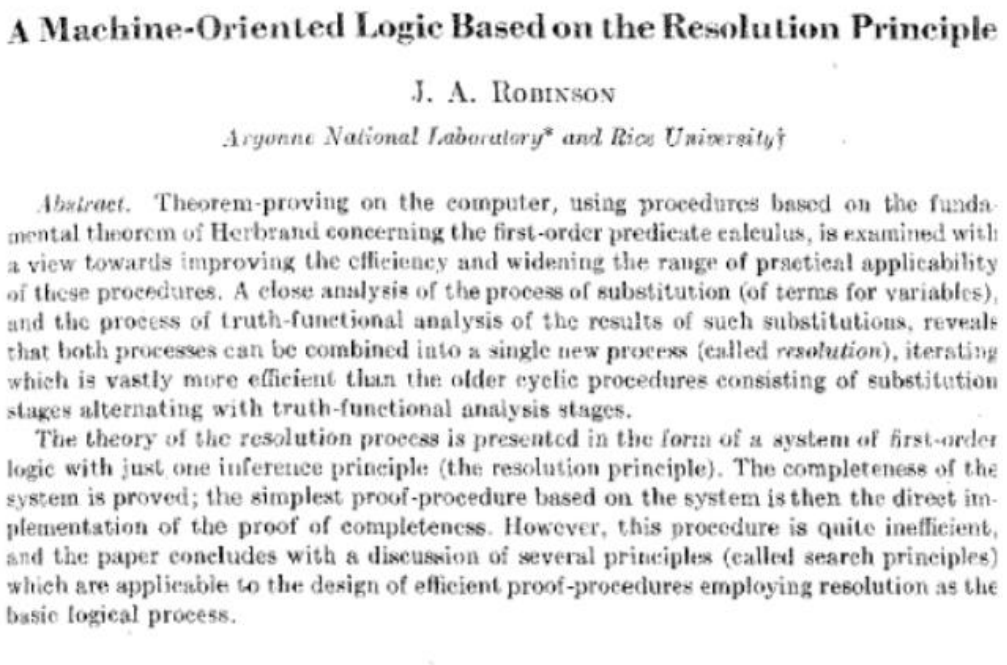


Les Systèmes-Q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur, Colmerauer, 1970

De l'analyse syntaxique au résonnement, Robinson et le "Boom" de la résolution

"Et puis au bout d'un certain temps, j'en avais un petit peu marre de faire de la traduction automatique [...]. Les considérations sont extrêmement syntaxiques et je m'intéressais quand même à faire des programmes qui traitaient l'information qui était contenue dans une phrase […] Il y avait à l'époque des gens qui s'occupaient de démontrer des théorèmes en utilisant une machine. Et c'est là effectivement que j'ai trouvé un article extrêmement intéressant d'un dénommé Alan Robinson. Qui a proposé une technique de démonstration qu'il appelait resolution principle."

Alain Colmerauer, France Culture, 1991



A Machine-Oriented Logic Based on the Resolution Principle

J. A. Robinson

Argonne National Laboratory and Rice University†*

Abstract. Theorem-proving on the computer, using procedures based on the fundamental theorem of Herbrand concerning the first-order predicate calculus, is examined with a view towards improving the efficiency and widening the range of practical applicability of these procedures. A close analysis of the process of substitution (of terms for variables), and the process of truth-functional analysis of the results of such substitutions, reveals that both processes can be combined into a single new process (called *resolution*), iterating which is vastly more efficient than the older cyclic procedures consisting of substitution stages alternating with truth-functional analysis stages.

The theory of the resolution process is presented in the form of a system of first-order logic with just one inference principle (the resolution principle). The completeness of the system is proved; the simplest proof-procedure based on the system is then the direct implementation of the proof of completeness. However, this procedure is quite inefficient, and the paper concludes with a discussion of several principles (called search principles) which are applicable to the design of efficient proof-procedures employing resolution as the basic logical process.

A Machine-Oriented Logic Based on the Resolution Principle J.A. Robinson, 1965

Le Groupe d'Intelligence Artificielle, création d'un système de question réponse avec l'ordinateur

Texte soumis

TOUT PSYCHIATRE EST UNE PERSONNE.
CHAQUE PERSONNE QU'IL ANALYSE, EST MALADE.
*JACQUES EST UN PSYCHIATRE A *MARSEILLE.
EST-CE QUE *JACQUES EST UNE PERSONNE?
OU EST *JACQUES?
EST-CE QUE *JACQUES EST MALADE?

Réponse

OUI

A MARSEILLE

JE NE SAIS PAS

Groupe d'Intelligence Artificielle, Université d'Aix-Marseille, 1973, Un système de communication homme-machine en Français, Rapport de recherche sur le contrat CRI n. 72-18

le cnam

Emergence progressive de Prolog par essai-erreur, le choix de l'efficacité

(1) d'améliorer notre formalisme général à base de logique du 1er ordre et de démonstration automatique, qui deviendra ainsi, en quelque sorte, notre langage de programmation.

(2) d'utiliser ce "langage de programmation" pour compléter l'écriture de:

-notre analyseur du français
-notre "raisonnaur"
-notre synthétiseur du français.

Groupe d'Intelligence Artificielle, Université d'Aix-Marseille, 1972, Rapport scientifique

```
OPERATEURS ARITH
  BINAIRE DG + .
  BINAIRE DG . .
AMEN

LIRE AXIOMES
 +T(*A + *B,*C)-T(*A,*A1)-T(*B,*B1)-TPLUS(*A1,*B1,*C)..
 +T(*A . *B,*C)-T(*A,*A1)-T(*B,*B1)-TMULT(*A1,*B1,*C)..
 +T(*A,*A)..

 +TPLUS(*A + *B,*C,*A+*D)-TPLUS(*B,*C,*D)..
 +TPLUS(*A,*B,*A+*B)..

 +TMULT(*A,*B+*C,*D)-TMULT(*A,*B,*D1)-TMULT(*A,*C,*D2)
                    -TPLUS(*D1,*D2,*D)..
 +TMULT(*A+*B,*C,*D)-TMULT(*A,*C,*D1)-TMULT(*B,*C,*D2)
                    -TPLUS(*D1,*D2,*D)..
 +TMULT(*A.*B,*C,*A.*D)-TMULT(*B,*C,*D)..
 +TMULT(*A,*B,*A.*B)..
AMEN

LIRE DONNEES
-T( (A+B).(C+E.F+A).(E+H)+I+(A+B.C) , *X) /+REP(*X)..
AMEN

DEMONTRER(AXIOMES,DONNEES,RESULT)
ECRIRE(RESULT)
AMEN
```

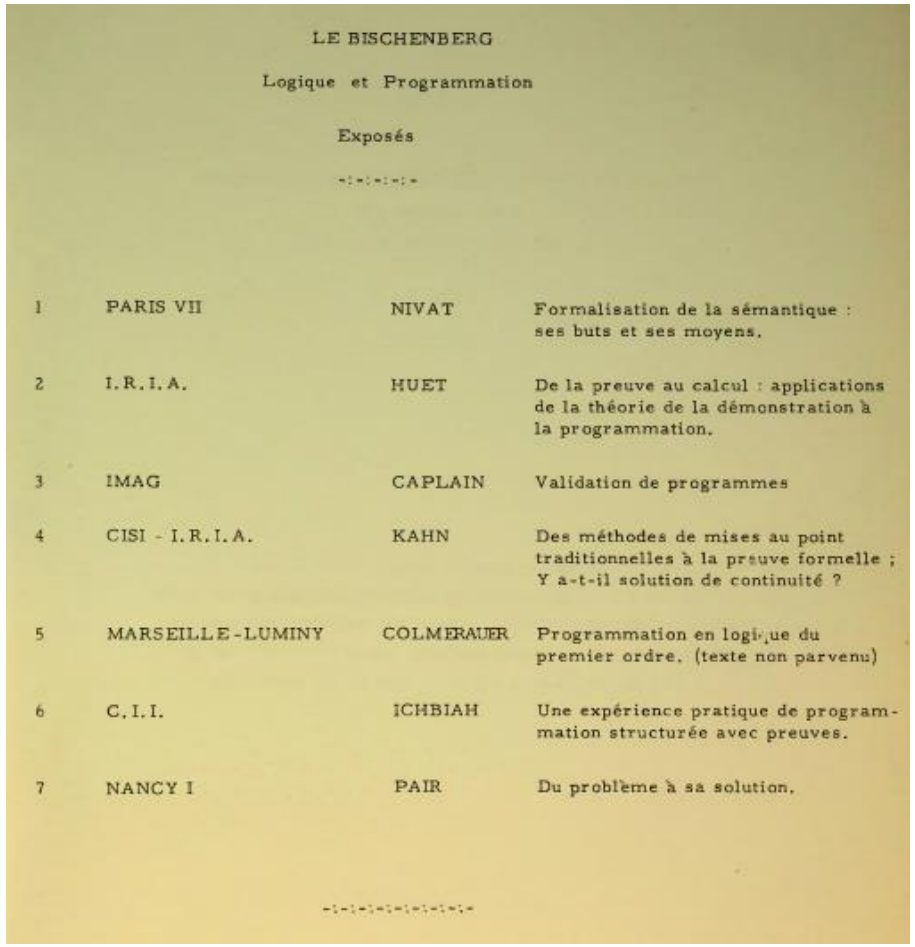
Groupe d'Intelligence Artificielle, Université d'Aix-Marseille, 1973, Un système de communication homme-machine en Français, Rapport de recherche sur le contrat CRI n. 72-18

le cnam

Intelligence artificielle versus informatique théorique

Je m'attendais à une salle qui partagerait mon enthousiasme et mon admiration et au lieu de cela, ce fut un feu roulant de critiques de la part des informaticiens qui étaient les plus versés en logique. Les objections les plus virulentes venaient de deux d'entre eux, le premier reprochant, entre autres, à la logique sous-jacente de ne pas être décidable, […] le second critiquant l'emploi d'une unification sans test d'occurrence et le slash qui brise la complétude.

Pierre Lescanne, 2021



LE BISCHENBERG

Logique et Programmation

Exposés

1	PARIS VII	NIVAT	Formalisation de la sémantique : ses buts et ses moyens.
2	I.R.I.A.	HUET	De la preuve au calcul : applications de la théorie de la démonstration à la programmation.
3	IMAG	CAPLAIN	Validation de programmes
4	CISI - I.R.I.A.	KAHN	Des méthodes de mises au point traditionnelles à la preuve formelle ; Y a-t-il solution de continuité ?
5	MARSEILLE-LUMINY	COLMERAUER	Programmation en logique du premier ordre. (texte non parvenu)
6	C.I.I.	ICHBIAH	Une expérience pratique de programmation structurée avec preuves.
7	NANCY I	PAIR	Du problème à sa solution.

