

# The birth of Prolog

Alain Colmerauer and Philippe Roussel

November 1992

## Abstract

The programming language, Prolog, was born of a project aimed not at producing a programming language but at processing natural languages; in this case, French. The project gave rise to a preliminary version of Prolog at the end of 1971 and a more definitive version at the end of 1972. This article gives the history of this project and describes in detail the preliminary and then the final versions of Prolog. The authors also felt it appropriate to describe the Q-systems since it was a language which played a prominent part in Prolog's genesis.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Part I. The history</b>	<b>3</b>
2.1	1971: The first steps . . . . .	3
2.2	1972: The application that created Prolog . . . . .	6
2.3	1973: The final Prolog . . . . .	9
2.4	1974 and 1975: The distribution of Prolog . . . . .	10
<b>3</b>	<b>Part II. A forerunner of Prolog, the Q-systems</b>	<b>12</b>
3.1	One-way unification . . . . .	12
3.2	Rule application strategy . . . . .	13
3.3	Implementation . . . . .	15
<b>4</b>	<b>Part III. The preliminary Prolog</b>	<b>15</b>
4.1	Reasons for the choice of resolution method . . . . .	16
4.2	Characteristics of the preliminary Prolog . . . . .	18
4.3	Implementation of the preliminary Prolog . . . . .	20
<b>5</b>	<b>Part IV. The final Prolog</b>	<b>21</b>
5.1	Resolution strategy . . . . .	22
5.2	Syntax and Primitives . . . . .	23
5.3	A programming example . . . . .	24
5.4	Implementation of the interpreter . . . . .	28

## 1 Introduction

As is well known, the name ‘Prolog’ was invented in Marseilles in 1972. It was Philippe Roussel chose the name as an abbreviation for ‘PROgrammation en LOGique’ to refer to the software tool designed to implement a man-machine communication system in natural language. It can be said that Prolog was the offspring of a successful marriage between natural language processing and automated theorem-proving. The idea of using a natural language like French to reason and communicate directly with a computer seemed like a crazy idea, yet this was the basis of the project set up by Alain Colmerauer in the summer of ‘70. Alain had some experience in the computer processing of natural languages and wanted to expand his research.

We have now presented the two co-authors of this article – Alain Colmerauer and Philippe Roussel – but clearly, as in any project of this kind, many other people were involved. To remain objective in our account of how Prolog – a language which is now already twenty years old – came to be, we took a second look at all the documents still in our possession, and played the part of historians. To begin with, we followed the chronology in order to present the facts and describe the participants over the time from the summer of 1970 until the end of 1976. This constitutes the first part of the article. The other parts are of a more technical nature. They are devoted to three programming languages which rapidly succeeded one another: Q-systems, conceived for machine translation; the preliminary version of Prolog, created at the same time as its application; and the definitive version of Prolog, created independently of any application.

This paper is not the first to be written about the history of Prolog. We must mention the paper written by Jacques Cohen [9], which directly concerns Prolog, and the paper by Robert Kowalski [27] about the birth of ‘Logic Programming’ as a discipline. Also worthy of attention are the history of automated theorem-proving as described by Donald Loveland [28] and the prior existence of the Absys language, a possible competitor with Prolog in the opinion of E. Elcock [16].

## 2 Part I. The history

At the beginning of July ‘70, Robert Pasero and Philippe arrived in Montreal. They had been invited by Alain who was then Assistant Professor of Computer Science at the University of Montreal and was leading the automatic translation project, TAUM (Traduction Automatique de l’Université de Montréal). All were at turning points in their careers. Robert and Philippe were then 25 years old and had just been awarded teaching positions in Computer Science at the new Luminy Science Faculty. Alain was 29 years old and, after a 3-year stay in Canada, was soon to return to France.

During their two-month stay in Montreal, Robert and Philippe familiarized themselves with the computer processing of natural languages. They wrote

several non-deterministic context-free analyzers in Algol 60 and a French paraphrase generator using Q-systems, the programming language which Alain had developed for the translation project (see Part II).

At the same time, Jean Trudel, a Canadian researcher and a doctoral student of Alain's, had chosen to work on automated theorem-proving. His point of reference was Alan Robinson's article [36] on the resolution principle. It was a difficult article to understand in 1970 but Jean had the advantage of having taken a course in logic given by Martin Davis in New York. He had already developed a complete theorem prover in which unification was written in an up-to-date programming style: all the computations consisted in modifying pointers.

## 2.1 1971: The first steps

At the beginning of 1971, everyone was back in Marseilles. Alain had obtained a teaching position in Computer Science (*maître de conférence*) and Jean Trudel had been able to accompany him, thanks to a two-year grant from Hydro-Quebec. The aim of the project was to make deductions based on texts written in French. The work was divided up as follows: Jean Trudel and Philippe worked on the deduction part, Robert Pasero and Alain on the natural language part.

We had access to a machine at the Marseilles university computing center: an IBM 360-44 installed in our department, offering approximately 900Kb of internal memory and equipped with an operating system without virtual memory. By means of an interactive monitor developed by Jean Trudel and by using the machine at night to take advantage of all the available memory, we were able to work in conditions which, in France at that time, were exceptionally good: practically 1Mb of memory to execute the programs and an interactive mode for communications between users and programs via the operator's console.

Jean Trudel improved his prover and then, beginning in May, Philippe produced a whole series of provers written in Algol-W. A primitive natural-language communication system was then developed by the entire team [12]. The interfaces between the logical formulæ and French consisted of 50 Q-system rules for the input and 17 Q-system rules for the output. The reasoning part was implemented through one of Philippe's provers. It was thus possible to have the

following conversation with the computer:

User

Cats kill mice.

Tom is a cat who does not like mice who eat cheese.

Jerry is a mouse who eats cheese.

Max is not a mouse.

What does Tom do?

Computer

Tom does not like mice who eat cheese.

Tom kills mice.

User

Who is a cat?

Computer

Tom.

User

What does Jerry eat?

Computer

Cheese.

User

Who does not like mice who eat cheese?

Computer

Tom.

User

What does Tom eat?

Computer

What cats who do not like mice who eat cheese eat.

The logical formulas created made use of: constants representing elements,  
*Tom, Jerry, Max, Cheese*;

constants representing sets,

*Cats, Mice, MiceWhoEatCheese, CatsWhoDoNotLikeMiceWhoEatCheese*;

constants representing binary relations between sets,

*Kill, DoesNotLike, Eat*;

a functional symbol of arity 1 and two relational symbols of arity 2 and 3,

*The, Subset, True*.

A term of the form *The(a)* was taken to represent the set consisting only of the element *a*. A formula of the form *Subset(x, y)* expressed the inclusion of set *x* in set *y* and a formula of the form *True(r, x, y)* expressed that the sets *x* and *y* were in the relation *r*. To the clauses that encode the sentences, Jean Trudel added four clauses relating the three symbols *The, Subset, True*:

$(\forall x)[Subset(x, x)],$

$(\forall x)(\forall y)(\forall z)[Subset(x, y) \wedge Subset(y, z) \Rightarrow Subset(x, z)],$

$(\forall a)(\forall b)[Subset(The(a), The(b)) \Rightarrow Subset(The(b), The(a))],$

$(\forall x)(\forall y)(\forall r)(\forall x')(\forall y')[True(r, x, y) \wedge Subset(x, x') \wedge Subset(y, y') \Rightarrow True(r, x', y')].$

The main problem was to avoid untimely production of inferences due to the transitivity and reflexivity axioms of the inclusion relation *Subset*.

