People, Ideas, and the Path Ahead

Marcello Balduccini

Saint Joseph's University
Elemental Cognition
marcello.balduccini@gmail.com

Abstract. While recent advances in machine learning have yielded impressive results, researchers, practitioners, and even companies are beginning to recognize that true artificial intelligence requires much more sophisticated reasoning capabilities. Knowledge representation and declarative programming are arguably in a premier position to aid in the achievement of such capabilities. In this paper, I reflect on people and ideas that have had a great influence on my view of knowledge representation and of declarative programming. Through these lenses, I will discuss what I consider to be some of the most important milestones in the evolution of the field over the past years. I will conclude my reflection with my take on what this may tell us about the path that lies ahead and about areas where research efforts may yield considerable benefits.

Keywords: Knowledge Representation, Answer Set Programming, Non-Monotonic Reasoning, Declarative Programming, Practical Applications.

1 Introduction

I was honored to be asked to give an invited talk to PADL 2022. This article summarizes the key points of my talk.

I decided to structure my talk as reflection on people and ideas that have had a great influence on my view of knowledge representation and of declarative programming. Interestingly, these ideas have corresponded to important milestones in the evolution of the field over the past years. While many researchers have contributed to these milestones, in this article I will refer specifically to the people who have communicated those idea to me first-hand, as a way to honor their impact on my views: Michael Gelfond's idea that the Knowledge Representation (KR) methodology and Answer Set Programming (ASP) [15, 23] itself were viable for practical applications; Henry Kautz's suggestion that declarative languages of different nature could be hybridized into languages suitable for use in industrial applications; and David Ferrucci's idea that agents can be thought partners, capable of intelligently engaging humans when faced with problems beyond their individual capabilities. I conclude my reflection with my take on the path that lies ahead and about areas where research efforts may yield considerable benefits.

2 KR Methodology and Practical Applications

At the time it was conceived, Michael Gelfond's idea was a bold one: that ASP, coupled with a rigorous KR methodology, would be viable for practical applications, both in terms of convenience of use and in terms of scalability.

I was exposed to this idea when I joined Michael's lab at Texas Tech University as a fresh Ph.D. student, and got involved in Michael's and Monica Nogueira's research on the USA Advisor reasoning system [4]. At the time we started working on the USA Advisor, demonstrations of ASP were mostly limited to the level of academic exercises and initial performance evaluations (e.g., [11]). There were some concerns that ASP programs would not scale enough to be usable in practical applications, particularly when the programs were written following a rigorous KR methodology and, even more so in the case of an action-language based approach [16].

The rigorous KR methodology I am referring to is what Michael was very careful in instilling all of his students. When one is formalizing a dynamic domain, one should first of all answer the questions: What are the objects of the domain? What are the relations? What are the actions? Following this categorization, one would then proceed to representing the effects of actions in terms of dynamic causal laws, state constraints and executability conditions, either directly encoded in an action language or translated to ASP. Very importantly, fluents and actions should have a precise informal meaning stated in English. The problem should, then, be formulated first using precise English statements that (a) follow the expression patterns of laws of action languages, and (b) leverage the English phrases associated with fluents and actions. The English statements should then be translated into ASP in a direct way, so that every ASP statement, when it was read back into English, would match the original English statement.

This approach is designed to yield elegant and fully declarative specifications, which was in fact the case in the USA Advisor. Two of my favorites:[1]

```
% Tank node N1 is pressurized by tank X if it is connected
% by an open valve to a node which is pressurized by tank X.

h(pressurized_by(N1,X),T) :- time(T),
                             tank_of(N1,R),
                             link(N2,N1,V),
                             h(in_state(V,open),T),
                             tank_of(X,R),
                             h(pressurized_by(N2,X),T).

% If the input value of a NOT gate is S1 at time t and its delay
% is d then its output value is opposite value S2 at time t+d.

h(value(W2,S2),T1) :-
```