Textes Présentés aux Journées Logique et Programmation, Le Bischenberg, IRIA, 19-20 novembre 1975

le cnam

Prolog à Edimbourg, support d'une recherche théorique

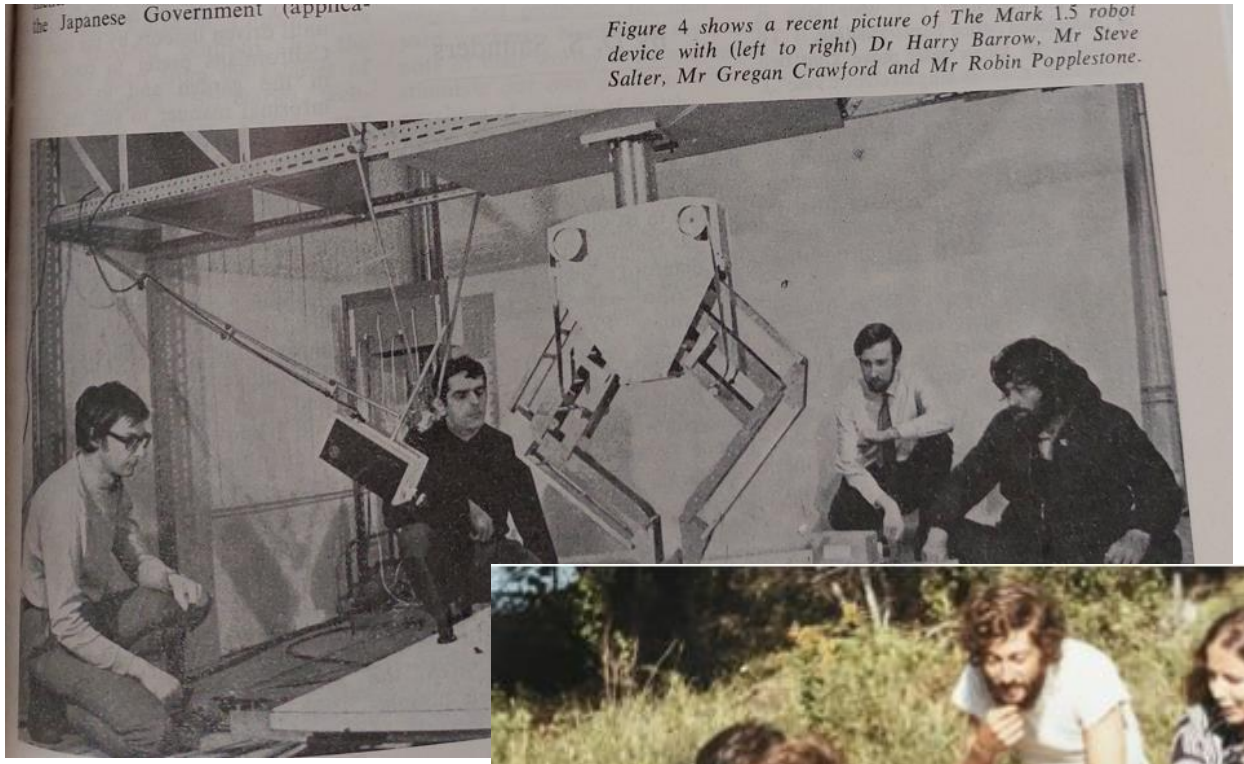
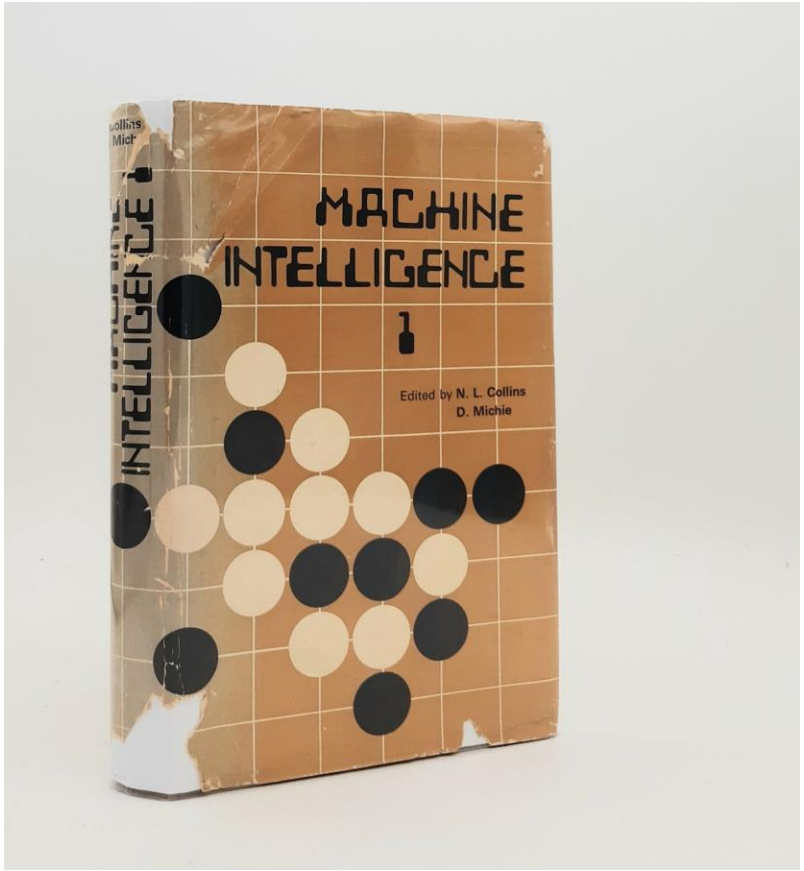

the Japanese Government (applica-

Figure 4 shows a recent picture of The Mark 1.5 robot device with (left to right) Dr Harry Barrow, Mr Steve Salter, Mr Gregan Crawford and Mr Robin Popplestone.

Edimbourg et le Machine Intelligence Department



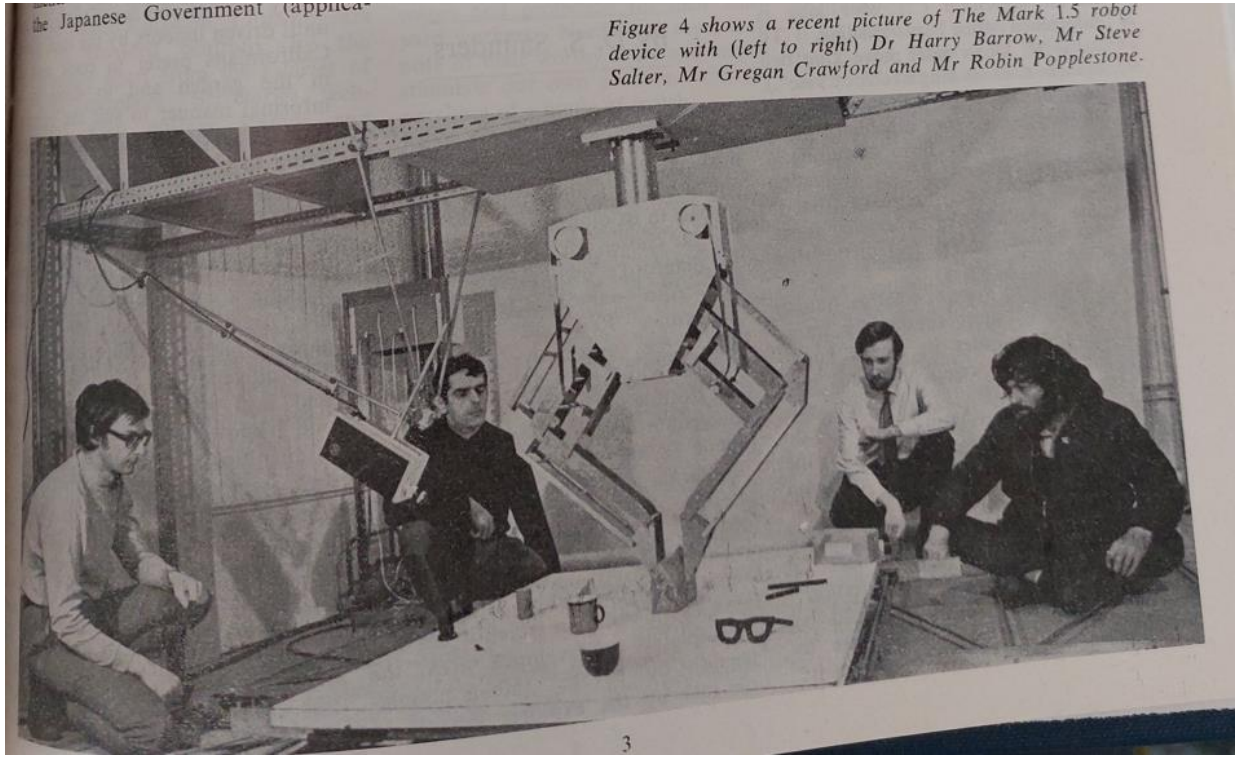

the Japanese Government (applica-

Figure 4 shows a recent picture of The Mark 1.5 robot device with (left to right) Dr Harry Barrow, Mr Steve Salter, Mr Gregan Crawford and Mr Robin Popplestone.

Bob Kowalski et la programmation logique, début d'un paradigme

Linear Resolution with Selection Function, Robert Kowalski, 1971

Linear Resolution with Selection Function

Robert Kowalski and Donald Kuehner
Metamathematics Unit, University of Edinburgh

Recommended by B. Meltzer

Predicate logic as programming language, Robert Kowalski, 1974

INFORMATION PROCESSING 74 – NORTH-HOLLAND PUBLISHING COMPANY (1974)

PREDICATE LOGIC AS PROGRAMMING LANGUAGE*

Robert KOWALSKI
University of Edinburgh, Department of Computational Logic
Edinburgh, Scotland

The interpretation of predicate logic as a programming language is based upon the interpretation of implications

B if A_1 and ... and A_n

as procedure declarations, where B is the procedure name and $A_1,...,A_n$ is the set of procedure calls A_i constituting the procedure body. An axiomatisation of a problem domain is a program for solving problems in that domain. Individual problems are posed as theorems to be proved. Proofs are computations generated by the theorem-prover which executes the program incorporated in the axioms. Our thesis is that predicate logic is a useful and practical, high-level, non-deterministic programming language with sound theoretical foundations.

1. INTRODUCTION

The purpose of programming languages is to enable the communication from man to machine of problems and their general means of solution.

The first programming languages were machine languages. To communicate, the programmer had to learn the psychology of the machine and to express his problems in machine-oriented terms. Higher-level languages developed from machine languages through the provision of facilities for the expression of problems in terms closer to their original conceptualisation.

Concerned with the other end of the man-to-machine communication problem, predicate logic derives from efforts to formalise the properties of rational human thought. Until recently, it was studied with little interest in its potential as a language for man-machine communication. This potential has been realised by recent discoveries in computational logic which have made possible the interpretation of sentences in predicate logic as programs, of derivations as computations and of proof procedures as feasible executors of predicate logic programs.

As a programming language, predicate logic is the only language which is entirely user-oriented. It differs from existing high-level languages in that it possesses no features which are meaningful only in machine-level terms. It differs from functional languages like LISP, based on the λ-calculus, in that it derives from the normative study of human logic, rather than from investigations into the mathematical logic of functions.

This paper deals only in a preliminary way with some of the issues raised by the consideration of predicate logic as a programming language. The semantics of predicate logic as a programming language is investigated in another paper with Maarten van Emden [5]. A more comprehensive investigation of the use of predicate logic for the representation of knowledge is in preparation. Hayes [8] and Sandewall [23] have also concerned themselves with topics related to the programming language interpretation of predicate logic. An earlier investigation with similar objectives was carried out by Cordell Green [7].

2. SYNTAX

All questions concerning logical implication in first order logic can be replaced by questions concerning unsatisfiability of sentences in clausal form.

*This research was sponsored by a grant from the Science Research Council.

Such sentences have an especially simple syntax and lack none of the expressive power of the full predicate calculus. A sentence in clausal form is a set of clauses. A clause is a pair of sets of atomic formulas, written

$$B_1,...,B_m \leftarrow A_1,...,A_n$$

An atomic formula has the form $P(t_1,...,t_k)$ where P is a k-ary predicate symbol and the t_i are terms. A term is either a variable x,y,z,... or an expression $f(t_1,...,t_k)$, where f is a k-ary function symbol and the t_i are terms. The sets of all predicate symbols, function symbols and variables are any three sets of mutually disjoint symbols. Constants are 0-ary function symbols.

3. SEMANTICS

The semantics of sentences in clausal form is as simple as their syntax. Interpret a set of clauses $\{C_1,...,C_n\}$ as a conjunction,

C_1 and C_2 and ... and C_n .

Interpret a clause $B_1,...,B_m \leftarrow A_1,...,A_n$, containing variables $x_1,...,x_k$ as a universally quantified implication,

for all $x_1,...,x_k$, B_1 or ... or B_m
is implied by A_1 and ... and A_n.

The special cases where m = 0 or n = 0 deserve special readings.

If n = 0, read

for all $x_1,...,x_k$, B_1 or ... or B_m .