While continuing his research on automated theorem-proving, Jean Trudel came across a very interesting method: SL-resolution [24]. He persuaded us to invite one of its inventors, Robert Kowalski, who came to visit us for a week in June 1971. It was an unforgettable encounter. For the first time, we talked to a specialist in automated theorem-proving who was able to explain the resolution principle, its variants and refinements. As for Robert Kowalski, he met people who were deeply interested in his research and who were determined to make use of it in natural language processing.

While attending an IJCAI convention in September '71 with Jean Trudel, we met Robert Kowalski again and heard a lecture by Terry Winograd on natural language processing. The fact that he did not use a unified formalism left us puzzled. It was at this time that we learned of the existence of Carl Hewitt's programming language, Planner [20]. The lack of formalization of this language, our ignorance of Lisp and, above all, the fact that we were absolutely devoted to logic meant that this work had little influence on our later research.

## 2.2 1972: The application that created Prolog

The year 1972 was the most fruitful one. First of all, in February, the group obtained a grant of 122,000 FF (at that time about \$20,000) for a period of 18 months from the Institut de Recherche d'Informatique et d'Automatique, a computer research institution affiliated with the French Ministry of Industry. This contract made it possible for us to purchase a teletype terminal (30 characters per second) and to connect it to the IBM 360-67 (equipped with the marvelous operating system CP-CMS, which managed virtual machines) at the University of Grenoble using a dedicated 300 baud link. For the next three years, this was to be by far the most convenient computing system available to the team; everyone used it, including the many researchers who visited us. It took us several years to pay Grenoble back the machine-hour debt thus accumulated. Finally, the contract also enabled us to hire a secretary and a researcher, Henry Kanoui, a post-graduate student who would work on the French morphology. At his end, Kowalski obtained funding from NATO, which financed numerous exchanges between Edinburgh and Marseilles.

Of all the resolution systems implemented by Philippe, the SL-resolution of R. Kowalski and D. Kuehner seemed to be the most interesting. Its stack type operating mode was similar to the management of procedure calls in a standard programming language and was thus particularly well-suited to processing non-determinism by backtracking à la Robert Floyd [18] rather than by copying and saving the resolvents. SL-resolution then became the focus of Philippe's thesis on the processing of formal equality in automated theorem-proving [34]. Formal equality is less expressive than standard equality but it can be processed more efficiently. Philippe's thesis would lead to the introduction of the dif predicate (for  $\neq$ ) into the very first version of Prolog.

We again invited Robert Kowalski but this time for a longer period; April and May. Together, we then all had more computational knowledge of automated theorem-proving. We knew how to axiomatize small problems (addition of integers, list concatenation, list reversal, etc.) so that an SL-resolution prover computed the result efficiently. We were not, however, aware of the Horn clause

paradigm; moreover Alain did not yet see how to do without Q-systems as far as natural language analysis was concerned.

After the departure of Robert, Alain ultimately found a way of developing powerful analyzers. He associated a binary predicate  $N(x, y)$  with each non-terminal symbol  $N$  of the grammar, signifying that  $x$  and  $y$  are terminal strings for which the string  $u$  defined by  $x = uy$  exists and can be derived from  $N$ . By representing  $x$  and  $y$  by lists, each grammar rule can then be encoded by a clause having exactly the same number of literals as occurrences of non-terminal symbols. It was thus possible to do without list concatenation. (This technique is now known as the difference lists technique). Alain also introduced additional parameters into each non-terminal to propagate and compute information. As in Q-systems, the analyzer not only verified that the sentence is correct but also extracted a formula representing the information that it contains. Nothing now stood in the way of the creation of a man-machine communication system entirely in 'logic'

A draconian decision was made: at the cost of incompleteness, we chose linear resolution with unification only between the heads of clauses. Without knowing it, we had discovered the strategy which is complete when only Horn clauses are used. Robert Kowalski [25] demonstrated this point later and together with Maarten van Emden he would go on to define the modern fixed point semantics of Horn clause programming [26].

During the fall of 1972, the first Prolog system was implemented by Philippe in Niklaus Wirt's language Algol-W; in parallel, Alain and Robert Pasero created the eagerly awaited man-machine communication system in French [13]. There was constant interaction between Philippe, who was implementing Prolog, and Alain and Robert Pasero, who programmed in a language which was being created step by step. This preliminary version of Prolog is described in detail in Part III of this paper. It was also at this time that the language received its definitive name following a suggestion from Philippe's wife based on keywords which had been given to her.

The man-machine communication system was the first large Prolog program ever to be written [13]. It had 610 clauses: Alain wrote 334 of them, mainly the analysis part; Robert Pasero 162, the purely deductive part, and Henry Kanoui wrote a French morphology in 104 clauses, which makes possible the link between the singular and plural of all common nouns and all verbs, even irregular ones, in the third person singular present tense. Here is an example of a text submitted to the man-machine communication system in 1972:

```
Every psychiatrist is a person.  
Every person he analyzes is sick.  
Jacques is a psychiatrist in Marseille.  
Is Jacques a person?  
Where is Jacques?  
Is Jacques sick?
```

and here are the answers obtained for the three questions at the end:

```
Yes. In Marseille.  
I don't know.
```

The original text followed by the three answers was in fact as follows:

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?  
OUI. A MARSEILLE. JE NE SAIS PAS.

All the inferences were made from pronouns (he, she, they, etc.), articles (the, a, every, etc.), subjects and complement structures with or without prepositions (from, to, etc.). In fact, the system knew only about pronouns, articles and prepositions (the vocabulary was encoded by 164 clauses), it recognized proper nouns from the mandatory asterisk which had to precede them as well as the verbs and common nouns on the basis of the 104 clauses for French morphology.

In November, together with Robert Pasero, we undertook an extensive tour of the American research laboratories after a visit in Edinburgh. We took with us a preliminary report on our natural language communication system and our very first Prolog. We left copies of the report almost everywhere. Jacques Cohen welcomed us in Boston and introduced us at MIT, where we received a warm welcome and talked with Minsky, Charniak, Hewitt and Winograd. We also visited Woods at BBN. We then went to Stanford, visited the SRI and John McCarthy's AI laboratory, met Cordell Green, presented our work to a very critical J. Feldman and spent Thanksgiving at Robert Floyd's home.

### 2.3 1973: The final Prolog

At the beginning of the year or, to be exact, in April, our group attained official status. The CNRS recognized us as an 'associated research team' entitled 'Man-machine dialogue in natural language' and provided financial support in the amount of 39,000 FF (about \$6,500) for the first year. This sum should be compared to the 316,880 FF (about \$50,000) we received in October from IRIA to renew the contract for 'man-machine communication in natural language with automated deduction' for a period of two and a half years.

Users of the preliminary version of Prolog at the laboratory had now done sufficient programming for their experience to serve as the basis for a second version of Prolog, a version firmly oriented toward a programming language and not just a kind of automated deductive system. Besides the communication system in French in 1972, two other applications had been developed using this initial version of Prolog: a symbolic computation system [3, 4, 22] and a general problem-solving system called Sugiton [21]. Also, Robert Pasero continued to use it for his work on French semantics, leading to the successful completion of his thesis in May [32].

Between February and April 1973, at the invitation of Robert Kowalski, Philippe visited the School of Artificial Intelligence at the University of Edinburgh, which was within the Department of Computational Logic directed by Bernard Meltzer. Besides the many discussions with the latter and with David Warren, Philippe also met Roger Boyer and Jay Moore. They had constructed an implementation of resolution using an extremely ingenious method based on

a structure-sharing technique to represent the logical formulæ generated during a deduction. The result of this visit and the laboratory's need to acquire a true programming language prompted our decision to lay the foundations for a second Prolog.

