The Birth of Prolog

Aaron Bembenek

October 2016

1 A Machine-Oriented Logic Based on the Resolution Principle

```
@article{Robinson:1965,
 author = {Robinson, J. A.},
 title = {A Machine-Oriented Logic Based on the Resolution Principle},
 journal = {J. ACM},
 volume = {12},
 number = {1},
 month = jan,
 year = {1965},
 pages = {23--41},
 publisher = {ACM},
 address = {New York, NY, USA},
}
```

Summary: Robinson presents the resolution principle, an inference principle that can be used to determine the unsatisfiability of a sentence in first-order logic that is especially well-suited for automation. He observes that existing automated theorem proving techniques run into avoidable combinatorial problems because they depend on simple inference principles that were originally designed for human comprehension and are not a good fit for automation. He proposes instead the resolution principle, a generalization of the cut rule that allows inferences that are not necessarily intuitive, but does not suffer the combinatorial issues encountered by previous methods. While a resolution rule for propositional logic had been presented earlier in Davis and Putnam [1960], Robinson's contribution is generalizing resolution to the first-order setting, which he accomplishes by giving an algorithm for finding the most general unifying substitution of a set of literals. Robinson concludes his paper with some "search principles," improvements to the resolution principle that in some cases quicken the convergence of resolution or even help resolution converge when it would not otherwise. An alternative title for this paper could be "Resolution: A Logical Inference Principle for Efficient, Automated Theorem Proving."

Evaluation: Lloyd [1984] describes Robinson's paper as a "landmark paper." It led to intense research activity on refinements and variations of resolution in the late 1960s, as well as a line of research pushing back against

resolution in favor of more human-oriented, heuristic-based theorem provers (Loveland [1984]). Beyond its importance for automated theorem proving, the resolution principle largely enabled the development of logic programming and is the theoretical foundation of the Prolog line of languages. According to Google Scholar, this paper has been cited over 5000 times.

2 The Birth of Prolog

```
@incollection{Colmerauer:1996,
 author = {Colmerauer, Alain and Roussel, Philippe},
 chapter = {The Birth of Prolog},
 title = {History of Programming languages---II},
 editor = {Bergin,Jr., Thomas J. and Gibson,Jr., Richard G.},
 year = {1996},
 pages = {331--367},
 publisher = {ACM},
 address = {New York, NY, USA},
}
```

Summary: This paper reports on the original motivations behind the development of the logic programming language Prolog and the particular context in which it was created. It makes clear that Prolog was developed at first not as a general-purpose programming language, but for a specific application (namely, making deductions from French-language texts), and reveals that some of the design choices behind the features most characteristic of Prolog were made without a full understanding of their theoretical ramifications. The authors trace the development of Prolog from the Q-systems (a language for concisely expressing text-rewriting rules), through the creation of the preliminary version of Prolog in 1972, and conclude with the "final" Prolog of 1973. An alternative title for this paper could be "Prolog = NLP + Automated Theorem Proving."

Evaluation: Since this is not a research paper, we do not evaluate its influence *per se*. However, we can certainly say that Prolog has been (and continues to be, despite its age) the most influential and most popular language to come from the logic programming paradigm. It motivated research formalizing the programming language semantics of the Horn clause formulation of first-order logic (see, for instance, Van Emden and Kowalski [1976]), and also inspired a line of research exploring the use of more expressive logics as programming languages (e.g., non-monotonic logics, as in Ling [1990]). It has served as the basis both for more restricted languages like Datalog (a syntactic subset of Prolog) and for languages that extend Prolog with features from outside the logic programming paradigm, such as Picat. However, despite fluctuations in academic interest, logic programming remains overall a niche interest.

3 Algorithm = Logic + Control

```
@article{Kowalski:1979,
 author = {Kowalski, Robert},
 title = {Algorithm = Logic + Control},
 journal = {Commun. ACM},
 volume = {22},
 number = {7},
 month = jul,
 year = {1979},
 pages = {424--436},
 publisher = {ACM},
 address = {New York, NY, USA},
}
```

Summary: Kowalski argues that programs can naturally be split into two distinct components: logic, which is a declarative expression of what the program does, and control, which is how the program does it. He claims that programs would be easier to write correctly and to reason about if programming languages made a clear distinction between the two. He demonstrates his point through examples using a logic programming language based (similarly to Prolog) on the Horn clause formulation of first-order logic, and shows how, for instance, the same set of logic clauses can be successfully evaluated using either a bottom-up or top-down technique. His argument can be seen as a powerful justification for the use of logic programming. He also points out that at the time of writing the database community had already embraced a similar division of logic and control. An alternative title for this paper could be "Another Layer of Abstraction: Separating What Programs Do From How They Do It."

Evaluation: According to Lloyd [1984], Kowalski's argument is "[o]ne of the main ideas of logic programming." While this claim is certainly true, logic programming languages have in practice adopted this idea to varying degrees; for instance, Datalog makes a very clear separation between logic and control, but Prolog fails to divide the two as cleanly. Kowalski's argument plays into the tension between abstraction and performance, a common theme in programming languages.

References

Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, July 1960.

T. W. Ling. The prolog not-predicate and negation as failure rule. *New Gen. Comput.*, 8(1):5–31, July 1990.

J.W. Lloyd. *Foundations of Logic Programming.* Symbolic Computation. Springer-Verlag, Berlin, 1984.

Donald W. Loveland. Automated theorem-proving: A quarter-century review. *Contemporary Mathematics*, 29:1–45, 1984.

M. H. Van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, October 1976.