If m = 0

for no $x_1,...,x_k$, A_1 and ... and A_n.

If both m = 0 and n = 0, write the null clause,

□

interpreted as denoting falsity (or contradiction).

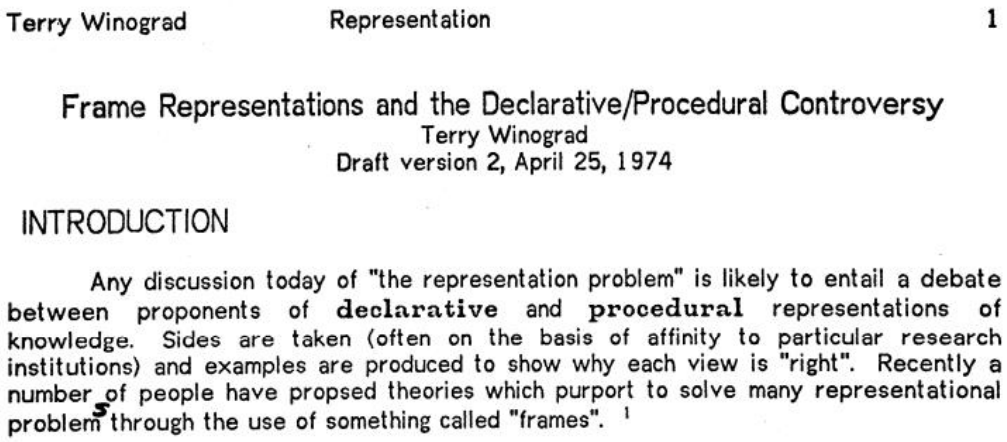
Methods for transforming arbitrary first-order sentences into clausal form are described in Nilsson's book [19]. It is our thesis, however, that clausal form defines a natural and useful language in its own right, that thoughts can conveniently be expressed directly in clausal form, and that literal translation from another language, such as full predicate logic, often distorts the original thought.

4. EXAMPLE: A PROGRAM FOR COMPUTING FACTORIAL

(F1) Fact(0,s(0)) ←
(F2) Fact(s(x),u) ← Fact(x,v),Times(s(x),v,u)

Regard the terms 0, s(0), s(s(0)),... as the numerals 0,1,2,.... Read Fact(x,y) as stating that the

Controverses, logiciens vs. procéduralistes



Terry Winograd Representation 1

Frame Representations and the Declarative/Procedural Controversy

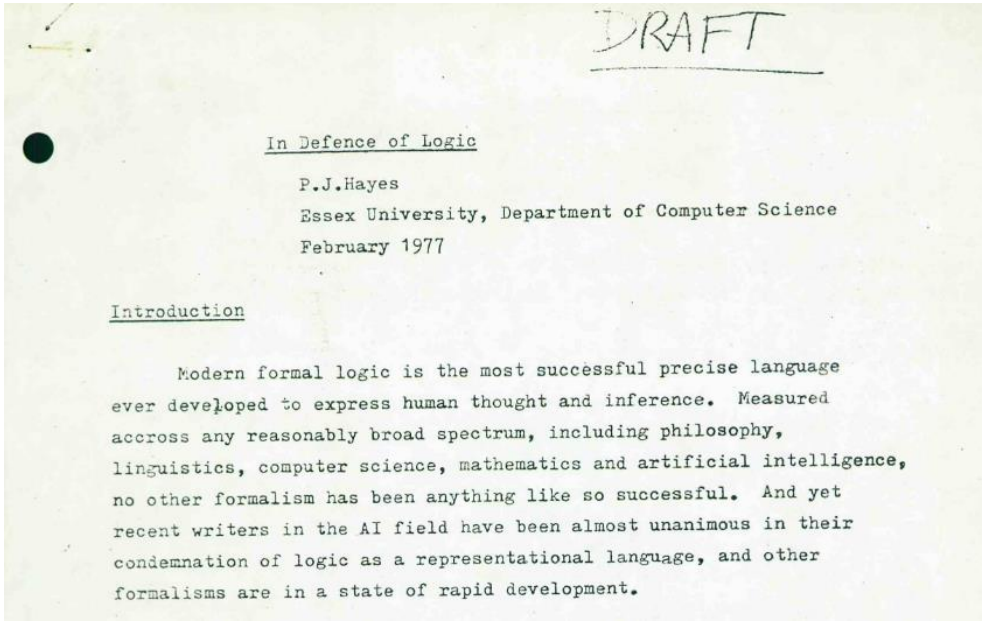
Terry Winograd

Draft version 2, April 25, 1974

INTRODUCTION

Any discussion today of "the representation problem" is likely to entail a debate between proponents of **declarative** and **procedural** representations of knowledge. Sides are taken (often on the basis of affinity to particular research institutions) and examples are produced to show why each view is "right". Recently a number of people have propsed theories which purport to solve many representational problem through the use of something called "frames". [1]

Frame représentation and the Declarative/Procédural Controversy, T. Winograd, 1974



DRAFT

In Defence of Logic

P.J.Hayes

Essex University, Department of Computer Science

February 1977

Introduction

Modern formal logic is the most successful precise language ever developed to express human thought and inference. Measured accross any reasonably broad spectrum, including philosophy, linguistics, computer science, mathematics and artificial intelligence, no other formalism has been anything like so successful. And yet recent writers in the AI field have been almost unanimous in their condemnation of logic as a representational language, and other formalisms are in a state of rapid development.

In Défense of Logic, Pat Hayes, 1977

Prolog comme preuve de validité de l'approche logique

The predicate calculus program written by the Artificial Intelligence Group in Marseille is the most ambitious, large-scale application anywhere of predicate calculus as a programming language. Our own more theoretical research has sometimes been handicapped in the past by being too distant from practical applications. This has often had its advantages. But at this stage, practical experience with useful, running programs is what our investigations need most. On the other hand, no one has made greater appli-

Grant application to the North Atlantic Treaty Organization, Robert Kowalski, 1973

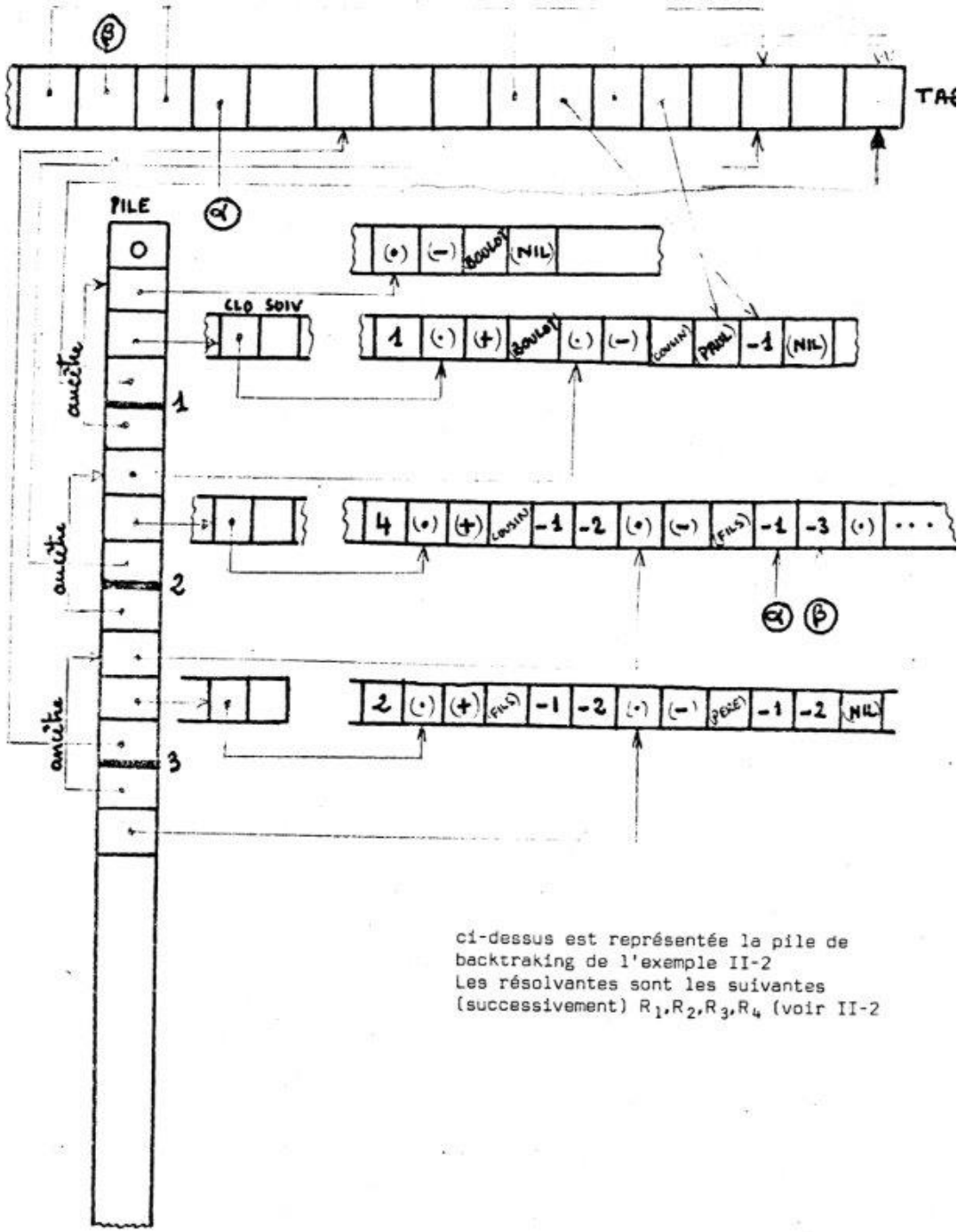
ACKNOWLEDGMENT

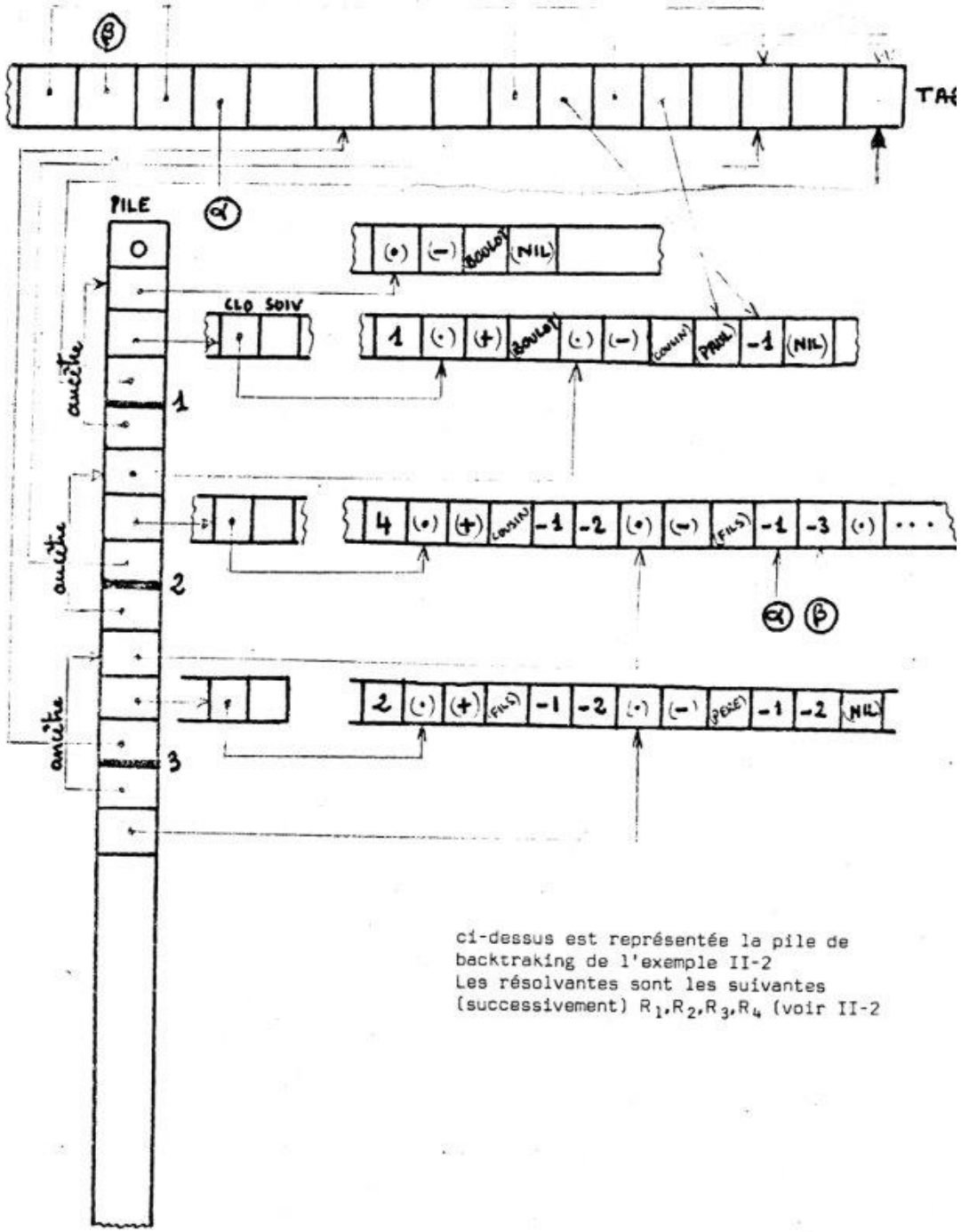
That sets of axioms are like programs, in the way that different formulations can have equivalent meanings but very different influences on efficiency, is a point of view which runs counter to the prevailing moods in symbolic logic and in artificial intelligence In particular, the attacks by Anderson and Hayes {1} and by Minsky and Papert {17} against the utility of the theorem-proving paradigm depend upon the assumption that axioms convey meaning but not pragmatic information. Our contrary point of view was reinforced by joint research with Alain Colmerauer (reported in {11}) on axiomatisations of grammars, regarded as programs for syntactic analysis. Further