In May and June 1973, we laid out the main lines of the language, in particular the choice of syntax, basic primitives and the interpreter's computing methods, all of which tended toward a simplification of the initial version. From June to the end of the year, G  rard Battani, Henry M  loni and Ren  e Bazzoli, postgraduate students at the time, wrote the interpreter in Fortran and its supervisor in Prolog.

As the reader will see from the detailed description of this new Prolog in Part IV of this paper, all the new basic features of current Prologs were introduced. We observe in passing that this was also the time when the 'occur check' disappeared as it was found to be too costly.

## 2.4 1974 and 1975: The distribution of Prolog

The interactive version of Prolog which operated at Grenoble using teletype was in great demand. David Warren, who stayed with us from January to March, used it to write his plan generation system, Warplan [38]. He notes:

'The present system is implemented partly in Fortran, partly in Prolog itself and, running on an IBM 360-67, achieves roughly 200 unifications per second.'

Henry Kanoui and Marc Bergman used it to develop a symbolic manipulation system of quite some size called Sycophante [5, 23]. G  rard Battani and Henry Meloni used it to develop a speech recognition system enabling questions to be asked of the IBM operating system CP-CMS at Grenoble [2, 30]. The interface between the acoustic signal and the sequence of phonemes it represented was borrowed from the CNET at Lannion and was obviously not written in Prolog.

Early in 1975, Alain Colmerauer had completely rewritten the supervisor keeping the infix operator declarations in Prolog but adding a compiler of the so-called 'metamorphosis' grammars. This time, in contrast to Ren   Bazzoli, he used a top-down analyzer to read the Prolog rules. This was a good exercise in meta-programming. David Warren later included grammar rules of this sort in his compiled version of Prolog [39] and together with Fernando Pereira rechristened a simplified variant of metamorphosis grammars with the name, 'definite clause grammars' [33]. Metamorphosis grammars enabled parameterized grammar rules to be written directly as they were in Q-systems. The supervisor compiled these rules into efficient Prolog clauses by adding two additional parameters. To prove the efficiency and expressiveness of the metamorphosis grammars, Alain wrote a small model compiler from an Algol-style language to a fictitious machine language and a complete man-machine dialogue system in French with automated deductions. All this work was published [14], along with the theoretical foundations of the metamorphosis grammars.

G  rard Battani and Henry M  loni were kept very busy with the distribution of Prolog. They sent it to Budapest, Warsaw, Toronto, Waterloo (Canada) and traveled to Edinburgh to assist David Warren in installing it on a PDP 10. A former student, H  l  ne Le Gloan, installed it at the University of Montreal. Michel Van Caneghem did the same at the IRIA in Paris before coming to work



with us. Finally, Maurice Bruynooghe took Prolog to Leuven (Belgium) after a three-month stay in Marseilles (October through December 1975).

Indeed, as David Warren has pointed out, Prolog spread as much, or more, by people becoming interested and taking away copies either directly from Marseilles or from intermediaries such as Edinburgh. Thus, Prolog was not really distributed; rather it ‘escaped’ and ‘multiplied’.

During 1975, the whole team carried out the porting of the interpreter onto a 16-bit mini-computer: the T1600 from the French company, Télémécanique. The machine only had 64K bytes and so a virtual memory management system had to be specially written. Pierre Basso undertook this task and also won the contest for the shortest instruction sequence that performs an addressing on 32 bits while also testing the page default. Each laboratory member then received two pages of Fortran to translate into machine language. The translated fragments of program were reassembled and it worked! After 5 years, we at last had our very own machine and, what is more, our cherished Prolog ran; slowly, but it ran all the same.

### 3 Part II. A forerunner of Prolog, the Q-systems

The history of the birth of Prolog thus comes to a halt at the end of 1975. We now turn to more technical aspects and, first of all, describe the Q-systems, the result of a first gamble: to develop a very high-level programming language, even if the execution times it entailed might seem bewildering [11]. That gamble, and the experience acquired in implementing the Q-systems was determinative for the second gamble: Prolog.

#### 3.1 One-way unification

A Q-system consists of a set of rewriting rules dealing with sequences of complex symbols separated by the sign +. Each rule is of the form

$$e_1 + e_2 + \dots + e_m \rightarrow f_1 + f_2 + \dots + f_n$$

and means: in the sequence of trees we are manipulating, any sub-sequence of the form  $e_1 e_2 \dots e_m$  can be replaced by the sub-sequence  $f_1 f_2 \dots f_n$ . The  $e_i$ 's and the  $f_i$ 's are parenthesized expressions representing trees, with a strong resemblance to present Prolog terms but using three types of variables. Depending on whether the variable starts with a letter in the set  $\{A, B, C, D, E, F\}$ ,  $\{I, J, K, L, M, N\}$  or  $\{U, V, W, X, Y, Z\}$  it denotes either a label, a tree or a (possibly empty) sequence of trees separated by commas. For example, the rule

$$P + A*(X*, I*, Y*) \rightarrow I* + A*(X*, Y*)$$

(variables are followed by an asterisk) applied to the sequence

$$P + Q(R, S, T) + P$$

produces three possible sequences

$$R + Q(S, T) + P,$$

$$S + Q(R, T) + P,$$

$$T + Q(R, S) + P.$$

The concept of unification was therefore already present but it was one-way only; the variables appeared in the rules but never in the sequence of trees which was being transformed. However, unification took account of the associativity of the concatenation and, as in the above example, could produce several results.

### 3.2 Rule application strategy

This relates to the unification part. Concerning the rule application strategy, Alain Colmerauer [11] wrote:

‘It is difficult to use a computer to analyze a sentence. The main problem is combinatorial in nature: taken separately, each group of elements in the sentence can be combined in different ways with other groups to form new groups which can in turn be combined again and so on. Usually, there is only one correct way of grouping all the elements but to discover it, all the possible groupings must be tried. To describe this multitude of groupings in an economical way, I use an oriented graph in which each arrow is labeled by a parenthesized expression representing a tree. A Q-system is nothing more than a set of rules allowing such a graph to be transformed into another graph. This information may correspond to an analysis, to a sentence synthesis or to a formal manipulation of this type.’

For example the sequence

$A + A + B + B + C + C$

is represented by the graph



and the application of the 4 rules

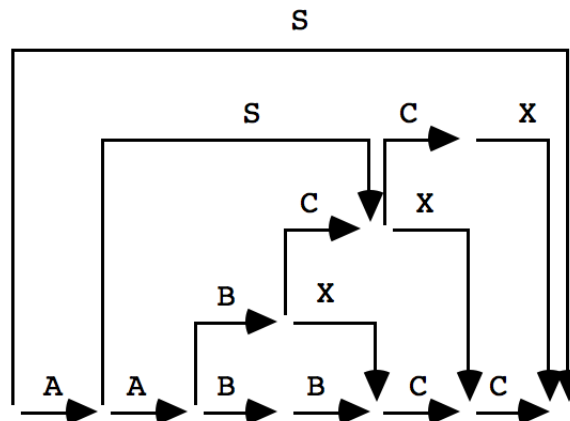
$A + B + C \rightarrow S$

$A + S + X + C \rightarrow S$

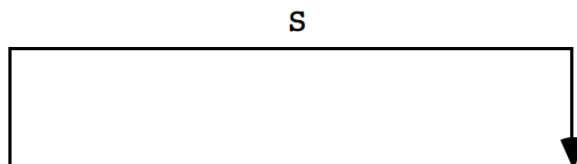
$X + C \rightarrow C + X$

$B + B \rightarrow B + X$

produces the graph



One retains all the paths that lead from the entry point to the end point and do not contain any arrows used in the production of other arrows. One thus retains the unique arrow



i.e. the sequence reduced to the single symbol S.

