Machines as Thought Partners: Reflections on 50 Years of Prolog

Gregory Gelfond[1], Marcello Balduccini[1,2], David Ferrucci[1], Adi Kalyanpur[1], and Adam Lally[1]

[1] Elemental Cognition Inc.
[2] Saint Joseph's University

Abstract. In 1972, Kowalski and Colmerauer started a revolution with the advent of the Prolog programming language. As with LISP, the language enabled us to *think previously impossible thoughts*, and ushered in both logic programming and the declarative programming paradigm. Since that time, a number of descendants of Prolog have been brought into the world, among them constraint logic programming and answer-set prolog. In this paper, we celebrate the 50th anniversary of the Prolog language, and give a brief introduction to a new member of the Prolog family of languages — the logic programming language *Cogent*.

Keywords: Logic programming · Knowledge Representation · Programming Languages · Prolog Anniversary · Cogent.

1 Introduction

In his 1972 Turing Award Lecture, Edsger Dijkstra notes that LISP "has assisted a number of our most gifted fellow humans in thinking previously impossible thoughts." Curiously, it was during that same year that Prolog was developed. We do not know if it was felt at that time just how important the discovery of the Prolog language was, but it is not surprising that the name of the language, an acronym for *"Programming in Logic"*, is a homophone for *prologue*. Robert Kowalski's and Alain Colmerauer's language was an introduction to a new way of thinking about programming, one which in some ways is alluded to by an old joke at the language's expense:

> Prolog is what you get when you create a language and system that has the intelligence of a six-year-old - it simply says "no" to everything.

The joke hints at just how revolutionary the language was. For the first time, we now had a language that rather than having a programmer answer the question of *"how"*, we had one that enabled us to answer the question of *"what"*. In other words, the language freed us from thinking about and describing the mechanics of an algorithm, and allowed us to focus on describing the goal, or specification that the algorithm was intended to meet. So, if we come back to the notion of a six-year-old child, it turned a programmer into a teacher, and the computer into

a student. This shift, to return to Dijkstra's quote on LISP, enabled us to think previously impossible thoughts – and therefore, to ask previously impossible questions.

Two other aspects of the language's nature – its connection to both *Horn clauses* and *context-free grammars* shed light on the kinds of heretofore impossible thoughts we now find ourselves engaged with. SLD and its successor SLDNF resolution enabled us to both simply encode and render computable part of the language of thought itself. This in turn shifted our gaze to the question of: "What kinds of reasoning can be described (i.e., taught) to a machine?" The search for answers to these questions (and others such as uncovering the nature of negation-as-failure) gave rise to other languages and their attendant semantics, such as the *well-founded* [9] and *answer-set semantics* [1,2], advancing our understanding of how we ourselves reason and how the kind of reasoning we carry out can be imparted to a machine. These questions yielded further lines of inquiry into areas such as commonsense reasoning, natural language understanding, reasoning about actions and change, and algorithmics, many of which are part of the foundation of the artificial intelligence technologies in active development here at Elemental Cognition[3].

Elemental Cognition (EC), a company founded by Dave Ferrucci after his success in helming IBM's Watson Project[4] through its landmark success in beating the best humans at the question-answering game of Jeopardy, is a particular beneficiary of the foundations laid by Kowalski and those who followed him. The fields of *knowledge representation*, *non-monotonic reasoning*, and *declarative programming* can trace part of their ancestry to Kowalski's work, and provide the logical foundations of the work done at EC. In particular, our vision of artificial agents as "thought partners" capable of collaborating with humans, rather than just acting autonomously, depends on numerous developments in these fields.