Predicate logic as programming language, Robert Kowalski, 1974

Prolog comme artefact circulant

1974-1980





ci-dessus est représentée la pile de backtraking de l'exemple II-2
Les résolvantes sont les suivantes (successivement) R_1, R_2, R_3, R_4 (voir II-2

Le tournant de l'interpréteur FORTRAN

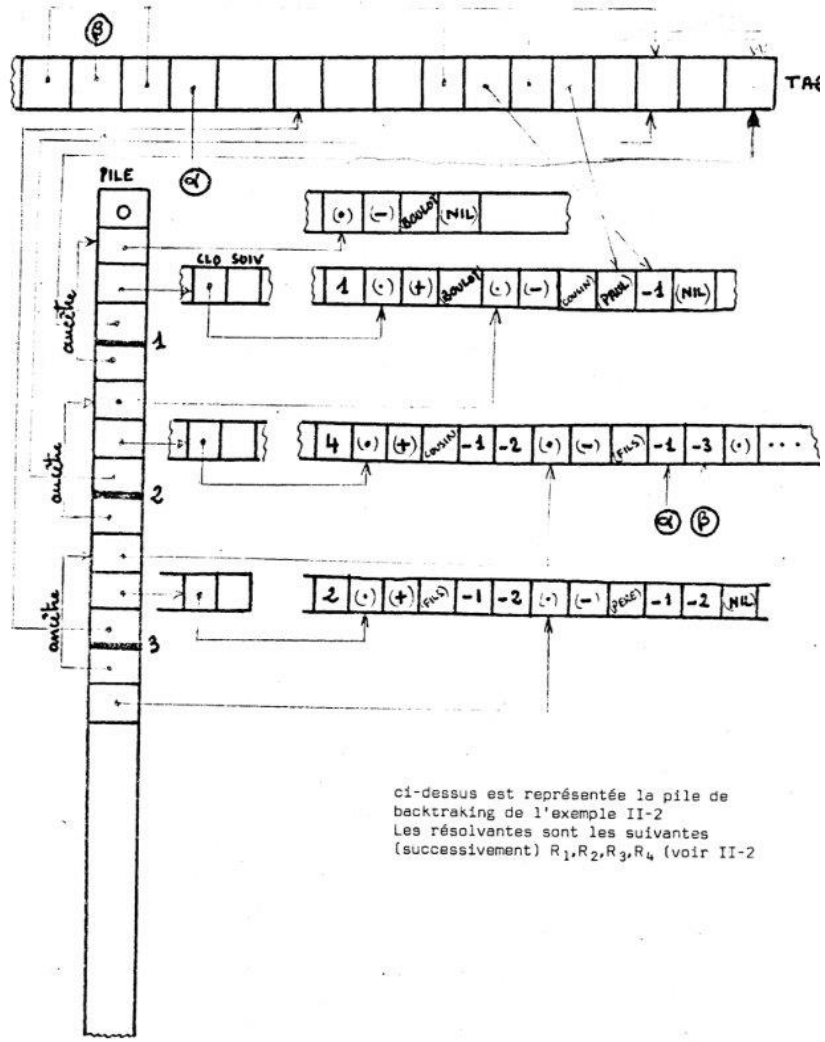
Prolog devient un objet autonome

III L'INTERPRETEUR

III-1 FONCTIONNEMENT - GENERALITES

L'interpréteur est écrit en FORTRAN. Ce langage étant encore le plus répandu, cela permet d'adapter l'interpréteur sur toutes les machines possèdant un compilateur de FORTRAN.

Interpréteur du langage de programmation Prolog, G. Battani et H. Meloni, Université d'Aix Marseille, 1973



ci-dessus est représentée la pile de backtraking de l'exemple II-2
Les résolvantes sont les suivantes (successivement) R_1, R_2, R_3, R_4 (voir II-2

David Warren, Prolog comme factoriseur et langue commune pour l'IA

4.1 The Database

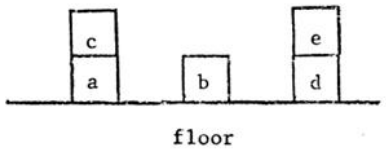
```
+ add ( on(U,W) ,   move(U,V,W) ).
+ add ( clear(V),   move(U,V,W) ).

+ del ( on (U,Z),   move(U,V,W) ).
+ del ( clear(W),   move(U,V,W) ).

+ can ( move(U,V,floor),  on(U,V)&V≠floor&clear (U) ).
+ can ( move(U,V,W)    ,  clear(W)&on (U,V)&U≠W&clear(U) ).

+ imposs ( on(X,Y) &clear(Y) ).
+ imposs ( on(X,Y) &on(X,Z) &Y ≠ Z ).
+ imposs ( on(X,X)  ).

+ given (start, on(a,floor)).
+ given (start, on(b,floor)).
+ given (start, on(c,a) ).
+ given (start, on(d,floor)).
+ given (start, on(e,d) ).
+ given (start, clear(b)).
+ given (start, clear(c)).
+ given (start, clear(e)).
```



WARPLAN: a system for generating plans, David Warren, 1974

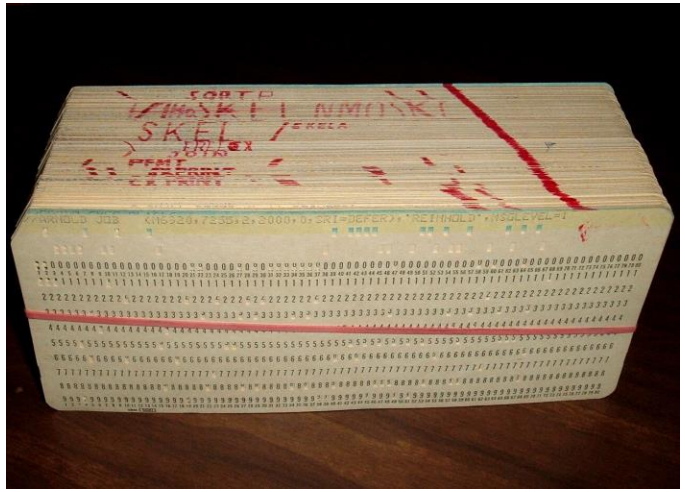
High-level languages have progressed by successively transferring responsibilities from the user to the software engineer. The user is relieved of certain machine-oriented tasks which would make his program clumsy to formulate and difficult to debog, leaving him free to concentrate on the problem-oriented aspects.

WARPLAN: a system for generating plans, David Warren, 1974

A factor hindering the progress of AI seems to be that AI programs are not produced in a form suitable for easy compréhension by humans. The essential ideas underlying the program become obscured when it is "coded". Consequently the programs themeselves are rarely published, but are only described, leading inevitably to vagueness and ambiguity.

WARPLAN: a system for generating plans, David Warren, 1974

Prolog traverse la manche



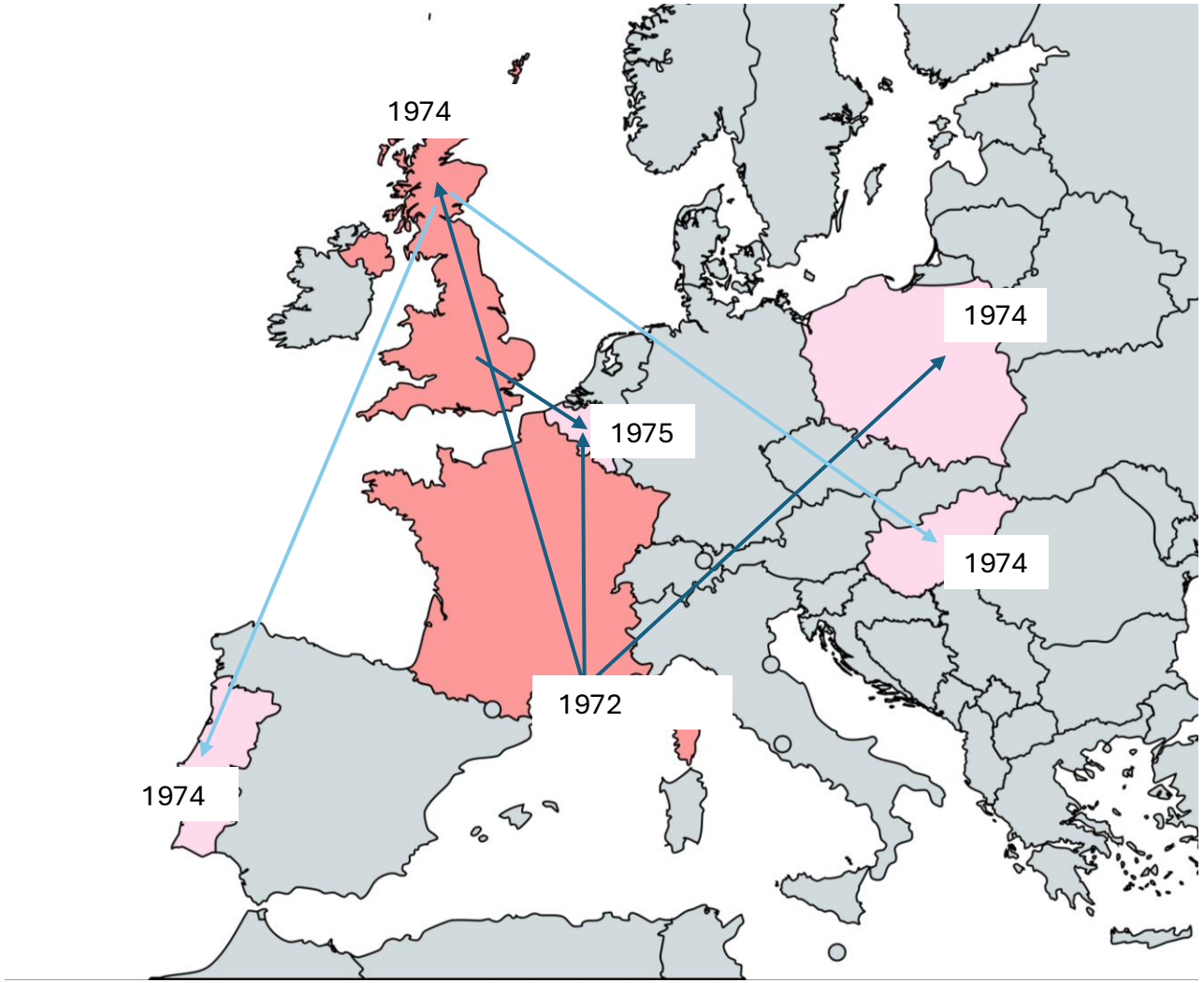
"I came back with a big deck of cards with the Prolog system and my own Warplan. Miraculously it all worked on the DEC10 in Edimbourg even though it was developped for an IBM mainframe in Grenoble."