[1] For historical faithfulness, I copy them here in their entirety, including the original comments from the formalization of the USA Advisor.

```
of_type(G,not_gate),
delay(G,D),
time(T),
time(T1),
T1 = T+D,
input(W1,G),
output(W2,G),
opposite(S1,S2),
h(value(W1,S1),T),
not is_stuck(W2,G).
```

At the time when we created the USA Advisor, the first rule was a striking example of ASP's ability to elegantly capture complex kinds of knowledge. Not only the rule was part of a recursive definition, which would have been a challenge in many other languages at the time, it embedded such a recursive definition within a state constraint, thus capturing the evolution of the state of the domain over time in a way that mirrored faithfully the intuition. The second rule was, too, related to recursive definitions, but my fascination with it lies in the fact that it was in fact not a rule that we had written for the USA Advisor.

Some time before we focused on the USA Advisor, we had worked on using ASP and the KR methodology for representing digital circuits and reasoning about them [3]. This rule is part of what we called the "General Theory of Digital Circuits in A-Prolog."

Thus, what is striking to me is that, even at that time, Michael had such a clear conception of modularity in ASP, that it allowed us to reuse in the USA Advisor a rather sophisticated module that we had previously created independently of it, and we were able to do so without any substantial modifications.

But what was the scalability of the formalization? Did it scale to the needs of this first industrial application of ASP? Indeed, it was, and to some extent it surprised even us by how well it performed. We had been given a time threshold of 20 minutes for every problem instance and we found that the system was, on average, orders of magnitude faster than that and never got even close to that threshold even in the most challenging cases.

The fact that we were concerned about scalability is clearly visible in certain statements. For instance, we preferred to ground manually the rules encoding the law of inertia and in doing so we restricted the groundings only to the fluents that we knew "mattered", e.g.

```
% Tanks maintain correct pressure unless some leak
% occurs along their path for some time.

h(pressurized_by(X,X),T1) :- next(T,T1),
                             tank_of(X,R),
                             h(pressurized_by(X,X),T),
                             not nh(pressurized_by(X,X),T1).
```

This would be unnecessary nowadays. In fact, given what we learned later about the performance of the system, in hindsight we could have probably trusted the grounding of the rules of inertia to the lparse grounder. However, this shows the level of concern and lack of a generally clear picture of what might affect performance and what might not.

Something that is important to remark is that this success was not only due to Michael's great KR capabilities and intuition. Instrumental to the success of this project was also the excellent work by Illka Niemela, Tommi Syrjanen, and others that had given us grounder lparse and solver smodels [25].

3 Hybrid Declarative Languages for Practical Applications

Henry Kautz was my supervisor when I joined the Eastman Kodak Research Labs. I had been asked to investigate ways to automate decision-making processes of commercial print shops: given a set of print jobs (e.g., books or magazines), which presses, cutters, binders, and other devices should be used, which device configurations would minimize waste and costs, and what was the best schedule for the work? The agent should also be able to respond to sudden unexpected events, such as devices becoming unavailable or "rush jobs" coming it while others are already in production. The response should consist of incremental changes that minimize disruptions and additional costs.

It was known in the industry that, when humans experts were carrying out those decision-making processes, an important factor of their success was the heuristic knowledge that they had accumulated over the years – so much so that (I was told) experienced people were paid substantial salaries and were regarded as key elements of the manufacturing process. Because of this, an additional constraint I was given was that the system should make it possible to easily incorporate heuristic knowledge as it might be provided by human experts.

On the one side, I expected the expert knowledge to be in the form of commonsensical statements, possibly defaults and their exceptions, and so it was clear to me that a non-monotonic language like ASP would be most appropriate. On the other hand, intuition and preliminary experiments showed me that ASP-based formalizations of this underlying "planning-while-scheduling" problem would not scale well for practical use. It seemed clear that a representation based on Constraint Satisfaction Problems (CSP) [19] would be most appropriate for that.