This procedure is relatively efficient, since it retains the maximum number of common parts in all the sequences. Another aspect of the Q-systems is that they can be applied one after the other. Each one takes as input the graph resulting from the previous system. This technique was widely used in the automatic translation project, where an English sentence would undergo no fewer than fifteen Q-systems before being translated into French. Two Q-systems dealt with morphology, another with the analysis of English, two more with the transfer from an English structure to a French structure, one with the synthesis of French, and nine with French morphology [37].

Let us draw attention to the reversibility of the Q-systems. The rewriting sign that we have represented by  $\rightarrow$  was in fact written  $\Rightarrow$  and, depending on which specified option was chosen at the start of the program, it was interpreted as either a rewriting from left to right or from right to left. That is, the same program could be used to describe a transformation and its reverse transformation such as the analysis and the synthesis of a sentence.

It is interesting to note that in contrast to the analyzers written in Prolog that used a top-down strategy, the analyzers written in Q-systems used a bottom-up strategy. In fact, Alain had extensive experience with this type of strategy. The subject of his thesis done in Grenoble (France) had been bottom-up analyzers which, however, operated by backtracking [10]. Non-determinism was then reduced by precedence relations very similar to those of Robert Floyd [17]. In addition, just before developing the Q-systems, and still as part of the automatic translation project, Alain had written an analyzer and a general synthesizer for W-grammars, the formalism introduced by A. van Wijngaarden to describe Algol 68 [40]. Here again a bottom-up analyzer was used to find the structure of complex symbols (defined by a meta-grammar) as well as a second bottom-up analyzer for analysis of the text itself [7].

### 3.3 Implementation

The Q-systems were written in Algol by Alain Colmerauer and were operational by October '69. Michel van Caneghem and François Stellin, then completing a master's degree, developed a Fortran version, and Gilles Steward developed an ultra-fast version in machine language for the CDC 6400 computer of the University of Montreal.

These Q-systems were used by the entire TAUM project team to construct a complete chain of automatic English-French translations. The English mor-

phology was written by Brian Harris, Richard Kittredge wrote a substantial grammar for the analysis of English, Gilles Stewart wrote the transfer phase, Jules Danserau wrote the grammar for the French synthesis and Michel van Caneghem developed a complete French morphology [37]. The Q-systems were also used a few years later to write the METEO system, a current industrial version of which produces daily translations of Canadian weather forecasts from English into French.

## 4 Part III. The preliminary Prolog

Let us now turn to the preliminary version of Prolog which was created in the fall of 1972 in connection with the development of the man-machine communication system [13].

Recall that the objective we had set for ourselves was extremely ambitious: to have a tool for the syntactic and semantic analysis of natural language, by considering first order logic not only as a programming language but also as a knowledge representation language. In other words, the logical formulation was to serve not only for the different modules of the natural language dialogue system but also for the data exchanged between them, including the data exchanged with the user.

The choice having settled on first order logic (in clausal form) and not on a higher order logic, it then seemed that we were faced with an insurmountable difficulty in wanting programs to be able to manipulate other programs. Of course, this problem was solved using non-logical mechanisms. In fact, this initial version of Prolog was conceived more as an application tool than as a universal programming language. Nevertheless, the basis for such a language was already there.

### 4.1 Reasons for the choice of resolution method

The success of the project hinged on the decision concerning the choice of the logic system and on the basic inference mechanism to be adopted. Although Robinson's resolution principle naturally suggested itself by the simplicity of its clausal form, the uniqueness of the inference rule and its similarity with procedure calls in standard languages, it was difficult to decide what type of adaptation was necessary to fulfill our requirements. Among the considerations to be taken into account were the validity and logical completeness of the system, the problems for implementation on the machine, and, especially, the risks of combinatorial explosion, which we were well aware of from our experiments.

Among the question-answering systems and the problem-solving techniques which we had explored, there were those of D. Luckham [29] and N.J. Nilson, of J.L. Darlington [15] and of Cordell Green [19]. Our exploration, the tests by Jean Trudel and Robert Pasero using experimental versions of Philippe's provers, Alain's research on the logical formulation of grammars and numerous discussions with Robert Kowalski: all these elements led us to view the resolution principle from a point of view different from the prevailing one. Rather than demonstrating a theorem by reduction ad absurdum, we wanted to calculate

an ‘interesting’ set of clauses that were deducible from a given set of clauses. Since, to our way of thinking, such sets constituted programs, we would thus have programs generating other programs. This idea constantly underlay the conception of this preliminary version of the language as it did the realization of the application.

The final choice was an adaptation of the resolution method similar to those of the subsequent Prologs but comprising some very novel elements, even compared with modern Prologs. Each execution was performed with a set of clauses constituting the ‘program’ and a set of clauses constituting the ‘questions’. Both of them produced a set of clauses constituting the ‘answers’. The clauses’ literals were ordered from left to right and the resolution was performed between the head literal of the resolvent and the head literal of one of the program clauses. The novelty resided in the fact that in each clause a part of the literals, separated by the ‘/’ sign, was not processed during the proof. Instead, they were accumulated to produce one of the answer clauses at the end of deduction. In addition, certain predicates (such as *DIF*) were processed by delayed evaluation and could also be transmitted as an answer. Finally, it was decided that non-determinism should be processed by backtracking, meaning that only a single branch of the search tree was stored at any given time in the memory.

Formally, the chosen deduction method can be described by the three deduction rules below, where ‘question’, ‘chosen clause’ and ‘answer’ denote three clauses taken respectively from the sets ‘questions’, ‘program’ and ‘answers’ while ‘resolvent’ denotes the current list of goals.

Deduction initialization rule

$$\begin{array}{l} \text{question: } L_1 \dots L_m / R_1 \dots R_n \\ \text{resolvent: } L_1 \dots L_m / R_1 \dots R_n \end{array}$$

Basic deduction rule

$$\begin{array}{l} \text{resolvent: } L_0 L_1 \dots L_m / R_1 \dots R_n, \text{ chosen clause: } L'_0 L'_1 \dots L'_{m'} / R'_1 \dots R'_{n'} \\ \text{resolvent: } \sigma(L'_1) \dots \sigma(L'_{m'}) \sigma(L_1) \dots \sigma(L_m) / \sigma(R'_1) \dots \sigma(R'_{n'}) \sigma(R_1) \dots \sigma(R_n) \end{array}$$

End of deduction rule

$$\begin{array}{l} \text{resolvent: } / R'_1 \dots R'_{n'} \\ \text{answer: } R'_1 \dots R'_{n'} \end{array}$$

where of course,  $L_0$  and  $L'_0$  are complementary unifiable literals and  $\sigma$  is the most general substitution that unifies them. (An example is given later.)

The first reason for choosing this linear resolution technique with a predefined order of literal selection was its simplicity and the fact that we could produce clauses that were logically deducible from the program which thus guaranteed in a way the validity of the results. To a large extent, we were inspired by Robert Kowalski’s SL-Resolution which Philippe had implemented for his thesis on formal equality. However, despite its stack-like functioning analogous to procedure calling in standard languages, we knew that this method introduced computations which were certainly necessary in the general case but unnecessary for most of our examples. We therefore adopted the extremely simplified version of SL-Resolution described by the three rules above, which continues to serve as the basis for all Prologs.