Entretien avec David Warren, Mathilde Fichen, 16 avril 2024

"The effect was electrifying. Actually being able to run programs made a huge difference, though it shouldn't have from a rational point of view."

Entretien avec Maarten van Emden, The Software Preservation Group 2006

Et se diffuse de proche en proche

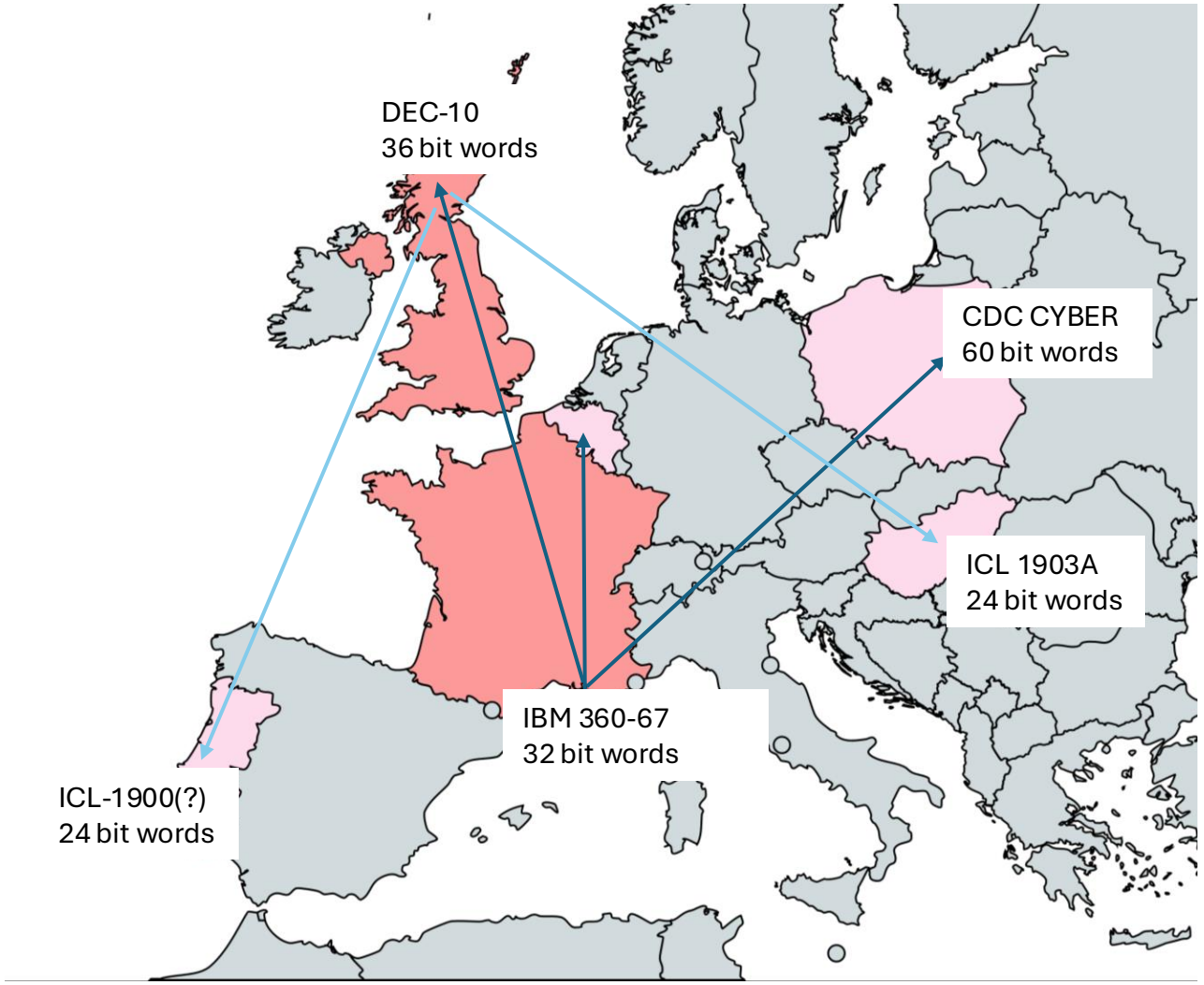


Distribution of the Marseille Prolog FORTRAN interpreter in 1974-1975

Un problème de bit

"The effect was that our CDC CYBER 73, which ran at approximately 1.2 million instructions per second, read in Prolog programs at the average speed of 5 seconds per clause." (F. Kluzniak, Software Preservation Group)

"Porting of the Marseille Fortran interpreter to the ICL 1903A computer of NIM IGÜSZI was included into our research contract for 1975. A member of Németi's group, Péter Tóth, who was given this task, encountered unexpected problems due to the different word and character size (ICL 1900 had 24 bit words and 6 bit characters)." (P.Szeredi, Software Preservation Group)



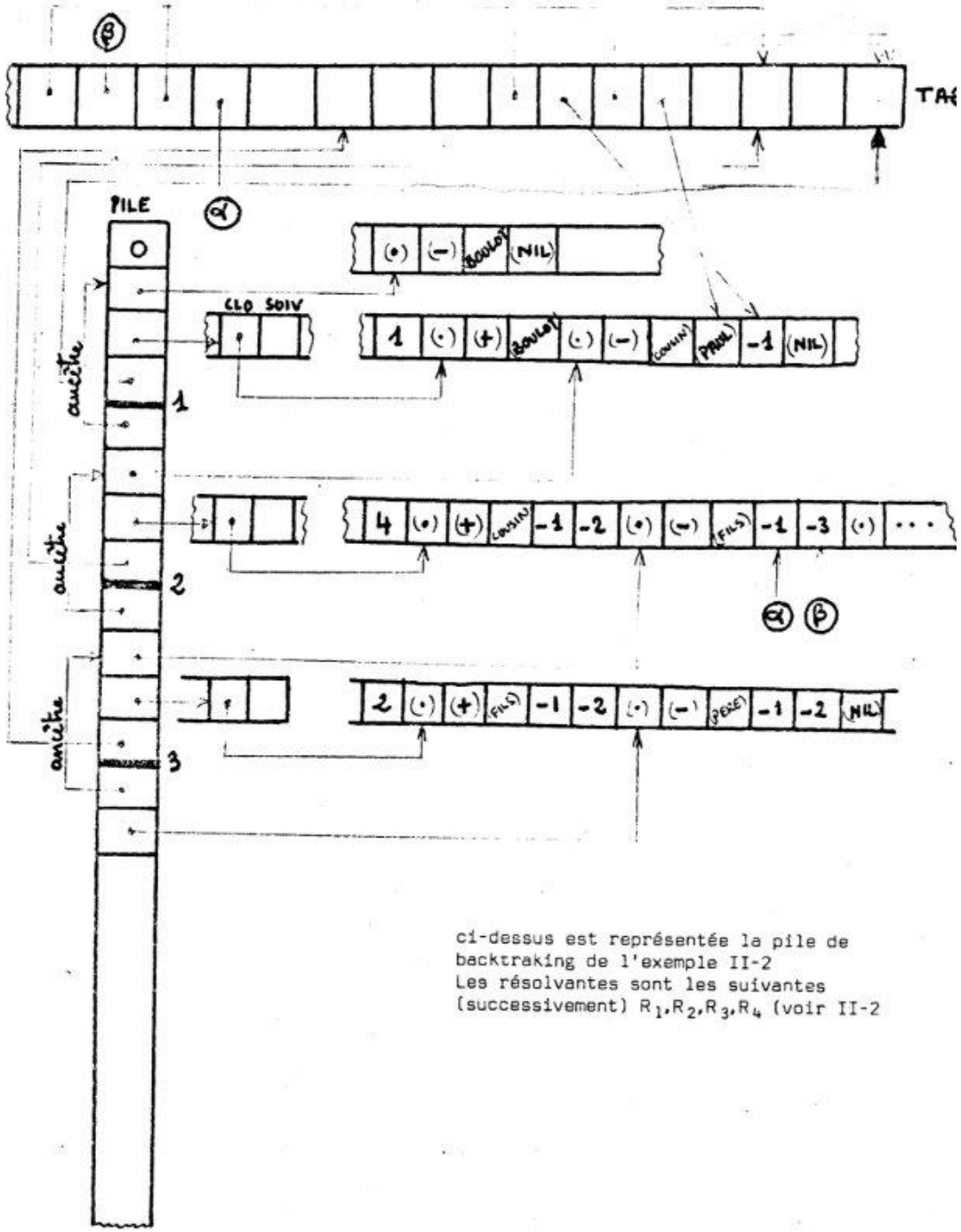
Un profile récurrent, l'ingénieur logicien

David Warren (Edimbourg)
- *Diplômé en mathématiques à l'université de Cambridge*
- *Ingénieur chez IBM et ICL*
- *Thèse à la School of AI, Metamathematics Unit (Edimbourg)*

Fernando Pereira (Lisbon)
- *Etudie d'abord l'ingéniérie électronique puis les mathématiques à l'Université technique de Lisbonne*
- *Travail étudiant comme programmeur au centre de calcul au aboratoire d'ingénierie civil de Lisbonne*
- *S'intéresse et lit des articles de logique mathématique, d'IA*
- *Thèse à la School of AI d'Edimbourg*

Peter Szeredi (Budapest)
- *Diplômé en mathématiques à l'université Eötvös Loránd University, Budapest*
- *Travail étudiant (programmeur) puis à plein temps (chercheur) à NIM IGÜSZI (Centre de calcul du ministère des industries lourdes)*

Maurice Bruynooghe (Leuven)
- *Ingénieur en informatique, KU Leuven*
- *Thèse de master sur la preuve automatique de théorèmes*
- *Thèse en informatique, KU Leuven*



ci-dessus est représentée la pile de backtraking de l'exemple II-2
Les résolvantes sont les suivantes (successivement) R_1, R_2, R_3, R_4 (voir II-2

Un langage, deux (?) implémentations

Un compilateur pour le DEC-10

USER'S GUIDE to DECsystem-10 PROLOG

September 1978

Luis Moniz Pereira

Divisao de Informatica
Laboratorio Nacional
de Engenharia Civil
Lisbon

Fernando C N Pereira & David H D Warren

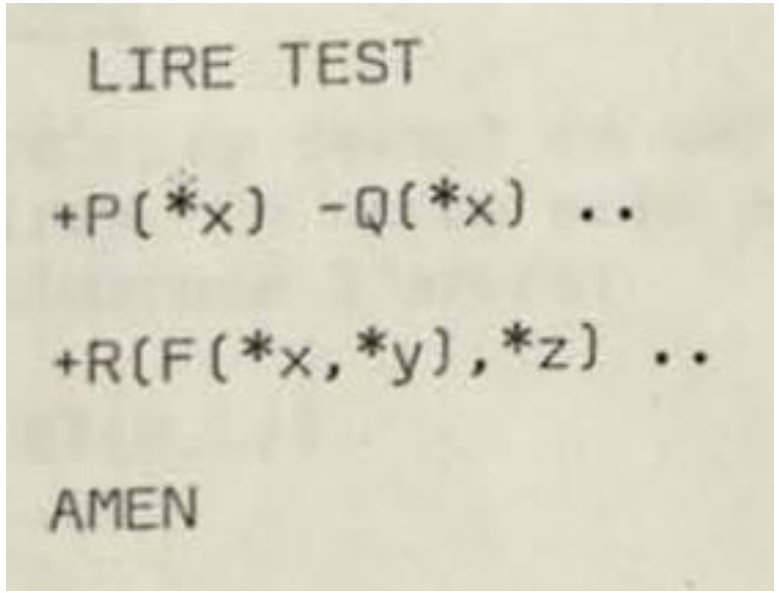
Department of Artificial Intelligence
University of Edinburgh

NB. This issue of the User's Guide describes the DECsystem-10 Prolog interpreter version 1.32 and compiler version 1.11. Comments and suggestions are welcomed.