Of course, the underlying problem was that whichever approach I adopted, would need to be viable for a practical, industry-sized application. I was explaining all of this to Henry in a meeting, when he pointed me to the approach taken in Satisfaction Modulo Theories (SMT) [26] as a possible solution.

SMT had demonstrated that it was possible to overcome the expressive shortcomings and performance shortcomings of individual declarative languages by hybridizing them. The bet was that ASP and Constraint Programming could be hybridized in a way that allowed us to solve this "planning-while-scheduling"

problem augmented with expert knowledge, and to do so in a way that was scalable. [2]

The idea of hybridizing ASP and CP was not entirely new. A couple of years earlier, Baselice, Bonatti and Gelfond had published a paper proposing a possible approach [9], and I was aware that Veena Mellarkod, Michael Gelfond and Yuanlin Zhang had been working for some time on a related implementation [24]. However, what was surprising – almost shocking – to me was the proposal of using such a hybrid for a practical application, when the work I was aware of had been limited to academic exercises.

Sure enough, preliminary tests on Mellarkod's system showed that it would not scale to the type of application we were building. Additionally, Veena's system leveraged specific solving algorithms. That was a problem for me, since there was substantial uncertainty in my mind on exactly which solvers to use. Lparse and smodels? The new gringo grounder [12] and clasp solver [13]? And which constraint solver? Ilog, to which I had been introduced in a previous Kodak project? The constraint solver embedded in some Constraint Logic Programming (CLP) system? And should the constraint solver need to support finite domains only or larger domain variables?

This prompted me to develop EZCSP [1, 6, 5], which I literally intended as an easy (hence "EZ") way of encoding CSP by means of a host language featuring strong KR foundations and support for non-monotonic constructs. The view of ASP as a host language for a constraint satisfaction language allowed me to adopt a loosely coupled view of the two languages and of the underlying solvers, making it possible to experiment with multiple combinations of ASP solvers and constraint solvers, as well as different types of variable domains.

Henry's intuition proved to be a good one. Developing EZCSP as a lightweight layer that leveraged existing solvers without modifications allowed me to conduct experiments with various solver combinations and quickly identify one that scaled to the level needed for our application. Remarkably, the first use of EZCSP coincided with its first industrial-sized, deployed application [1], which to the best of my knowledge was also the first deployed application of what we later came to call Constraint ASP [5]. The final product, which was running EZCSP at its core, even came with a user interface designed by UX expert Stacie Hibino.

I think all of this is an amazing demonstration of the power of constraint-based languages, a term that I use to denote both ASP and CP languages, as well as of Henry's intuition about the potential of a hybrid solution of ASP and CP.

[2] As an aside, our work on the USA Advisor had also been influenced by one of Henry's ideas, specifically Kautz and Selman's work on solving planning problems by reducing them to satisfiability problems [20].

4 Intelligent Agents as Thought Partners

David Ferrucci, the creator of IBM Watson, currently heads Elemental Cognition[3] (EC), an AI technology company aiming at developing and using a full spectrum of AI techniques to deliver revolutionary products that solve challenging, real-world problems.

When I first started collaborating with EC, two aspects fascinated me: on one side, David and EC fully understood the importance of KR, non-monotonic reasoning and declarative programming in achieving sophisticated agent capabilities. On the other side, David was adamant in his vision that agents should be "thought partners." As such, they should collaborate with humans rather than just acting autonomously. One particularly striking instance of this is that it may well happen that agent may sometimes be unable to solve a certain problem, either due to lack of knowledge or to performance limitations. In those cases, the agent should be capable of recognizing the issue and of engaging the humans in intelligent ways, in order to overcome the obstacle together.

This latter idea was quite striking to me, as it was in partial divergence with idea of agents as fully autonomous, to which I had subscribed since my early steps in AI. Then again, while at a first glance it may seem that giving up full autonomy might be an indication of a reduced level of intelligence, David's idea of agents as thought partners requires in fact even greater intelligence.