The choice of the treatment of non-determinism was basically a matter of efficiency. By programming a certain number of methods, Philippe had shown

that one of the crucial problems was combinatorial explosion and a consequent lack of memory. Backtracking was selected early on for management of non-determinism, in preference to a management system of several branch calculations simultaneously resident in memory, whose effect would have been to considerably increase the memory size required for execution of the deductions. Alain had a preference for this method, introduced by Robert Floyd [18] to process non-deterministic languages, and was teaching it to all his students. Although, certainly, the use of backtracking led to a loss of completeness in deductions comprising infinite branches, we felt that, given the simplicity of the deduction strategy (the execution of literals from left to right and the choice of clauses in the order they were written), it was up to the programmer to make sure that the execution of his program terminated.

## 4.2 Characteristics of the preliminary Prolog

Apart from the deduction mechanism we have already discussed, a number of built-in predicates were added to the system as Alain and Robert Pasero required them: predicates to trace an execution, *COPY* to copy a term, *BOUM* to split an identifier into a list of characters or reconstitute it and *DIF* to process the symbolic (i.e. syntactic) equality of Philippe's thesis. It should be noted that we refused to include input-output predicates in this list as they were considered to be too far removed from logic. Input-output, specification of the initial resolvents, and chaining between programs were specified in a command language applied to sets of clauses (read, copy, write, merge, prove, etc.). This language, without any control instructions, allowed chainings to be defined only statically but had the virtue of treating communications uniformly by way of sets of clauses. It should be emphasized that this first version already included lazy evaluation (or co-routined evaluation, if you prefer) of certain predicates; in this case *DIF* and *BOUM*. The *DIF* predicate was abandoned in the next version but reappeared in modern Prologs. The only control operators were placed at the end of clauses as punctuation marks, and their function was to perform cuts in the search space. The annotations:

```
.. performed a cut after the head of the clause,
.; performed a cut after execution of the whole rule,
;. performed a cut after production of at least one answer,
;; had no effect.
```

These extra-logical operators were exotic enough to cause their users problems and were therefore subsequently abandoned. Curiously, this punctuation had been introduced by Alain following his discovery of the optional and mandatory transformation rules of the linguist, Noam Chomsky [8].

On the syntactic level, the terms were written in functional form although it was possible to introduce unary or binary operators defined by precedences as well as an infix binary operator that could be represented by an absence of sign (like a product in mathematics and very useful in string input-output). Here is an example of a sequence of programs that produced and wrote a set of clauses defining the great-nephews of a person named *MARIE*:

```
READ
```

```

RULES
+DESC(*X,*Y) -CHILD(*X,*Y);;
+DESC(*X,*Z) -CHILD(*X,*Y) -DESC(*Y,*Z);;
+BROTHERSISTER(*X,*Y) -CHILD(*Z,*X) -CHILD(*Z,*Y) -DIF(*X,*Y);;
AMEN

READ
FACTS
+CHILD(PAUL,MARIE);;
+CHILD(PAUL,PIERRE);;
+CHILD(PAUL,JEAN);;
+CHILD(PIERRE,ALAIN);;
+CHILD(PIERRE,PHILIPPE);;
+CHILD(ALAIN,SOPHIE);;
+CHILD(PHILIPPE,ROBERT);;
AMEN

READ
QUESTION
-BROTHERSISTER(MARIE,*X) -DESC(*X,*Y) / +GREATNEPHEW(*Y) -MASC(*Y)..
AMEN

CONCATENATE(FAMILYTIES,RULES,FACTS)

PROVE(FAMILYTIES,QUESTION,ANSWER)

WRITE(ANSWER)

AMEN

```

The output from this program was thus not a term but instead the following set of binary clauses:

```

+GREATNEPHEW(SOPHIE) -MASC(SOPHIE);.
+GREATNEPHEW(ROBERT) -MASC(ROBERT);.

```

The *READ* command read a set of clauses preceded by a name  $x$  and ending with *AMEN* and assigned it the name  $x$ . The command *CONCATENATE*( $y, x_1, \dots, x_n$ ) computed the union  $y$  of the sets of clauses  $x_1, \dots, x_n$ . The command *PROVE*( $x, y, z$ ) was the most important one; it started a procedure where the program is  $x$ , the initial resolvent is  $y$  and the set of answers is  $z$ . Finally, the command *WRITE*( $x$ ) printed the set of clauses  $x$ .

The man-machine communication system operated in four phases and made use of 4 programs, i.e. 4 sets of clauses:

$C1$  to analyze a text  $T0$  and produce a deep structure  $T1$ ,  
 $C2$  to find in  $T1$  the antecedents of the pronouns and produce a logical form  $T2$ ,  
 $C3$  to split the logical formula  $T2$  into a set  $T3$  of elementary information,  
 $C4$  to carry out deductions from  $T3$  and produce the answers in French  $T4$ .

The sequence of commands was therefore

```

PROVE(C1,T0,T1) PROVE(C2,T1,T2) PROVE(C3,T2,T3) PROVE(C4,T3,T4),

```

where  $T0$ , the French text to process, and  $T4$ , the answers produced, were represented by elementary facts over lists of characters.

### 4.3 Implementation of the preliminary Prolog

Since the Luminy computing center had been moved, the interpreter was implemented by Philippe in Algol-W, on the IBM 360-67 machine of the University of Grenoble computing center, equipped with the CP-CMS operating system based on the virtual machine concept. We were connected to this machine via a special telephone line. The machine had two unique characteristics almost unknown at this time, characteristics that were essential for our work: it could provide a programmer with a virtual memory of 1Mb if necessary (and it was), and it allowed us to write interactive programs. So it was, that, on a single console operating at 300 baud, we developed not only the interpreter but also the question-answering system itself. The choice of Algol-W was imposed on us, since it was the only high level language we had available that enabled us to create structured objects dynamically while also being equipped with garbage collection.

The basis for the implementation of the resolution was an encoding of the clauses into inter-pointing structures with anticipated copying of each rule used in a deduction. Non-determinism was managed by a backtracking stack and substitutions were performed only by creating chains of pointers. This approach eliminated the copying of terms during unifications, and thus greatly improved the computing times and memory space used. The clause analyzer was also written in Algol-W, in which the atoms were managed by a standard "hash-code" technique. This analyzer constituted a not insignificant part of the system which strengthened Alain's desire to solve these syntax problems in Prolog itself. However, experience was still lacking on this topic, since the purpose of the first application was to reveal the very principles of syntactic analysis in logic programming.

## 5 Part IV. The final Prolog

Now that we have described the two forerunners at length, it is time to lay out the fact sheet on the definitive Prolog of 1973. Our major preoccupation after the preliminary version, was the reinforcement of Prolog's programming language aspects by minimizing concepts and improving its interactive capabilities in program management. Prolog was becoming a language based on the resolution principle alone and on the provision of a set of built-in predicates (procedures) making it possible to do everything in the language itself. This set was conceived as a minimum set enabling the user to:

- create and modify programs in memory,
- read source programs, analyze them and load them in memory,
- interpret queries dynamically with a structure analogous to other elements of the language,
- have access dynamically to the structure and the elements of a deduction,



- control program execution as simply as possible.