User's guide to DEC-10 PROLOG, Pereira, Pereira, Warren, 1978

Un langage, deux syntaxes

Marseille PROLOG



```
LIRE TEST

+P(*x) -Q(*x) ..

+R(F(*x,*y),*z) ..

AMEN
```

Robert Pasero, Representation du Francais en logique du premier ordre en vue de dialoguer avec un ordinateur, 1973

Marseille Prolog :

```
+density(*C,*D) -pop(*C,*P) -area(*C,*A) -(D=*P·1000/A)

demontrer(density(France, X))
```

Edimbourg Prolog :

```
density(C,D) :- pop(C,P), area(C,A), D is (P×1000)/A.

?- density(france, X).          X= 246
```

D. Warren, L. Pereira and F. Pereira. Prolog - the Language and its Implementation compared with Lisp, Slides, 1977

Une syntaxe permettant de positionner Prolog comme langage

PROLOG - THE LANGUAGE AND ITS IMPLEMENTATION COMPARED WITH LISP

stricted to lists. Let us therefore start by translating some elementary Lisp functions into Prolog:-

```
append[x;y]=
   [null[x] -> y;
    T -> cons[car[x];append[cdr[x];y]]]

nreverse[x]=
   [null[x] -> nil;
    T -> append[nreverse[cdr[x]];cons[car[x];
                                      nil]]]
```

These functions for list concatenation and (naive) list reversal are equivalent to the following two Prolog procedures:

```
append(nil,Y,Y).
append((A.B),Y,(A.B1)) :- append(B,Y,B1).

nreverse(nil,nil).
nreverse((A.B),Y):-
nreverse(B,YO),append(YO,(A.nil),Y).
```

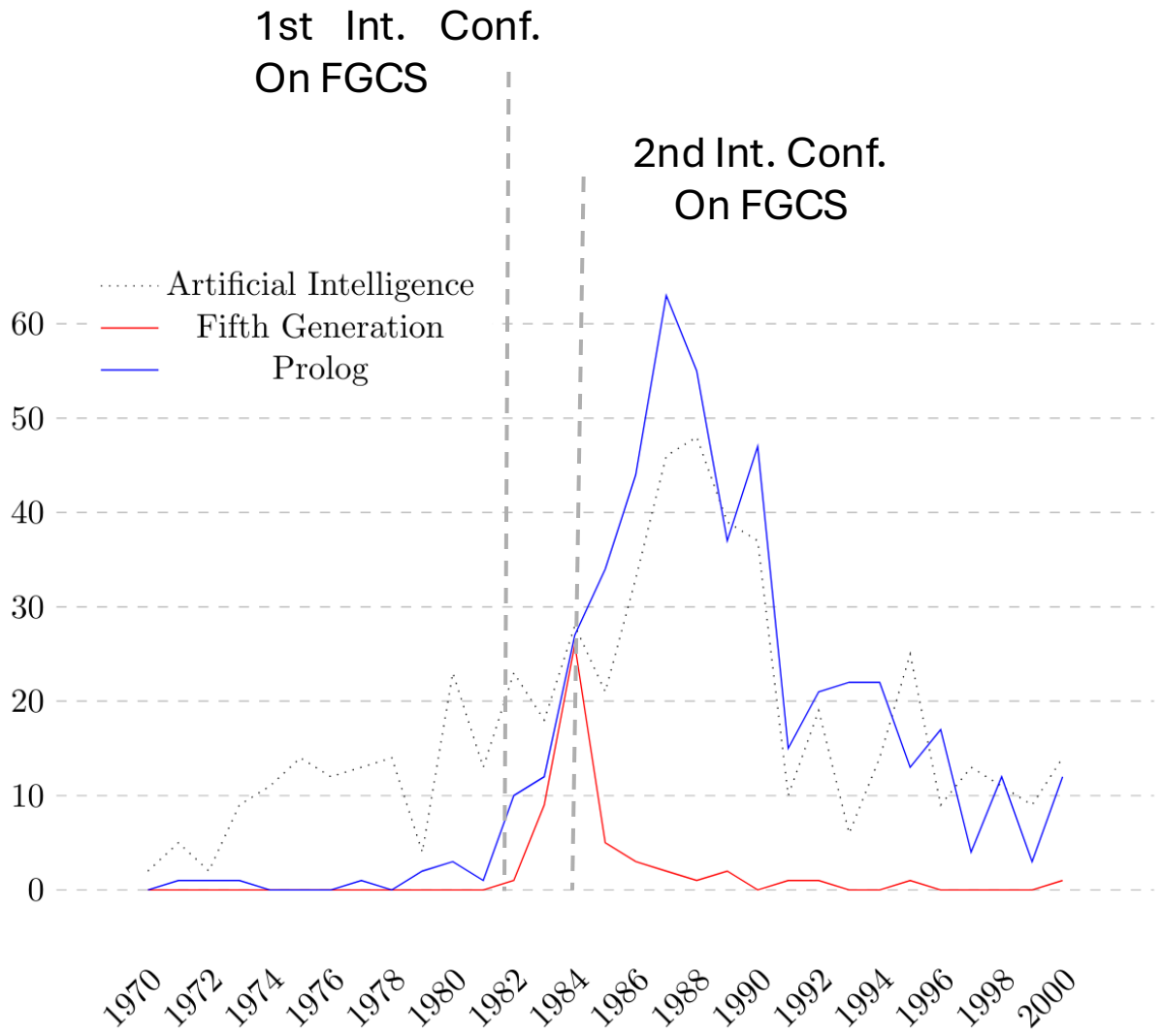
D. Warren, L. Pereira and F. Pereira. Prolog - the Language and its Implementation compared with Lisp, 1977

Prolog devient un logiciel

1980-1985

le cnam

Le "Fifth generation computer system" japonais, première course à l'IA



Publication count on the ACM digital library based on presence of given key-word in article abstract

JAPAN LEAPS THE GAP

Whatever Western critics may say, the truth about Japan's Icot project is that its top researchers have quickly developed the beginnings of a credible fifth generation computer. Liz Else reports

Icot director Kayuhiro Fuchi stresses the need for an overall picture of research in the IT field.

Over the past four years the Japanese have found themselves centre stage in the worldwide effort to build the 'intelligent' computers of the future. But because theirs was the initiative that triggered the fervour, every research blind alley, inter-company wrangle, and budgetary cutback has been seized on by the West as a sign that Japan's greatest foray into basic research is faltering.

In fact the truth is far more complex.

The Institute for New Generation Computer Technology (Icot), the joint government and industry research body which spearheads the fifth generation work, is now seven months into the key intermediate stage of the 10-year programme.

And while there have been genuine problems – both technical and political – there has also been genuine progress, with Icot moving from standing start to the beginning of a credible fifth generation computer in a very few months.

Icot was set up in April 1982

Icot's researchers are divided into three main groups. The first of these covers machine architecture and work is split into three areas: sequential inference machine, relational database machine, and parallel inference machine.

The second group concentrates on software and is also split into three research areas covering work on the Kernel Language (KL), programming environment and operating system, and natural language and knowledge representation.

The third group acts as a sort of inflight corrector, by working on applications in relational database management and expert systems and validating the approach of the theory propounded in the rest of Icot and its affiliates.

Icot's fifth generation

Equally importantly, the whole Icot programme was predicated on a quantum leap beyond the increasing limitations of von Neumann architectures to workable parallel architectures.

Underpinning it all was a formidable research infrastructure in Japan, fuelled both by central government intervention and the vast R & D budgets of the private sector, and, above all, a philosophical structure into which the fifth generation 'hypothesis' would fit.

Kazuhiro Fuchi, Icot's director, made this very clear last November during a conference to review the progress of the first phase.

'Our basic concept is intended to grasp the progress and future direction of research in the information processing field and

marketed by Symbolics.

subsequent Icot 'open days' hundreds of visitors were given the opportunity to assess just how far Fuchi and his colleagues had progressed.

One of Icot's major research themes is the development of a knowledge base machine and this has taken the form of the prototype Delta, which was on display at the conference.

It is being expanded to four dual processor relational data base engines with 20 gigabytes storage for the final configuration. According to Fuchi, however, there have been problems which have shown up during the recent evaluation. 'It is a very big model which includes some subsystems. As a compromise we included some conventional computers so the software is now very complicated. Maybe we need to redesign

Unknown source, circa 1982

le cnam

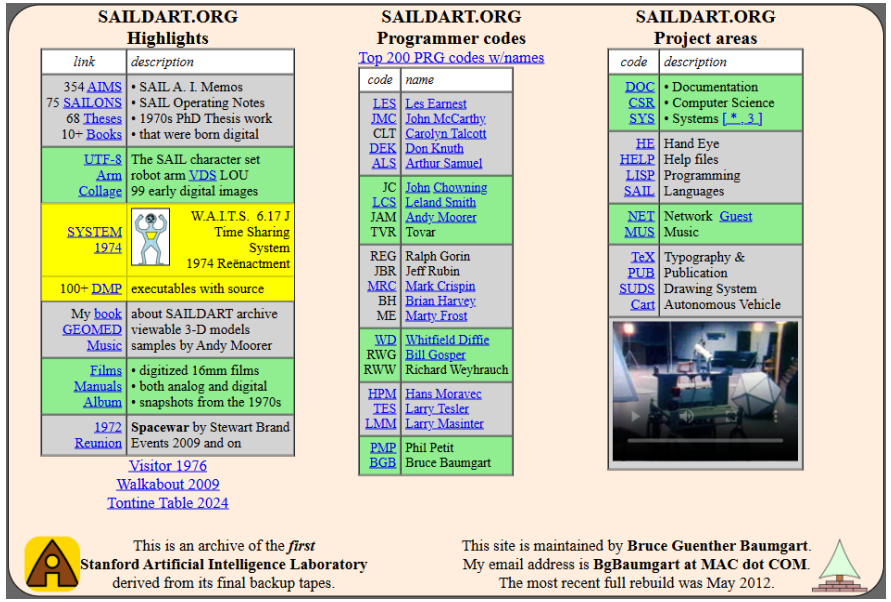
Le logiciel en tant que produit packagé (Haigh 2013)

- « Ware » au sens d'un produit pouvant être partagé et distribué
- Une certaine forme d'unicité / d'intégrité dans le corpus du code source
- Fourni avec de la documentation et des connaissances tacites

Mahoney, M. S. (2008). What makes the history of software hard. IEEE Annals of the History of Computing, 30(3), 8-18.

Haigh, T. (2013). Software and souls; Programs and packages. *Communications of the ACM*, *56*(9), 31-34.