David's vision has struck a chord with a several parties, and has already led to successful projects. One of my favorite is a project that contributed to making the Superbowl possible in 2021 in spite of having been held in the midst of the covid-19 pandemic.[4] EC's PolicyPath app and underlying systems were involved in enforcing the access policy for corporate-level employees – approximately 40,000 people.

There were a number of important lessons that we learned from this project in relation to the use of KR, some of which are discussed in more details in [2]. In a nutshell, we faced even more challenges than I had faced before, or than I had expected.

On the one side, intelligent interaction with a user requires a level of introspection by the agent that is extremely challenging. Together with that comes the agent's need to explain its reasoning and its conclusions in terms that the user can understand – likely in an interactive dialogue – which is another very challenging task.

However, there are also challenges at the level of more typical KR and reasoning tasks, which were surprising to me. While reasoning about actions and change has made enormous progress since John McCarthy's seminal papers [17], some types of statements that are straightforward for humans are still difficult to formalize precisely and efficiently. I will summarize here some of these observations.

[3] https://ec.ai

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

Consider for instance the statement "after international travel, one is not allowed access to the office for 14 days." Formalizing such statements requires some notion of "wall-clock" time and mechanisms allowing fluents to change value without intervening observations.

Action languages such as $\mathcal{H}$ [10], while possibly suitable, have a complex semantics and can reduce performance considerably due to the complexity of the underlying implementation. Additionally, in all access policies we studied, time and change were simpler than in typical uses of $\mathcal{H}$ and did not justify its use. Additive fluents [21] offer a potential solution, but once again appeared to be more sophisticated than needed, and it is unclear how to conveniently cause value changes in fluents, even if one were to formalize them via triggers. For instance, the statement "one is not allowed access to the office for 14 days" requires the ability to count down the given amount of time and to cause a fluent, say, *has_access* to be false for that duration and to become true at the end unless other causes intervene. While additive fluents could be used to represent the amount of time left, changing that value over time in a convenient way seems less straightforward. An additional challenge was that one would also want somehow to ensure that *has_access* is allowed to revert to true at the end of that period.

We were able to solve this challenge by introducing the notion of "timed" fluents – essentially, numerical fluents whose value naturally decays (or, in principle, increases) over time. However, this raised a performance problem: representing explicitly every state and state transition related to the evolution of a timed fluent becomes problematic when one is considering a long period of time, e.g. in the order of months or years. That is because, in principle, every change in the value of a timed fluent causes a state transition. It should be noted that this is strongly related to the problem faced in formalizing hybrid domains discussed in [10] and preceding articles.

With inspiration from that line of research, we realized that only part of those state transitions are critical to the evolution of the domain. In the example above, states remain "sufficiently" similar to each other until the timed fluent reaches 0, unless of course other causes intervene. We were thus able to refine the representation in such a way that only the "relevant" states were explicitly considered, and rules of the formalization itself were responsible for determining which states were relevant. For instance, in our example, there is only one "relevant" state 14 days from now.

While this approach yielded a satisfactory solution for medium-sized problem instances, our experiments showed that in the presence of larger time horizons or when more complex policies were considered, near real-time interaction with the user was still beyond reach.

One particularly interesting case was that of a subject for whose state the agent has already calculated the evolution over time, possibly for several months. Note that this evolution does not necessarily need to be only future-facing. It may contain past observations for a prolonged amount of time and the agent needs to be able to consider how the subject's state evolved in response to those

observations – as the user may want to ask questions about them – and how those observations may affect future states.

Suppose now that a new observation is received, such as a covid test result. The information might be about the current moment in time, but could also be previously unavailable information about a past moment. Recomputing the entire evolution from scratch was found to take several minutes, making near real-time interaction impossible.