## 5.1 Resolution strategy

The experience gained from the first version led us to use a simplified version of its resolution strategy. The decision was based not only on suggestions from the first programmers but also on criteria of efficiency and on the choice of Fortran to program the interpreter which forced us to manage the memory space. The essential differences with the previous version were:

- no more delayed evaluation (*DIF*, *BOUM*),
- replacement of the *BOUM* predicate by the more general *UNIV* predicate,
- the operations assert and retract, at that time written *AJOUT* and *SUPP*, are used to replace the mechanism that generates clauses as the result of a deduction,
- a single operator for backtracking management, the search space cut operator ‘!’ , at that time written ‘/’,
- the meta-call concept to use a variable instead of a literal,
- use of the predicates *ANCESTOR* and *STATE*, which have disappeared in present Prologs, to access ancestor literals and the current resolvent (considered as a term), for programmers wishing to define their own resolution mechanism.

Backtracking and ordering the set of clauses defining a predicate were the basic elements retained as a technique for managing non-determinism. The preliminary version of Prolog was quite satisfactory in this aspect. Alain’s reduction of backtracking control management to a single primitive (the cut), replacing the too numerous concepts in the first version produced an extraordinary simplification of the language. Not only could the programmer reduce the search space size according to purely pragmatic requirements but also he could process negation in a way, which, although simplified and reductive in its semantics, was extremely useful in most common types of programming.

In addition, after a visit to Edinburgh, Philippe had in mind the basis for an architecture that is extremely simple to implement from the point of view of memory management and much more efficient in terms of time and space, if we maintained the philosophy of managing non-determinism by backtracking. In the end, all the early experiences of programming had shown that this technique allowed our users to incorporate non-determinism fairly easily as an added dimension to the control of the execution of the predicates.

Concerning the processing of the implicit ‘or’ between literals inside a clause, sequentiality imposed itself there again as the most natural interpretation of this operator since, in formal terms, the order of goal execution has no effect at all on the set of results (modulo the pruning of infinite branches), as Robert Kowalski had proved concerning SL-resolution.

To summarize, these two choices concerning the processing of ‘and’ and ‘or’, were fully justified by the required objectives:

- employ a simple and predictable strategy that the user can control, enabling any extra-logical predicate (such as input-output) to be given an operational definition,
- provide an interpreter capable of processing deductions with thousands or tens of thousands of steps (an impossible objective in the deductive systems existing at that time).

## 5.2 Syntax and Primitives

On the whole, the syntax retained was the same as the syntax of the preliminary version of the language. On the lexical level, the identifier syntax was the same as that of most languages and therefore lower-case letters could not be used (the keyboards and operating systems at that time did not systematically allow this). It should be noted that among the basic primitives for processing morphology problems, one single primitive *UNIV* was used to create dynamically an atom from a character sequence, to construct a structured object from its elements, and, conversely, to perform the inverse splitting operations. This primitive was one of the basic tools used to create programs dynamically and to manipulate objects whose structures are unknown prior to the execution of the program.

Enabling the user to define his own unary and binary operators by specifying numeric precedences proved very useful and flexible although it complicated somewhat the clause analyzers. It still survives as such in the different current Prologs.

In the preliminary version of Prolog, it was possible to create clauses that were logically deducible from other clauses. Our experience with this version had showed us that sometimes it was necessary to manipulate clauses for purposes very far removed from first order logic: modeling of temporal type reasoning, management of information persisting for an uncertain lifetime, or simulation of exotic logics. We felt that much research would still be needed in the area of semantics in order to model the problems of updating sets of clauses. Hence we made the extremely pragmatic decision to introduce extra-logical primitives acting by side effect to modify a program (*ADD*, *DELETE*). This choice seems to have been the right one since these functions have all been retained.

One of the missing features of the preliminary Prolog was a mechanism that could compute a term that could then be taken as a literal to be resolved. This is an essential function needed for meta-programming such as a command interpreter; this feature is very easy to implement from a syntactic point of view. In any event, a variable denoting a term can play the role of a literal.

In the same spirit – and originally intended for specialists in computational logic – various functions giving access to the current deduction by considering it as a Prolog object appeared in the basic primitives (*STATE*, *ANCESTOR*). Similarly, the predicate ‘/’ (pronounced ‘cut’ by Edinburghers) became parametrizable in a very powerful manner by access to ancestors.

## 5.3 A programming example

To show the reader what an early Prolog program looked like, we introduce an old example dealing with flights between cities. From a base of facts which

describes direct flights, the program can calculate routes which satisfy some scheduling constraints.

Direct flights are represented by unary clauses under the following format:

```
+FLIGHT(<departure city>,<arrival city>,<departure time>,<arrival time>,<flight identifier>)
```

where time schedules are represented by pairs of integers under the format  $\langle hours \rangle : \langle minutes \rangle$ . All flights are supposed to be completed in the same day.

The following predicate will be called by the user to plan a route.

```
PLAN(<departure city>,<arrival city>,<departure time>,<arrival time>,<departure min time>,<arrival max time>)
```

It enumerates (and outputs as results) all pairs of cities connected by a route which can be a direct flight or a sequence of flights. Except for circuits, the same city cannot be visited more than once. Parameters  $\langle departure\ min\ time \rangle$  and  $\langle arrival\ max\ time \rangle$  denote constraints given by the user about departure and arrival times. The first flight of the route should leave after  $\langle departure\ min\ time \rangle$  and the last one should arrive before  $\langle arrival\ max\ time \rangle$ .

In order to calculate *PLAN*, several predicates are defined. The predicate:

```
ROUTE(<departure city>,<arrival city>,<departure time>,<arrival time>,<plan>,<visits>,<departure mini time>,<arrival maxi time>)
```

is similar to the *PLAN* predicate, except for two additional parameters: the input parameter  $\langle visits \rangle$  is given represents the list  $c_k.c_{k-1} \dots c_1.NIL$  of already visited cities (in inverse order), the output parameter  $\langle plan \rangle$  is the list  $f_1 \dots f_k.NIL$  of calculated flight names. The predicates *BEFORE*( $t_1, t_2$ ) and *ADDTIMES*( $t_1, t_2, t_3$ ) deal with arithmetic aspects of time schedules. The predicate *WRITEPLAN* writes the sequence of flight names. Finally, *NOT*( $p$ ) defines negation by failure, and *ELEMENT*( $e, l$ ) succeeds if  $e$  is among the list  $l$ .