4 bandes magnétiques



SAILDART.ORG
Highlights

link	description
354 AIMS 75 SAILONS 68 Theses 10+ Books	• SAIL A. I. Memos • SAIL Operating Notes • 1970s PhD Thesis work • that were born digital
UTF-8 Arm Collage	The SAIL character set robot arm VDS LOU 99 early digital images
SYSTEM 1974	W.A.I.T.S. 6.17 J Time Sharing System 1974 Reënactment
100+ DMP	executables with source
My book GEOMED Music	about SAILDART archive viewable 3-D models samples by Andy Moorer
Films Manuals Album	• digitized 16mm films • both analog and digital • snapshots from the 1970s
1972 Reunion	**Spacewar** by Stewart Brand Events 2009 and on

Visitor 1976
Walkabout 2009
Tontine Table 2024

SAILDART.ORG
Programmer codes

Top 200 PRG codes w/names

code	name
LES	Les Earnest
JMC	John McCarthy
CLT	Carolyn Talcott
DEK	Don Knuth
ALS	Arthur Samuel
JC	John Chowning
LCS	Leland Smith
JAM	Andy Moorer
TVR	Tovar
REG	Ralph Gorin
JBR	Jeff Rubin
MRC	Mark Crispin
BH	Brian Harvey
ME	Marty Frost
WD	Whitfield Diffie
RWG	Bill Gosper
RWW	Richard Weyhrauch
HPM	Hans Moravec
TES	Larry Tesler
LMM	Larry Masinter
PMP	Phil Petit
BGB	Bruce Baumgart

SAILDART.ORG
Project areas

code	description
DOC CSR SYS	• Documentation • Computer Science • Systems [*, 3]
HE HELP LISP SAIL	Hand Eye Help files Programming Languages
NET MUS	Network Guest Music
TeX PUB SUDS Cart	Typography & Publication Drawing System Autonomous Vehicle

This is an archive of the ***first***
Stanford Artificial Intelligence Laboratory
derived from its final backup tapes.

This site is maintained by **Bruce Guenther Baumgart**.
My email address is **BgBaumgart at MAC dot COM**.
The most recent full rebuild was May 2012.

Bruce Baumgart, *https://www.saildart.org/*

Snapshot of the content of a magnetic tape, as preserved on saildart.org

buttons:[prev][next]

[datetime]	[Filnam.ext]	[prj,prg]	rev	[size]
1981-07-29 14:19	0 .EXT	[PRO,SYS]	1	1620
1981-07-29 14:19	0 .FNS	[PRO,SYS]	1	4660
1982-02-17 10:01	3 .CCL	[PRO,SYS]	1	165
1982-02-17 09:50	BUGS .TXT	[PRO,SYS]	1	670
1982-02-17 09:50	COMMAC.CCL	[PRO,SYS]	1	250
1982-02-17 09:50	COMMON.LIB	[PRO,SYS]	1	124150
1982-02-17 09:50	COMMON.LIB	[PRO,SYS]	2	124150
1982-02-17 09:51	COMMON.MAC	[PRO,SYS]	1	1775
1982-02-17 09:51	COMMON.MIC	[PRO,SYS]	1	235
1982-02-17 09:51	COMMON.REL	[PRO,SYS]	1	4015
1982-02-17 09:51	COMMON.REL	[PRO,SYS]	2	4015
1982-02-17 09:51	COMP .MIC	[PRO,SYS]	1	125
1982-02-17 09:51	COMPIL.LIB	[PRO,SYS]	1	216520
1982-02-17 09:53	COMPIL.MIC	[PRO,SYS]	1	210
1982-02-17 09:53	COMPIL.MIC	[PRO,SYS]	2	210
1982-02-17 09:53	CTRAP .MAC	[PRO,SYS]	1	2200
1982-02-17 09:53	CTRAP .MAC	[PRO,SYS]	2	2200
1982-02-17 09:53	CTRAP .REL	[PRO,SYS]	1	1040
1982-02-17 09:53	DBOTH .MIC	[PRO,SYS]	1	120
1981-07-29 14:19	DEBUG .MEM	[PRO,SYS]	1	56025
1982-02-17 09:53	DINT .MIC	[PRO,SYS]	1	115
1982-02-17 09:53	EDIT . PL	[PRO,SYS]	1	1430
1982-02-17 09:53	EDIT . PL	[PRO,SYS]	2	1430
1982-02-17 09:53	EDITS .TXT	[PRO,SYS]	1	8310
1982-02-17 09:53	EQS2 .MAC	[PRO,SYS]	1	490
1981-07-29 14:19	EQS2 .UNV	[PRO,SYS]	1	450
1982-02-17 09:53	FILL .	[PRO,SYS]	1	150
1982-02-17 09:53	GARBGE.MAC	[PRO,SYS]	1	11600
1982-02-17 09:53	GARBGE.MAC	[PRO,SYS]	2	11600
1982-02-17 10:02	GARBGE.REL	[PRO,SYS]	1	5335
1981-07-29 14:19	GUIDE3.MEM	[PRO,SYS]	1	91005
1981-07-29 14:19	GUIDE3.MEM	[PRO,SYS]	2	91005
1982-02-17 10:02	HCOMP .MIC	[PRO,SYS]	1	150
1982-02-17 10:02	HDINT .MIC	[PRO,SYS]	1	135
1982-02-17 10:02	HINT .MIC	[PRO,SYS]	1	195
1982-02-17 10:02	HVER .MIC	[PRO,SYS]	1	260
1982-02-17 10:02	IDENS2.MAC	[PRO,SYS]	1	515
1982-02-17 10:02	IDENS2.UNV	[PRO,SYS]	1	465
1982-02-17 10:02	IDENSF.MAC	[PRO,SYS]	1	800
1982-02-17 10:02	IDENSF.REL	[PRO,SYS]	1	620
1982-02-17 10:02	IDENSF.UNV	[PRO,SYS]	1	675
1982-02-17 10:02	INDIR .MAC	[PRO,SYS]	1	1530
1982-02-17 10:02	INDIR .REL	[PRO,SYS]	1	1350
1982-02-17 10:02	INDIR .REL	[PRO,SYS]	2	1350
1982-02-17 10:02	INT .MIC	[PRO,SYS]	1	155
1982-02-17 10:02	INTERP.LIB	[PRO,SYS]	1	141800
1982-02-17 10:02	INTERP.LIB	[PRO,SYS]	2	141800
1982-02-17 10:03	INTERP.MIC	[PRO,SYS]	1	195
1982-02-17 10:03	INTMAC.CCL	[PRO,SYS]	1	30
1982-02-17 10:03	IOLIB2.MAC	[PRO,SYS]	1	7825
1982-02-17 10:03	IOLIB2.MAC	[PRO,SYS]	2	7825
1982-02-17 10:03	IOLIB2.REL	[PRO,SYS]	1	2685
1982-02-17 10:03	IOLIB2.REL	[PRO,SYS]	2	2685
1982-02-17 10:03	JADLN0. PL	[PRO,SYS]	1	6675
1982-02-17 10:03	JADMN0. PL	[PRO,SYS]	1	6675
1982-02-17 10:03	JADMN1. PL	[PRO,SYS]	1	4055
1982-02-17 10:03	JADMN1. PL	[PRO,SYS]	2	4055

perm filename 0.EXT[PRO,SYS] blob sn#601365 filedate 1981-07-29 generic text, type T, neo UTF8

```
ext(=,2,pequal).
ext(=..,2,puniv).
ext(==,2,pegalf).
ext(\==,2,pneglf).
ext(?,1,pquest).
ext(arg,3,'$ARGN').
ext(asserta,1,'$ASSTA').
ext(assertz,1,passrt).
ext(assert,1,passrt).
ext(atom,1,patom).
ext(atomic,1,patic).
ext(c,3,'$CONNE').
ext(call,1,'$CALL').
ext(clause,2,'$CLAU2').
ext(close,1,'$PCLOS').
ext(consult,1,pcsult).
ext(core↔image,0,'$CORIM').
ext(display,1,psorte).
ext(erase,1,perase).
ext(eraseall,1,perasa).
ext(fileerrors,0,'$PFLER').
ext(functor,3,'$FUNCT').
ext(gc,0,'$GCON').
ext(gcguide,1,pgarbg).
ext(getvec,0,pgvec).
ext(halt,0,'$EXIT1').
ext(infixop,4,pinfop).
ext(instance,2,pinst).
ext(integer,1,pinteg).
ext('NOLC',0,plcoff).
ext('LC',0,plcon).
ext(length,2,plengt).
ext(listing,0,'$LISTI').
ext(listing,2,'$LISTA').
ext(name,2,pname).
ext(nl,0,pnewli).
ext(nofileerrors,0,'$PNFLE').
ext(nogc,0,'$GCOFF').
ext(notrace,0,ptoff).
ext(numbervars,3,'$NUVAR').
ext(prefixop,3,ppreop).
ext(read,1,pread).
ext(readtokens,1,ptoken).
ext(reconsult,1,prcslt).
ext(record,1,precor).
ext(recorda,1,preca).
ext(recorded,2,precd).
ext(rename,2,'$RENAM').
ext(retract,1,pretra).
ext(save,1,'$PSAVE').
ext(see,1,'$PSEE').
ext(seeing,1,'$PSEEI').
ext(seen,0,'$CLSEE').
ext(statistics,0,'$STATS').
ext(statistics,2,'$STAT2').
ext(suffixop,3,psufop).
ext(syntaxerror,1,'$SYNER').
```

le cnam

Une distribution informelle et décentralisée

le cnam

Un "ware" qui circule

Various indications of origins, 1980-1983, recovered magnetic tape from Stanford AI laboratory

```
USER'S GUIDE TO DECSYSTEM-10 PROLOG  [3]

                Prepared by

        Ernie Davis   and   Udi Shapiro

                December 1980
```

```
/*
This is the routine described in Prolog Digest V1 #13:

Trace - July 8, 1983  written by Russ Abbott and Alan Foonberg
```

```
% Kahn's Prolog interpreter in Concurrent Prolog.  17/11/83
```

```
/* Hanoi program.  Adapted from Yasukawa.  Ehud Shapiro 8/11/83 */
```

```
/* An implementation of bounded buffer communication, based on the paper
Interprocess communication in Concurrent Prolog, by Akikazu Takeuchi
and Koichi Furukawa, Logic Programming Workshop 83. */
```

```
                        PROLOG
                        ======

Prolog {r ett programmeringsspr}k baserat p} logik.
Prolog har ursprungligen utvecklats vid universitetet i Marseille.

Prolog-systemet p} DEC10/20 best}r av en interpretator och en
kompilator som till stora delar {r skrivna i Prolog sj{lvt.
Interpretatorn och kompilatorn har utvecklats vid universitetet
i Edinburgh av David Warren.
```

le cnam

Distribution décentralisée via Usenet et bandes magnétiques

utzoo!decvax!cwruecmp!harr
12/12/1982 16:24:21 UTC

Due to the still numerous requests by mail, I have compiled the
following listing of Prolog systems being distributed in one form
or another. The list is compiled from network jabber and personal
information. Some of the entries are very sketchy. Any deletions,
additions, or changes should be sent back to me. I especially welcome
any information from the sites which are listed here with sketchy
information. If there are many changes, I will post the information
again; otherwise the list will be available via network mail only.
Randolph E. Harr
Case Western Reserve University
decvax!cwruecmp!harr
Prolog versions available:

NAME: DEC-10 Prolog
VERSION: 1.37 Interpreter, 1.14 compiler
SOURCE/OS: Fortran(?)/DEC-10
FEATURES: Interpreter, Compiler
AVAILABILITY: Restricted License per machine
COST: ?
STATUS: Active (?)
CONTACT: Robert Rae
Department of Artificial Intelligence
University of Edinburgh
Forrest Hill
Edinburgh EH1 2QL
Scotland, U.K.
DATED: 1 December 1982
NOTES: Second implementation, one of the most extensive.