The obvious solution was to adopt the approach of an incremental computation, especially leveraging clingo's incremental solving capabilities [18]. Indeed, clingo's incremental solving has been demonstrated to yield substantial performance improvements. For instance, in planning problems, one can look for a plan up to a certain maximum length and, if no such plan is found, one can have the solver incrementally consider additional time steps, and do so by reusing and extending the search space built for the prior computation rather than recreating it from scratch every time.

Unfortunately, this approach did not quite seem to work for our use case, where the observations may trigger a recomputation of past states or may cause the discovery of new "relevant" states between states that had already been computed. Situations of these types violate the conditions of the Module Theorem [27] and are thus not directly solvable with clingo's incremental solving capabilities.

We were eventually able to solve the problem by developing algorithms that essentially "roll back" the search space to the latest point in the search process from which incremental computations could be applied. While this allowed us to solve the problem at hand – and, in fact, improve performance by multiple orders of magnitude – the solution was quite specific to the particular formulation and problem domain, and to the best of our knowledge, no general solution is currently available.

EC's efforts have highlighted a number of additional, and very interesting, challenges, which are outside of the scope of this article, such as those related to the prevalence of incomplete knowledge in practical applications, more so than research exercises typically consider, and to the need to draw at least partial inferences in spite of such incomplete knowledge – all while maintaining the ability to interact with the user in near real-time.

5 Conclusion

In this paper, I have reflected on people and ideas that have had a great influence on my view of knowledge representation and of declarative programming, and that have coincided with what I consider to be important milestones in the evolution of our field, some of which I have been honored to be involved with at least in part: the idea that a rigorous KR methodology and ASP itself could be viable for practical applications, the idea that declarative languages of different nature could be hybridized into languages suitable for use in industrial applications, and the idea that agents could be thought partners, capable of in-

telligently engaging humans when faced with problems beyond their individual capabilities.

While enormous progress has been made over the past decades, much work still remains in order to design agents that are truly intelligent and can act as such thought partners. Work has already been under way for several years on explanatory reasoning and on building more efficient ASP solvers that rely on non-ground solving techniques. However, research will also be needed in other directions. In particular, the research on incremental solving so far seem to have only scratched the surface and will need to be extended beyond the confines of the Module Theorem. Additionally, research is needed on representation, and specifically on techniques for declaratively, but efficiently, stating the conditions under which an agent should stop reasoning and seek assistance – and what to ask for. At least in part, prior work on sensing actions is a starting point (e.g., [22, 7, 8], as is the research on epistemic specification (e.g., [14]).

References

1. Balduccini, M.: Industrial-Size Scheduling with ASP+CP. In: Delgrande, J.P., Faber, W. (eds.) 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR11). Lecture Notes in Artificial Intelligence (LNCS), vol. 6645, pp. 284–296. Springer Verlag, Berlin (2011)
2. Balduccini, M., Barborak, M., Ferrucci, D.: Action Languages and COVID-19: Lessons Learned. In: 2nd Workshop on Causal Reasoning and Explanation in Logic Programming (CAUSAL2020) (2020)
3. Balduccini, M., Gelfond, M., Nogueira, M.: A-Prolog as a tool for declarative programming. In: Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering (SEKE'2000). pp. 63–72 (2000)
4. Balduccini, M., Gelfond, M., Nogueira, M.: Answer Set Based Design of Knowledge Systems. Annals of Mathematics and Artificial Intelligence **47**(1–2), 183–219 (2006)
5. Balduccini, M., Lierler, Y.: Constraint Answer Set Solver EZCSP and Why Integration Schemas Matter. Journal of Theory and Practice of Logic Programming (TPLP) **17**(4), 462–515 (2017)
6. Balduccini, M., Lierler, Y., Schuller, P.: Prolog and ASP Inference Under One Roof. In: Cabalar, P., Son, T.C. (eds.) 12th International Conference on Logic Programming and Nonmonotonic Reasoning (Sep 2013)
7. Baral, C., McIlraith, S.A., Son, T.C.: Formulating diagnostic problem solving using an action language with narratives and sensing. In: Proceedings of the 2000 KR Conference. pp. 311–322 (2000)
8. Baral, C., Son, T.C.: Formalizing sensing actions – a transition function based approach. Artificial Intelligence Journal **125**(1–2), 19–91 (Jan 2001)
9. Baselice, S., Bonatti, P.A., Gelfond, M.: Towards an Integration of Answer Set and Constraint Solving. In: Proceedings of ICLP 2005 (2005)
10. Chintabathina, S., Watson, R.: Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday, chap. A New Incarnation of Action Language H, pp. 560–575. Lecture Notes in Artificial Intelligence (LNCS), Springer Verlag, Berlin (2011)

