Even though the creator of Prolog, Alain Colmerauer, had been an office mate of mine at the University of Grenoble and both of us were interested in similar topics — mainly parsing, compiling and non-determinism — it took me over a decade to get involved in the emerging area of Logic Programming.

My initial interest in automatic theorem proving — the foundation of Prolog — stemmed from two sources: a conference I attended in Menaggio, Italy in the late sixties, and my association with a colleague, Jean van Heijenoort, who taught formal logic at Brandeis.

One of the lecturers at Menaggio was Alan Robinson, who had proposed the ingenious and widely employed approach of combining resolution with unification to attempt to automatically prove theorems. I remember having programmed both algorithms but I had failed to see its relationship with non-determinism, parsing and programming languages. It took of course the major contributions of Colmerauer and Kowalski to reach the level of development of a programming language based on unification and Horn clauses; that evolution made resolution considerably more manageable by reducing the combinatorial explosion that occurred in the general case of the predicate calculus.

I was also privileged to have a long and fruitful association with Jean van Heijenoort [Fef 93]. I recall attending his formal logic lectures at Brandeis in which he used the tree method for proving theorems in the predicate calculus [Jef 67]. With one of my undergraduate students, Ann D. Rubin, I developed an interactive program to prove theorems using the tree method [CR 71]. A few years later, I described with Laurent Trilling and Peter Wegner a variant of those programs using Algol 68, a language that made the presentation much more succinct and rigorous [CTW 74]. I also recall helping Jean van Heijenoort implement a tree method interactive system for intuitionistic logic.

In 1980, I published a paper in the *ACM Computer Surveys* in which I described the importance of non-determinism as specified by annotations in programming languages [Coh 79b]. Both Alain Colmerauer and I had been intrigued by Robert Floyd’s work that showed how to solve combinatorial problems (e.g., eight queens) by introducing a few backtracking primitives in a flowchart language. With a student, Eileen Carton, I introduced those primitives in Fortran and other high-level languages [CC 74]. Non-determinism plays a key role in logic programming.

My first encounter with Prolog occurred in the late 1970s. I was invited to offer a course in compiling at the University of Marseille, where Alain Colmerauer was the department chair of Computer Science. Actually, I taught that course for a few years and I recall Alain mentioning that most of the material in parsing in compiler could be done in Prolog. It was only at that time that I read his paper on Metamorphosis Grammars in which he described in detail a Prolog implementation of a compiler [Col 78]. Several years later, David H. Warren illustrated the importance of Colmerauer’s work...
My first paper concerning Prolog involved the use of infinite trees. With Francis Giannesini, a colleague of Alain at Marseilles, we described a parser generator using Prolog II, whose compiler could handle infinite trees [GC 84]. It was after this paper that I became convinced of the multiple advantages of logic programming and strived to learn more about it.

At that time, the Japanese invested heavily in their fifth generation effort, and Americans started to pay increased attention to the logic programming paradigm. I was asked by Jim Maurer, the executive editor of Communications of the ACM, to write a paper on Prolog. That paper, “Describing Prolog by its Interpretation and Compilation” [Coh 85], was probably among the first papers appearing in an American journal to demonstrate a concern for that topic.

In the early 1980s, I co-organized (with Jan Komorowski) the first tutorial in Prolog in the Boston area. Since that time, I have authored and co-authored many papers involving LP. Two of them had to do with the historical origins of Prolog [Coh 88], [Coh 96b]. I also challenged myself to summarize most the algorithms presented in the “Dragon Book” [ASU 86], a classic text in compilers, using Prolog. A paper on parsing and compiling using Prolog co-authored with Timothy Hickey appeared in the ACM Transactions on Programming Languages and Systems [CH 87].

In the late 1980s, Alain Colmerauer and Jean-Louis Lassez (then at IBM) promoted a beautiful extension of the LP that they called Constraint Logic Programming (CLP). Alain, who had already conducted work in infinite trees and disequalities — which can be viewed constraints — embarked on the ambitious project of designing Prolog III: a language incorporating a variety of new types of constraints including Booleans and linear inequations (akin to those used in Danzig’s simplex method). The IBM group of Joxan Jaffar and Jean-Louis Lassez designed and implemented a language called CLP(R). In Prolog, backtracking occurs when unification fails. In CLP, backtracking occurs when a set of constraints becomes unsatisfiable. In the constraint context, unification is viewed as a set of constraints in the domain of trees.