NAME: Unix Prolog

"Prolog Systems Available" posted on NET.LANG.PROLOG user group in Dec. 1982, archived on usenetarchives.com

Un code copyrighté

Various copyright mentions, recovered magnetic tape from Stanford AI laboratory

```
Copyright (C) 1982  University of Edinburgh, Dept of Artificial Intelligence
```

```
%% Interpreter for Concurrent Prolog.        (C) Ehud Shapiro 1983.
```

```
/*  COPYRIGHT (C) 1983
    This module is the original work of R.A.O'Keefe done between
    Wednesday February 23 and Friday February 26 1983.  During this
    time I was employed (but not paid) by Systems Designers Ltd,
    but this work was done after 9pm in the evenings and outside
    their premises.  This particular copy was typed on their Systime
    5000, but that was to provide them with a copy for their own use,
    not because I used the machine at any point during the development
    of the code.  I didn't.  Any resemblance between this code and
    other grammar rule preprocessors written in Prolog is due to the
    nature of the problem and the directness of the solution, not to
    my having copied anyone else's code, which I have not done.
*/
```

Mais ouvert aux contributions

```
                        % Current prolog version

prolog←version
     :- display('Prolog-10  version 3.3
Copyright (C) 1981 by D. Warren, F. Pereira and L. Byrd
').


                        % Show the version and copyright message for all
                        %  the component parts of the current system.

version
     :- display('
** This system has been built from: **

'),
        prolog←version,
        tagged(version,version(List,[])),
        other←bits(List),
        fail.

version.



other←bits([]).

other←bits([First|Rest])
     :- ttynl,
        display(First), ttynl,
        other←bits(Rest).



                        % Add another bit to the version message
                        %  Note that these cannot be removed (modulo ddt)

version(Mesg)
     :- atom(Mesg),
        ( untag(version,version(List,[Mesg|Z]))  ;  List = [Mesg|Z] ),
        !,
        tag(version,version(List,Z)).

version(Mesg)
     :- display('! Invalid version message: '),
        display(Mesg), ttynl,
        fail.
```

```
%  (31 May 81)
%
%       Moved initialise stuff in here
%
%       Provided a new system of version messages.
%       When Prolog is first run it will print the prolog version banner
%       and copyright message (se prolog←version).  Prolog will NEVER
%       do this again, or call version/0 (ie NOT after restore, core←image,
%       or reinitialise). This allows the user to have his own banners
%       for his own programs. However version/0 is provided as an evaluable
%       predicate and can be called at any time to establish what the system
%       consists of (and to display the various copyright messages). A new
%       evaluable predicate version/1 is provided so that the user can add his
%       own banners and copyright message. These cannot then be removed, and
%       they will be included whenever version/0 is called from then on.
%       This mechanism is designed to cope with the obvious fact that many
%       systems will be closures of all sorts of bits and pieces (with Prolog
%       at the bottom), and that each bit may have originated from different
%       people. Prolog does not presume to print out all the associated
%       banners and copyright messages, since this is a patently ridiculous
%       thing for an often called interactive system to do.
%
```

Extracts from JMAIN.PL,
Alan Mycroft,1981,
recovered magnetic tape
from Stanford AI
laboratory

Un noyau central de code source et de nombreux modules complémentaires

le cnam

Edimbourg Prolog comme noyau

```
EDITS.TXT[400,447,PROLOG]                          Updated: 17 September 81

This file lists all the edits/changes made to the DEC10 Prolog system.
Entries refer to the date, new version number, and person responsible.

------------------------------------------------------------------------
16 August 81
3.37
Fernando

Changed COMMON.MAC and PLINI2.MAC to define a new global location,
$CCL, which records whether Prolog was entered at offset 0 ($CCL=0)
or at offset 1 ($CCL=-1).

------------------------------------------------------------------------
28 August 81
3.36
Fernando

Changed ORIGIN.MAC and COMMON.MAC to place the block $PROG which defines
the location in the file system of the Prolog system at 140 (octal).

------------------------------------------------------------------------
```

Extract from EDIT.TXT, 1981-1983, recovered magnetic tape from Stanford AI laboratory

Et des extensions écrites en Prolog

Extract from BUFFER.PL, Ehud Shapiro,1983, recovered magnetic tape from Stanford AI laboratory

```
%% Interpreter for Concurrent Prolog.        (C) Ehud Shapiro 1983.
%% that also simulates the Bagel 24/10/83.

:- public
    cp/1,
    cpclause/3,
    unify/2,
    display←counters/0,
    output/0,
    btrace/3,
    trace/2.

:- op(1150,fx,cp).
:- op(450,xf,'?').
:- op(750,yfx,'@').
:- op(700,xfx,':=').

:- fastcode.

cp(A@center) :- !,
    value(bagel,(N,M)),
    N1 is N/2, M1 is M/2,
    cp1(at(A,(N1,M1,north))).
cp(A) :- !,
    cp1(at(A,(1,1,north))).
```

Extract from TRACE.PL, Russ Abbott and Alan Foonberg,1983, recovered magnetic tape from Stanford AI laboratory

```
/*
This is the routine described in Prolog Digest V1 #13:

Trace - July 8, 1983  written by Russ Abbott and Alan Foonberg

This is a trace program similar to that provided by the PROLOG system.  A
term's execution is traced step-by-step by typing "trace(term).", where term
is that term to be traced.  At each step of the trace, you have an option to
enter a different trace mode.  The list of modes can be obtained by answering
this question with a '?'.  A typical trace line looks like this:

        2.   (3)   enter    b(Var, constant)

The number before the period indicates the count, the number in parenthesis
is the depth, the following phrase indicates the port, and the rest is the
current term being traced.  To obtain the full trace as a parameter, type
"trace(term, Full←Trace)."  Full←Trace will contain a list of the trace lines,
which can be printed out formatted by typing
 "trace$←write←term(trace←list(Full←Trace)."  A 'true trace' is also provided
by adding a third parameter to the trace call, and this can be printed in a
manner similar to that used to print the full trace.  The true trace contains
only those calls that succeeded which led to the satsification of the main
goal.  If you have any questions or suggestions, please send network mail to
foonberg@aerospace or abbott@aerospace.

(remember that dollar signs can be eliminated).

*/
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
:-(op(1100, xfy, [or])).
:-(op(1000, xfy, [and])).

function$is←built←in←with$numero$arguments(abolish, 2).
function$is←built←in←with$numero$arguments(abort, 0).
function$is←built←in←with$numero$arguments(acos, 1).
function$is←built←in←with$numero$arguments(arg, 3).
function$is←built←in←with$numero$arguments(asin, 1).
function$is←built←in←with$numero$arguments(assert, 1).
function$is←built←in←with$numero$arguments(assert, 2).
```

Un partage du savoir formel et tacite

le cnam

9. AI programming in Prolog.

All the above examples should have convinced the reader by now that Prolog is the ideal language for AI programming. But only to illustrate that, we show how McSam, a micro version of a Script Applyer Mechanism can be implemented easily in Prolog. Recall that the LISP implementation of McSam is nine pages of code long.

This example is sort of a "low blow" to LISP, since all it

does is drive two unifies, the one that unifies the story with the script, and the one that unifies the variables with their default names. All the rest Prolog does for you. But still, we expect McEli also to be relatively easier to program in Prolog than in LISP. Would you like to try?

Extracts from TUTORI.LPT, Ehud Shapiro, 1980, recovered magnetic tape from Stanford AI laboratory

Extract from TUTORI.PL, Ehud Shapiro, 1980, recovered magnetic tape from Stanforad AI laboratory

```
/* list processing */


member(X,[X|←]).
member(X,[←|L]) :- member(X,L).


duplicate(X,L1,L2) :- member(X,L1), member(X,L2).


append([],L,L).
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).



intersect([],←,[]).
intersect([X|L1],L2,[X|L3]) :- member(X,L2), intersect(L1,L2,L3).
```

Un logiciel libre ?

- L’organisation autour de Prolog est décentralisée et en grande partie informelle ; aucun contrôle ni aucune coordination n’est assuré par les contributeurs d’origine.

- L’unicité du langage repose sur certains de ses éléments fondamentaux (interpréteur/compilateur d’Édimbourg), mais il n’existe pas de frontières techniques clairement définies pour le langage ni de version centralisée unique.

- Il n’existe aucun mécanisme de retour pour améliorer la base de code (public récursif, Kelty 2008).

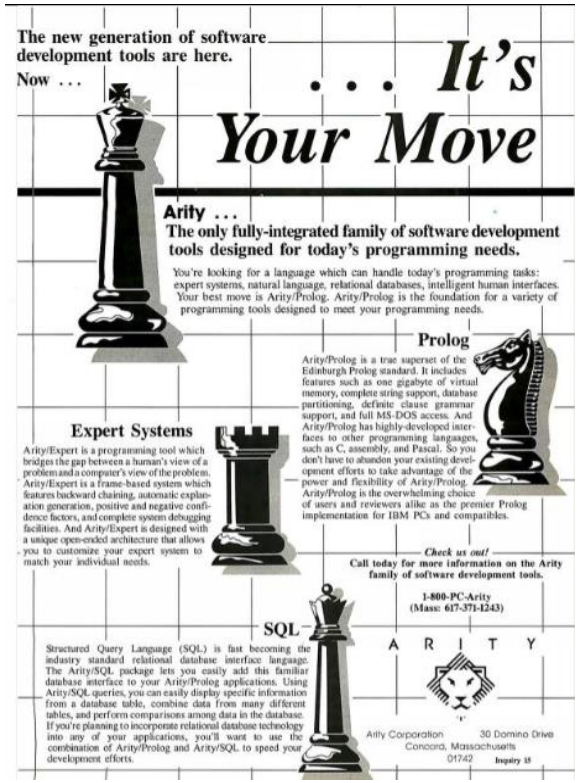
- Il n’existe pas de cadre juridique, institutionnel ou commercial (Campbell-Kelly 2003).

Kelty, C. (2008). Two bits. The Cultural Significance of Free Software. Duke University.
Campbell-Kelly, M. (2003). From airline reservations to sonic the hedgehog. A History of the Software Industry. Cambridge MA., London.

La croissance organique de Prolog a permis une adoption rapide, mais, en l'absence de coordination entre les contributeurs, elle a ouvert le champ à la concurrence du marché.

Byte Magazine Vol 10-04, 1985, Internet Archive

Byte Magazine Vol 12-09, 1987, Internet Archive



Kelty, C. (2008). Two bits. The Cultural Significance of Free Software. Duke University.
Alcaras, G. (2022). Des logiciels libres au contrôle du code: l'industrialisation de l'écriture informatique (Doctoral dissertation, Paris, EHESS).

Conclusion

- *Pas de critère "simple" permettant de définir ce qu'est un langage de programmation*
- *Critères techniques, et formalisation théorique ne sont pas suffisants*
- *Il convient de s'intéresser à l'environnement social et à l'usage fait du langage*

  Un programme de recherche, reflétant les enjeux scientifiques des équipes concernées

  Un artefact portable, qui permet de factoriser un certain nombre de fonctions utiles aux programmeurs d'une communauté donnée et de créer une langue commune

  Un logiciel libre (?), avec ses cannaux de distribution, une certaines unicité, une gestion des licenses

Thanks !

History of CS / AI

Mounier-Kuhn, P. E. (2010). *L'informatique en France de la seconde guerre mondiale au Plan Calcul*. PUPS.

Bertrand, E. (2018). Jacqueline Léon, Histoire de l'automatisation des sciences du langage. Revue d'histoire des sciences humaines, (32), 288-292.

Dupuy, J. P. (2013). Aux origines des sciences cognitives. La découverte.

McKevitt, P. (1997). Daniel Crevier, AI: The Tumultuous History of the Search for Artificial Intelligence. London and New York: Basic Books, 1993. Pp. xiv+ 386. ISBN 0-465-02997-3.£ 17.99, $27.50. The British Journal for the History of Science, 30(1), 101-121.

STS

Leigh Star, S. (2018). L'ethnographie des infrastructures. Tracés. Revue de sciences humaines, (35), 187-206.

MacKenzie, D., & Wajcman, J. (1999). Introductory essay: the social shaping of technology. The social shaping of technology, 2, 3-27.

Latour, B. (1987). Science in action: How to follow scientists and engineers through society. Harvard UP.

Akrich, M. (1992). The de-scription of technical objects. Shaping technology/building society. Studies in sociotechnical change, 205-224.

Merci !

Philosophy

White, G. (2004). The philosophy of computer languages. The Blackwell guide to the philosophy of computing and information, 237-247.

Turner, R. (2014). Programming languages as technical artifacts. Philosophy & technology, 27, 377-397.

Marino, M. C. (2020). Critical code studies. MIT Press.