11. Erdem, E.: Application of Logic Programming to Planning: Computational Experiments (1999), http://www.cs.utexas.edu/users/esra/papers.html
12. Gebser, M., Kaminski, R., Ostrowski, M., Schaub, T., Thiele, S.: On the input language of ASP grounder gringo. In: Erdem, E., Lin, F., Schaub, T. (eds.) 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR09). Lecture Notes in Artificial Intelligence (LNCS), vol. 5753, pp. 502–508. Springer Verlag, Berlin (Sep 2009)
13. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-Driven Answer Set Solving. In: Veloso, M.M. (ed.) Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07). pp. 386–392 (2007)
14. Gelfond, M.: New Semantics for Epistemic Specifications. In: Delgrande, J.P., Faber, W. (eds.) 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR11). Lecture Notes in Artificial Intelligence (LNCS), vol. 6645, pp. 260–265. Springer Verlag, Berlin (2011)
15. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing **9**, 365–385 (1991)
16. Gelfond, M., Lifschitz, V.: Representing Action and Change by Logic Programs. Journal of Logic Programming **17**(2–4), 301–321 (1993)
17. Hayes, P.J., McCarthy, J.: Some Philosophical Problems from the Standpoint of Artificial Intelligence. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence 4, pp. 463–502. Edinburgh University Press (1969)
18. Kaminski, R., Schaub, T., Wanko, P.: A Tutorial on Hybrid Answer Set Solving with Clingo. In: Proceedings of the Thirteenth International Summer School of the Reasoning Web (RW-2017). pp. 167–203 (2017)
19. Katriel, I., van Hoeve, W.J.: Handbook of Constraint Programming, chap. 6. Global Constraints, pp. 169–208. Foundations of Artificial Intelligence, Elsevier (2006)
20. Kautz, H., Selman, B.: Planning and Satisfiability. In: Proceedings of the 10th European Conference on Artificial Intelligence (ECAI92). pp. 359–363 (1992)
21. Lee, J., Lifschitz, V.: Additive Fluents. In: Provetti, A., Son, T.C. (eds.) Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning. AAAI 2001 Spring Symposium Series (Mar 2001)
22. Levesque, H.J.: What is planning in the presence of sensing? In: Proceedings of the 13th National Conference on Artificial Intelligence. pp. 1139–1146 (1996)
23. Marek, V.W., Truszczynski, M.: The Logic Programming Paradigm: a 25-Year Perspective, chap. Stable Models and an Alternative Logic Programming Paradigm, pp. 375–398. Springer Verlag, Berlin (1999)
24. Mellarkod, V.S., Gelfond, M., Zhang, Y.: Integrating Answer Set Programming and Constraint Logic Programming. Annals of Mathematics and Artificial Intelligence (2008)
25. Niemelä, I., Simons, P.: Logic-Based Artificial Intelligence, chap. Extending the Smodels System with Cardinality and Weight Constraints, pp. 491–521. Kluwer Academic Publishers (2000)
26. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT Module Theories: From an Abstract Davis-Putnam-Longemann-Loveland Procedure to DPLL(T). Journal of Artificial Intelligence Research **53**(6), 937–977 (2006)
27. Oikarinen, E., Janhunen, T.: Modular Equivalence for Normal Logic Programs. In: Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI'06). pp. 412–416 (2006)