Here then is the complete program, including data and the saving and execution commands.

```
* INFIXED OPERATORS.
```

```
-AJOP( ".", 1, "(X|X)|X" ) -AJOP(":", 2, "(X|X)|X" ) !
```

```
* USER PREDICATE.
```

```
+PLAN(*DEPC, *ARRC, *DEPT, *ARRT, *DEPMINT, *ARRMAXT)
  -ROUTE(*DEPC, *ARRC, *DEPT, *ARRT, *PLAN, *DEPC.NIL, *DEPMINT, *ARRMAXT)
  -SORM("-----")
  -LIGNE
  -SORM("FLYING ROUTE BETWEEN:  ")
  -SORT(*DEPC)
  -SORM(" AND ")
  -SORT(*ARRC)
  -LIGNE
```

```

-SORM("-----")
-LIGNE
-SORM("DEPARTURE TIME:  ")
-SORT(*DEPT)\
-LIGNE
-SORM("ARRIVAL TIME:  ")
-SORT(*ARRT)
-LIGNE
-SORM("FLIGHTS:  ")
-WRITEPLAN(*PLAN) -LIGNE -LIGNE.

* PRIVATE PREDICATES.

+ROUTE(*DEPC, *ARRC, *DEPT, *ARRT, *FLIGHTID.NIL, *VISITS, *DEPMINT, *ARRMAXT)
  -FLIGHT(*DEPC, *ARRC, *DEPT, *ARRT, *FLIGHTID)
  -BEFORE(*DEPMINT, *DEPT)
  -BEFORE(*ARRT, *ARRMAXT).

+ROUTE(*DEPC,*ARRC, *DEPT,*ARRT, *FLIGHTID.*PLAN, *VISITS,*DEPMINT, *ARRMAXT)
  -FLIGHT(*DEPC, *INTC, *DEPT, *INTT, *FLIGHTID)
  -BEFORE(*DEPMINT, *DEPT)
  -ADDTIMES(*INTT, 00:15, *INTMINDEPT)
  -BEFORE(*INTMINDEPT, *ARRMAXT)
  -NOT(ELEMENT(*INTC, *VISITS))
  -ROUTE(*INTC,*ARR,*INTDEPT,*HARR, *PLAN, *INTC.*VISITS, *INTMINDEPT, *ARRMAXT).

+BEFORE(*H1:*M1, *H2:*M2) -INF(H1, H2).
+BEFORE(*H1:*M1, *H1:*M2) -INF(M1, M2).

+ADDTIMES(*H1:*M1, *H2:*M2, *H3:*M3)
  -PLUS(*M1, *M2, *M)
  -RESTE(*M, 60, *M3)
  -DIV(*M, 60,*H)
  -PLUS(*H, *H1, *HH)
  -PLUS(*HH,*H2,*H3).

+WRITEPLAN(*X. NIL) -/ -SORT(*X).
+WRITEPLAN(*X.*Y) -SORT(*X) -ECRIT(-) -WRITEPLAN(*Y).

+ELEMENT(*X, *X.*Y).
+ELEMENT(*X, *Y.*Z) -ELEMENT(*X, *Z).

+NOT(*X) -*X -/ -FAIL.
+NOT(*X).

* LIST OF FLIGHTS.

```

```
+FLIGHT(PARIS, LONDON, 06:50, 07:30, AF201).
+FLIGHT(PARIS, LONDON, 07:35, 08:20, AF210).
+FLIGHT(PARIS, LONDON, 09:10, 09:55, BA304).
+FLIGHT(PARIS, LONDON, 11:40, 12:20, AF410).
+FLIGHT(MARSEILLES, PARIS, 06:15, 07:00, IT100).
+FLIGHT(MARSEILLES, PARIS, 06:45, 07:30, IT110).
+FLIGHT(MARSEILLES, PARIS, 08:10, 08:55, IT308).
+FLIGHT(MARSEILLES, PARIS, 10:00, 10:45, IT500).
+FLIGHT(MARSEILLES, LONDON, 08:15, 09:45, BA560).
+FLIGHT(MARSEILLES, LYON, 07:45, 08:15, IT115).
+FLIGHT(LYON, LONDON, 08:30, 09:25, TAT263).
```

```
* SAVING THE PROGRAM.
```

```
-SAUVE!
```

```
* QUERYING.
```

```
-PLAN(MARSEILLES, LONDON, *HD, *HA, 00:00, 09:30)!
```

```
This is the ouput of the program:
```

```
-----
FLYING ROUTE BETWEEN:  MARSEILLES AND LONDON
-----
```

```
DEPARTURE TIME:  06:15
ARRIVAL TIME:    08:20
FLIGHTS:         IT100-AF210
```

```
-----
FLYING ROUTE BETWEEN:  MARSEILLES AND LONDON
-----
```

```
DEPARTURE TIME:  07:45
ARRIVAL TIME:    09:25
FLIGHTS:         IT115-TAT263
```

## 5.4 Implementation of the interpreter

The resolution system, in which non-determinism was managed by backtracking, was implemented using a very novel method for representing clauses, halfway between the technique based on structure sharing used by Robert Boyer and Jay Moore in their work on the proofs of programs [6, 31] and the backtracking technique used in the preliminary version of Prolog. Philippe came up with this solution during his stay in Edinburgh after many discussions with Robert Boyer. In this new approach, the clauses of a program were encoded in memory as a series of templates, which can be instantiated without copying, several times

in the same deduction, by means of contexts containing the substitutions to be performed on the variables. This technique had many advantages compared to those normally used in automated theorem-proving:

- in all known systems, unification was performed in times which were, at best, linear in relation to the size of the terms unified. In our system, most of the unifications could be performed in constant time, determined not by the size of the data, but by that of the templates brought into action by the clauses of the called program. As a result, the concatenation of two lists was performed in a linear time corresponding to the size of the first and not in quadratic time as in all other systems based on copying techniques;
- in the same system, the memory space required for one step in a deduction is not a function of the data, but of the program clause used. Globally therefore, the concatenation of two lists used only a quantity of memory space proportional to the size of the first list;
- the implementation of non-determinism did not require a sophisticated garbage collector in the first approach but simply the use of several stacks synchronized on the backtracking, thus facilitating rapid management and yet remaining economical in the use of interpreter memory.

Concerning the representation of the templates in memory, we decided to use prefix representation (the exact opposite of Polish notation). The system consisted of the actual interpreter (i.e., the inference machine equipped with a library of built-in predicates), a loader to read clauses in a restricted syntax, and a supervisor written in Prolog. Among other things, this supervisor contained a query evaluator, an analyzer accepting extended syntax and the high level input-output predicates.

Alain, who like all of us disliked Fortran, succeeded nonetheless in persuading the team to program the interpreter in this language. This basic choice was based primarily on the fact that Fortran was widely distributed on all machines and that the machine we had access to at that time supported no other languages adapted to our task. We hoped in that way to have a portable system, a prediction which proved to be quite correct.

Under Philippe's supervision, Gérard Battani and Henri Meloni [1] developed the actual interpreter between June 1973 and October 1973 on a CII 10070 (variant of the SIGMA 7) while René Bazzoli, under Alain's direction, was given the task of writing the supervisor in the Prolog language itself. The program consisted of approximately 2000 instructions, of roughly the same size as the Algol-W program of the initial version.

The machine had a batch operating system with no possibility of interaction via a terminal. Hence, data and programs were entered by means of punched cards. That these young researchers could develop as complex a system as this under such conditions and in such short time is especially remarkable in light of the fact that none of them had ever written a line of Fortran before in their lives. The interpreter was finally completed in December 1973 by Gérard Battani and Henry Meloni after porting it onto the IBM 360-67 machine at Grenoble, thus providing somewhat more reasonable operating conditions. Philippe Roussel wrote the reference and user's manual for this new Prolog two years later [35].

## 6 Conclusion

After all these vicissitudes and all the technical details, it might be interesting to take a step back and to place the birth of Prolog in a wider perspective. The article published by Alan Robinson in January 1965, ‘A machine-oriented logic based on the resolution principle’, contained the seeds of the Prolog language. This article was the source of an important stream of works on automated theorem-proving and there is no question that Prolog is essentially a theorem prover ‘à la Robinson’.

Our contribution was to transform that theorem prover into a programming language. To that end, we did not hesitate to introduce purely computational mechanisms and restrictions that were heresies for the existing theoretical model. These modifications, so often criticized, assured the viability and thus the success of Prolog. Robert Kowalski’s contribution was to single out the concept of the ‘Horn clause’, which legitimized our principal heresy: a strategy of linear demonstration with backtracking and with unifications only at the heads of clauses.

Prolog is so simple that one has the sense that sooner or later someone had to discover it. Why did we discover it rather than anyone else? First of all, Alain had the right background for creating a programming language. He belonged to the first generation of Ph.D.s in computer science in France and his specialty was language theory. He had gained valuable experience in creating his first programming language, Q-systems, while on the staff of the machine translation project at the University of Montreal. Then our meeting, Philippe’s creativity and the particular working conditions at Marseilles did the rest. We benefitted from freedom of action in a newly created scientific center and, having no outside pressures, we were able to fully devote ourselves to our project.