EC's history with logic programming begins in some respects with Ferrucci's own background, and the IBM Watson project in particular. There, Prolog played a role in the project's natural language pipeline and was instrumental in the detection and extraction of semantic relations in both questions and natural language corpora. Prolog's simplicity and expressiveness enabled the developers to readily deal with rule sets consisting of more than 6,000 Prolog clauses, something which prior efforts involving custom pattern-matching frameworks failed to do. This work in no small part informed the design of EC's neuro-symbolic reasoner, *Braid* [3]. The expressivity and transparency of a Prolog-like language combined with the statistical pattern matching power of various machine learning models enabled a powerful *HybridAI* solution which had been applied to several "real-world" applications. This work in part involved the development of a backward chaining system that can be seen as an extension of Prolog's SLD resolution algorithm by features such as *statistical/fuzzy unification* and *probabilistic rules* generated by a machine learning model. This enabled the system to circumvent the knowledge acquisition bottleneck and potential brittleness of matching/unification,

[3] `https://ec.ai`
[4] `https://www.ibm.com/watson`

while retaining the elegance and simplicity of the declarative paradigm itself. Subsequent work has seen the Braid reasoning system evolve towards the use of the *answer-set semantics* and *constraint logic programming* [7].

All of this enabled a number of high-profile successes, such as our development of the PolicyPath[5] application which was used during Super Bowl LV in 2021 at the height of the Covid-19 pandemic [4]. The project was built on a declarative, logic-based representation of the related policies, and part of the reasoning mechanisms developed in the course of the project combined techniques for *reasoning about actions and change* with various flavors of logic programming including *answer-set programming* and *constraint logic programming*. Other successes include our partnership with the OneWorld Alliance[6] on the development of the virtual agent they employ for scheduling round-the-world travel.

In this paper we give an introduction to a new language called Cogent[7] under development at EC, which carries forward the torch that was lit by the introduction of Prolog.

2 From Prolog to Cogent

As was mentioned previously, the advent of logic programming enabled us to shift our focus from describing the *how* of a computation, to the *what*. In other words, it enabled us to focus our attention on what Niklaus Wirth termed *"the refinement of specification"*. As an example, let's consider the following example: a nurse scheduling program written in *answer-set prolog* (a descendant of Prolog based on the answer-set semantics of logic programs, and one of the elements at the core of EC's internal language known as Cordial).

Listing 1.1. Nurse Scheduling in Answer-Set Prolog

```
% The nurses are Andy, Betty, and Chris.
nurse(andy; betty; chris).

% The days are Monday, Tuesday, and Wednesday.
day(monday; tuesday; wednesday).

% The shifts are first, second, and third.
shift(1;2;3).

% We may choose for a nurse to be assigned to a shift on a day.
{ assigned(N,S,D) }:- nurse(N), shift(S), day(D).

% A nurse cannot be assigned to more than one shift on the same day.
:- nurse(N), day(D), #count{ S : assigned(N,S,D) } > 1.

% A shift is "covered" by a nurse on a day if the nurse is assigned to the
    shift on that day.
covered(S,N,D):- assigned(N,S,D).

% Each shift must be covered by exactly one nurse on each day.
:- shift(S), day(D), #count{ N : covered(S,N,D) } != 1.
```

[5] https://www.billboard.com/pro/super-bowl-halftime-show-covid-safety-coronavirus/

[6] https://ec.ai/case-travel

[7] https://ec.ai/cogent-features

```
21
22 % A nurse is "working on" a day if the nurse is assigned to a shift on that
       day.
23 working(N,D):- shift(S), assigned(N,S,D).
24
25 % Each nurse must be working on at least two days.
26 :- nurse(N), #count{ D : working(N,D) } < 2.
```

The important aspect of the program in Listing 1.1 is that none of the statements describe an algorithm for computing a potential solution. Rather, they encode *the specification itself*. It's worth reflecting and appreciating the power of such a syntactically simple and elegant language. Compare for example this program, against the equivalent programs written in an imperative language using Google's OR-Tools [8]. The difference is stark, and it raises an important question: "Why has the logic programming approach not gained in momentum since its discovery?"