Jaffar and Lassez proved remarkable properties of a meta-theory. If a constraint domain (e.g. reals, rationals) and the chosen predicates (e.g., equalities, inequalities, disequalities) satisfy certain conditions, then the foundations of LP regarding soundness and completeness are extended to a language manipulating those domains and constraints. Two papers describe my involvement in describing CLP in the context of LP: [Coh 90], [Coh 97b]. In addition, in the early 1990s, I co-organized, with Jean Louis Lassez, the first American workshop in CLP.

A new development took place in CLP in the early 1990s: the proposal of interval constraints, to represent the domain of reals within floating point notation. This work enabled CLP to consider non-linear numeric problems. My doctoral student, Suresh Kalathur, wrote his dissertation [Kal 99] on the parallel processing of interval constraints using SIMD and MIMD parallel architectures.

A publication, co-authored with Laurent Trilling and his doctoral student Dennis Bouhineau, describes the use of CLP in proving theorems in two-dimensional geometry [CBT 99]. This paper was part of a special issue of the journal Constraints of which I was the guest editor.

My views on future exciting research problems in CLP are: (1) soft or probabilistic constraints (see for example [VdB 13]), and (2) inductive CLP [ZL 08]. The first one attempts to “soften the rigidity” in constraint satisfaction and constraints are either satisfiable or not; it would be worthwhile to attempt to relax that rigidity and consider constraints that are “almost” satisfiable. Regarding the second topic, inductive CLP would allow the generation of CLP programs from positive and negative data.

Jacques Cohen, has been named one of the pioneers of Logic Programming. (Wikipedia)

Alain Colmerauer the architect of Logic Programming died in May of 2017. Alain’s web page summarizing his prodigious work is a must to anyone who wishes to have a deeper knowledge of his unusually creative mind.

References


Logic programming (LP) is a language paradigm based on logic. Its constructs are Boolean implications (e.g., q implies p meaning that p is true if q is true), compositions using the Boolean operators and (called conjunctions) and or (called disjunctions). LP can also be viewed as a procedural language in which the procedures are actually Boolean functions, the result of a program always being either true or false. In the case of implications, a major restriction applies: when q implies p, written p:- q, then q can consist of conjunctions but p has to be a singleton, representing the (sole) Boo Higher-order constructs extend the expressiveness of first-order (Constraint) Logic Programming ((C)LP) both syntactically and semantically. At the same time assertions have been in use for some time in (C)LP systems more. This paper contributes to filling this gap by extending the assertion-based approach to error detection and program validation to the higher-order context within (C)LP. We propose an extension of properties and assertions as used in (C)LP in order to be able to fully describe arguments that are predicates. The extension makes the full power of the assertion language available when describing higher-order arguments. “Computational complexity and constraint logic programming languages” In: Proceedings of the North-American Conference on Logic Programming (NACLP-90), Austin, TX, October. Dantzig, GB, Orden, A and Wolfe, P, 1955. “The generalized simplex method for minimizing a linear form under linear inequality constraints” Pacific J. Math. “Solving large scheduling problems in logic programming” In: EURO-TIMS Joint International Conference on Operations Research and Management Science, Paris, France, July. Dincbas, M, Simonis, H and Van Hentenryck, P, 1988c. “Solving the car sequencing problem in constraint logic programming” In: European Conference on Artificial Intelligence (ECAI-88), Munich, Germany, August. Constraint logic programming lies at the intersection of logic programming, optimisation and artificial intelligence. It has proved a successful tool for application in a variety of areas including production planning, transportation scheduling, numerical analysis and bioinformatics. Its migration from academic discipline to software development has been due in part to the availability of software systems that realise the underlying methodology; ECLi PSe is one of the leading such systems.