Undoubtedly, this is why that period of our lives remains one of the happiest in our memories. We have had the pleasure of recalling it for this paper over fresh almonds accompanied by a dry martini.

## Acknowledgement

We would like to thank all those who contributed to the English version of this paper: Andy Tom, Franz Gueünther, Mike Mahoney and Pamela Morton.

## References

- [1] Battani Gérard et Henry Meloni, *Interpréteur du langage PROLOG*, DEA report, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, 1973.
- [2] Battani Gérard, *Mise en oeuvre des contraintes phonologiques, syntaxiques et sémantiques dans un système de compréhension automatique de la parole*, Thesis, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, June 1975.

- [3] Bergman Marc and Henry Kanoui, *Application of mechanical theorem proving to symbolic calculus*, Third International Colloquium on advanced Computing Methods in Theoretical Physics, Marseilles, France, June 1973.
- [4] Bergman Marc, *Résolution par la démonstration automatique de quelques problèmes en intégration symbolique sur calculateur*, Thesis, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, Octobre 1973.
- [5] Bergman Marc and Henry Kanoui, *SYCOPHANTE, système de calcul formel sur ordinateur*, final report for a DRET contract (Direction des Recherches et Etudes Techniques), Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, 1975.
- [6] Boyer Roger S. and Jay S. Moore, *The sharing of Structure in Theorem Proving Programs*, Machine Intelligence 7, edited by B. Melzer et D. Michie, Edinburgh University Press, New York, pp. 101-116, 1972.
- [7] Chastellier (de) Guy and Alain Colmerauer, *W-Grammar. Proceedings of the ACM Congress*, San Francisco, August, ACM, New York, pp. 511- 518, 1969.
- [8] Chomsky Noam, *Aspects of the Theory of Syntax*, MIT Press, Cambridge, 1965.
- [9] Cohen Jacques, *A view of the origins and development of Prolog*, Commun. ACM 31, 1, pp. 26-36, January 1988.
- [10] Colmerauer Alain, *Total precedence relations*, J. ACM 17, 1, pp. 14-30, January 1970.
- [11] Colmerauer Alain, *Les systèmes-q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur*, Internal publication 43, Département d'informatique de l'Université de Montréal, September 1970.
- [12] Colmerauer Alain, Fernand Didier, Robert Pasero, Philippe Roussel, Jean Trudel, *Répondre à*, Internal publication, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, May 1971. This publication is a computer print-out with handwritten remarks.
- [13] Colmerauer Alain, Henry Kanoui, Robert Pasero et Philippe Roussel, *Un système de communication en français*, rapport préliminaire de fin de contrat IRIA, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, October 1972.
- [14] Colmerauer Alain, *Les grammaires de métamorphose GIA*, Internal publication, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, novembre 1975. English version, *Metamorphosis grammars, Natural Language Communication with Computers*, Lectures Notes in Computer Science 63, édité par L. Bolc, Springer Verlag, Berlin Heidelberg, New York, pp. 133-189, 1978, ISBN 3-540-08911-X.
- [15] Darlington J. L., *Theorem-proving and information retrieval*, Machine Intelligence 4, Edinburgh University Press, pp. 173-707, 1969.
- [16] Elcok E. W. *Absys: the first logic programming language – A retrospective and a commentary*. Journal of Logic Programming, 9(1):1-17, July 1990.



- [17] Floyd Robert W., *Syntactic analysis and operator precedence*. J.ACM 10, pp. 316-333, 1963.
- [18] Floyd Robert W., *Nondeterministic algorithms*. J. ACM 14, 4, pp. 636-644, October 1967.
- [19] Green Cordell C., *Application of theorem-proving to problem-solving*, Proceedings of First International Joint Conference on Artificial Intelligence, Washington D.C., pp. 219-239, 1969.
- [20] Hewitt Carl, *PLANNER: A language for proving theorems in robots*, Proceedings of First International Joint Conference on Artificial Intelligence, Washington D.C., pp. 295-301, 1969.
- [21] Joubert Michel, *Un système de résolution de problèmes à tendance naturelle*, Thesis, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, February 1974.
- [22] Kanoui Henry, *Application de la démonstration automatique aux manipulations algébriques et à l'intégration formelle sur ordinateur*, Thesis, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, Octobre 1973.
- [23] Kanoui Henry, *Some aspects of Symbolic Integration via Predicate Logic Programming*. ACM SIGSAM Bulletin, 1976.
- [24] Kowalski Robert A. et D. Kuehner, *Linear resolution with selection function*, memo 78, University of Edinburgh, School of Artificial Intelligence, 1971. Also in Artificial Intelligence Vol. 2, pp. 227-60, 1971.
- [25] Kowalski Robert A., *Predicate Logic as Programming Language*, memo 70, University of Edinburgh, School of Artificial Intelligence, November 1973. Also in Proceedings of IFIP 1974, North Holland Publishing Company, Amsterdam, pp. 569-574, 1974.
- [26] Kowalski Robert A. and Maarten van Emden, *The semantic of predicate logic as programming language*, memo 78, University of Edinburgh, School of Artificial Intelligence, 1974. Also in JACM 22, 1976, pp. 733-742.
- [27] Kowalski Robert A., *The early history of Logic Programming*, CACM vol. 31, no. 1, pp 38-43, 1988.
- [28] Loveland, D. W., *Automated theorem proving: A quarter-century review*, Am. Math. Soc. 29 , pp. 1-42, 1984.
- [29] Luckam D. and N. J. Nilson, *Extracting information from resolution proof trees*, Artificial Intelligence 12, 1, pp. 27-54, 1971
- [30] Meloni Henry, *Mise en oeuvre des contraintes phonologiques, syntaxiques et sémantiques dans un système de compréhension automatique de la parole*, Thesis, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, June 1975.
- [31] Moore J. Strother, *Computational Logic' Structure sharing and proof of program properties, part I and II*, memo 67, University of Edinburgh, School of Artificial Intelligence, 1974.

- [32] Pasero Robert, *Représentation du français en logique du premier ordre en vue de dialoguer avec un ordinateur*, Thesis, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, May 1973.
- [33] Pereira Fernando C. and David H. D. Warren, *Definite clause grammars for language analysis*, Artificial Intelligencec. 13, pp. 231-278, 1980.
- [34] Roussel Philippe, *Définition et traitement de l'égalité formelle en démonstration automatique*, Thesis, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, May 1972.
- [35] Roussel Philippe, *Prolog, manuel de référence et d'utilisation*, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, September 1975.
- [36] Robinson J. A., *A machine-oriented logic based on the resolution principle*, J. ACM 12, 1, pp. 23-41, January 1965.
- [37] *TAUM 71, Annual report, Projet de Traduction Automatique de l'Université de Montréal*, January 1971.
- [38] Warren David H. D., *Warplan, A System for Generating Plans*, research report, University of Edimburgh, Department of Computational Logic, memo 76, June 1974.
- [39] Warren David H. D., Luis M. Pereira and Fernando Pereira, *Prolog the language and its implementation*, Proceedings of the ACM, Symposium on Artificial Intelligence and Programming Languages, Rochester, N.Y., August 1977.
- [40] Wijngaarden(van) A., B. J. Mailloux, J. E.L .Peck and G.H.A Koster, *Final Draft Report on the Algorithmic Language Algol 68*, Mathematish Centrum, Amsterdam, December 1968.