There are many potential answers to this question. One possibility is that in addition to the cognitive load incurred by switching from an imperative to a declarative mindset, there is an additional cognitive load incurred by the close relationship between logic programming languages and the notations of formal logic. This dramatically increases the distance a potential user has to mentally travel in order to get to the current state of the art. Another way to view this, is that logic programming languages on some level, are still at the level of assembly language. The *declarative paradigm* is a higher level paradigm than *imperative programming*, but declarative languages by and large are still on too low a level to be readily adopted. *If* this is true, then a natural question to ask is: "What could a high-level, structured, declarative programming language look like?"

At EC, we believe that one potential answer to this question is *structured natural language*, in particular our own version of this known as *Cogent*. Similar work in this area exists, namely Kowalski's own work on logical English [5,6], but with Cogent we are able to leverage our expertise in both natural language understanding and knowledge representation to build a more flexible, and user friendly representation language. In particular, let's revisit the program from Listing 1.1, only this time in Cogent instead of ASP:

Listing 1.2. Nurse Scheduling in Cogent

```
1  The nurses are "Andy", "Betty", and "Chris".
2
3  The days are "Monday", "Tuesday", and "Wednesday".
4
5  The shifts are "first", "second", and "third".
6
7  A nurse may be "assigned to" a shift "on" a day.
8
9  A shift may be "covered by" a nurse "on" a day.
10
11 A nurse may be "working on" a day.
12
13 We may choose for a nurse to be assigned to a shift on a day.
14
15 A nurse cannot be assigned to more than one shift on the same day.
16
17 A shift is covered by a nurse on a day if the nurse is assigned to the shift
       on that day.
```

```
18
19  Each shift must be covered by exactly one nurse on each day.
20
21  A nurse is working on a day if the nurse is assigned to a shift on that day.
22
23  Each nurse must be working on at least two days.
```

The reader will notice that with the exception of lines 7, 9, and 11, the text of the program is the same as comments from the ASP encoding in Listing 1.1. Given this program, our reasoning engine is capable of finding solutions *just as efficiently* as the ASP encoding, yet the Cogent program is more accessible to a reader. Not only that, but the fact that the language is a structured form of natural language helps bridge the gap in terms of familiarity to aspiring users. The notion of accessibility to a reader, however is of special importance, since at EC, one of our motivating goals is to help develop *explainable AI*. One important aspect of this is to render the axioms of a domain that an AI system represents both *inspectable* and *clear* to as many users as possible. This kind of transparency enables deeper human and AI partnerships which furthers our vision of artificial agents as "thought partners" capable of collaborating with humans.

Cogent has features that overlap with those found in contemporary logic programming languages, such as *non-deterministic choice*, *aggregates*, *recursive definitions*, *costs*, *preferences*, a *declarative semantics for negation*, and *contradiction diagnosis*. In addition however, it features numerous advanced term building features that facilitate the construction of clear, concise natural language expressions. Consider the solution to the N-Queens problem given in Listing 1.3

Listing 1.3. N-Queens in Cogent

```
1   # Declarations
2
3   There is exactly one "board size", which is a number.
4
5   "Queen" is a type.
6   The "Row" of a queen can be any integer from 1 to the board size.
7   The "Column" of a queen can be any integer from 1 to the board size.
8
9   A queen may be "attacking" another queen.
10
11  # Rules of the Domain
12
13  A queen cannot be attacking another queen.
14
15  A queen is attacking another queen if the first queen's row is equal to the
        second queen's row.
16
17  A queen is attacking another queen if the first queen's column is equal to
        the second queen's column.
18
19  A queen is attacking another queen if
20    A - B = C - D
21  where
22    A is the row of the first queen, and
23    B is the row of the second queen, and
24    C is the column of the first queen, and
25    D is the column of the second queen.
26
27  A queen is attacking another queen if
28    A - B = D - C
29  where
```

```
30    A is the row of the first queen, and
31    B is the row of the second queen, and
32    C is the column of the first queen, and
33    D is the column of the second queen.
```

Listing 1.3 demonstrates several term building features of Cogent, as well as a natural encoding of the constraints of the domain. In addition, the language utilizes EC's Braid reasoning engine, making it capable of scaling to advanced production applications, such as the Round-the-World travel application developed for the OneWorld Alliance. While dramatically more complex in scope than the toy examples presented above, the encoding of various rules in Cogent (such as those shown in Listing 1.4) remains not only manageable, but clearly conveys their intention to a reader:

Listing 1.4. Round-the-World Rule Sampling

```
1   At most 4 international transfers can be located in any country.
2
3   At least one selected flight leg must be arriving in each continent group.
4
5   A visit is immediately preceding another visit if
6     a selected route is going from the first visit to the second visit.
7
8   At most one selected leg can be arriving in Asia unless
9     the Asia intercontinental arrival exception is in effect.
10
11  At most two selected legs can be arriving in Asia if
12    the Asia intercontinental arrival exception is in effect.
13
14  The Asia intercontinental arrival exception is in effect if
15    a selected leg is traveling from Southwest Pacific to Asia, and
16    another selected leg is traveling from Asia to Europe.
```

In addition to bridging the linguistic gap by being a controlled form of natural language, Cogent is coupled with a powerful AI authoring assistant to help bridge the gap even further, making for a system that we believe is greater than the sum of its parts. It is our belief at EC that Cogent provides a revolution in the arena of declarative programming, and programming at large by elevating the notion of *high-level language* to a new level.

3 Conclusion

In 1972, Kowalski and Colmerauer started a revolution with the advent of the Prolog programming language. The ability to think "previously impossible thoughts", led the community to ask previously unthinkable question, sparking revolutions in natural language understanding, knowledge representation, commonsense reasoning, and other diverse areas. For a time, these fields grew in isolation from each other, and now are coming together rapidly and in profound ways. With the development of Cogent, an ultimate grandchild of Prolog in some sense, we at Elemental Cognition hope to carry forward the tradition and enable a new class of impossible thoughts to be given voice. The community owes a debt to Kowalski, Colmerauer and the Prolog Language, and the great unexplored sea they revealed to us. Happy Birthday.

References

1. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: Kowalski, R., Bowen, Kenneth (eds.) Proceedings of International Logic Programming Conference and Symposium. pp. 1070–1080. MIT Press (1988), http://www.cs.utexas.edu/users/ai-lab?gel88
2. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing **9**, 365–385 (1991)
3. Kalyanpur, A., Breloff, T., Ferrucci, D.A.: Braid: Weaving Symbolic and Neural Knowledge into Coherent Logical Explanations. Proceedings of the AAAI Conference on Artificial Intelligence **36**(10), 10867–10874 (June 2022). https://doi.org/10.1609/aaai.v36i10.21333, https://ojs.aaai.org/index.php/AAAI/article/view/21333
4. Kaufman, G.: How the NFL Pulled Off a Safe Super Bowl LV Halftime Show in the Middle of a Pandemic (2 2021), https://www.billboard.com/pro/super-bowl-halftime-show-covid-safety-coronavirus/, non paywalled version can be found at https://www.bioreference.com/how-the-nfl-pulled-off-a-safe-super-bowl-lv-halftime-show-in-the-middle-of-a-pandemic/
5. Kowalski, R., Dávila Quintero, J., Calejo, M.: Logical English for Legal Applications (11 2021)
6. Kowalski, R., Dávila Quintero, J., Sartor Galileo Calejo, M.: Logical English for Law and Education. In: Warren, D.S., Dahl, V., Eiter, T., Hermenegildo, M., Kowalski, R., Rossi, F. (eds.) Prolog - The Next 50 Years. No. 13900 in LNCS, Springer (July 2023)
7. Marriott, K., Stuckey, P.J., Wallace, M.: Handbook of Constraint Programming, chap. 12. Constraint Logic Programming, pp. 409–452. Foundations of Artificial Intelligence, Elsevier (2006)
8. Perron, L., Furnon, V.: OR-Tools, https://developers.google.com/optimization/
9. Schlipf, J.S., Ross, K.A., Van Gelder, A.: The Well-Founded Semantics for General Logic Programs. Journal of the Association for Computing Machinery **38**(3), 620–650 (1991)