How to upgrade ASP for true dynamic modelling and solving?

Torsten Schaub*
Inria, Rennes, France and University of Potsdam, Germany
torsten@cs.uni-potsdam.de

1 Motivation

The world is dynamic, and ASP is not! This is a provocative way to say that ASP is not up to dealing with many complex real-world applications having a dynamic nature, let alone transitions over states, not even mentioning more fine-grained temporal structures.

Although ASP has already been applied in various domains in academia and industry with remarkable success [8, 9], a closer look reveals that this concerns mostly static or smaller dynamic domains. For example, ASP is highly competitive in static domains such as timetabling [2] and workforce management [19], whereas it lags behind when it comes to substantial dynamic ones, as for instance robotic intra-logistics as discussed below. In fact, there is still quite a chasm between its level of development for addressing static and dynamic domains. This is because its modeling language as well as its grounding and solving machinery aims almost exclusively at handling static knowledge, while dynamic knowledge is usually indirectly dealt with via reductions to the static case.

In order to overcome this barrier and to upgrade ASP to the next level, dealing with complex dynamic problems, three areas appear to be relevant to me.[1]

2 Modeling

The most popular languages for modeling dynamic systems in the realm of ASP are temporal extensions of Equilibrium Logic [1], the host logic of ASP, and action languages [14]. Although both constitute the main directions of non-monotonic temporal systems, their prevalence lags way behind the usage of plain ASP for

*Torsten Schaub has been supported by the German Science Foundation (DFG): projects SCHA 550/9 and 11.

[1]This personal view comes with a lot of self references — sorry for that!

modeling dynamic domains. Hence, notwithstanding the meticulous modeling of dynamics in ASP due to an explicit representation of time points, it seems that its pragmatic advantages, such as its rich (static) modeling language and readily available solvers, often seem to outweigh the firm logical foundations of both dedicated approaches. Although the true reasons are arguably inscrutable, let us discuss some possible causes.

The appeal of action languages lies in their elegant syntactic and semantic simplicity: they usually consist of static and dynamic laws inducing a unique transition system. Although most of them are implemented in ASP, their simplicity denies the expressive modeling possibilities of ASP. Also, despite some recent reconciliation [16], existing action languages lack the universality of ASP as reflected by the variety of variants.

Temporal Equilibrium Logic (TEL; [1]) builds upon an extension of the logic of Here-and-There [15] with Linear Temporal Logic (LTL; [18]). This results in an expressive non-monotonic modal logic, which relies upon the general syntax of LTL and possesses a computational complexity beyond LTL [3]. As in LTL, a model in TEL is an *infinite* sequence of states, called a trace. This rules out computation by ASP technology (and necessitates model checking) and is somewhat unnatural for applications like planning, where plans amount to finite prefixes of one or more (infinite) traces.

One proposal to overcome these issues is to restrict TEL to finite traces, similar to the restriction of LTL to LTL_f advocated in [5]. This is detailed in [4] and accompanied with an extension of the ASP system *clingo*, dubbed *telingo* and available at `https://github.com/potassco/telingo`. *telingo* extends the full-fledged input language of *clingo* with temporal operators and computes temporal models incrementally by multi-shot solving [11].

3 Encoding and Solving

The need for dedicated encoding and solving techniques for handling dynamic domains stems from the necessity to implement fluents, that is, propositions changing their value over time. In ASP, just as other constraint-based approaches like CP or SAT, this amounts to creating a copy of each fluent and related rules per time point. The reduction of the resulting redundancy is the primary target of the aforementioned dedicated reasoning techniques.

First of all, we should realize that modeling and encoding a dynamic domain may amount to quite different specifications, both being declarative but aiming at different conceptions at distinct levels of the domain.[2] The easiest way to realize the difference between modeling and encoding is to consider a temporal rule $a(X) \leftarrow \bullet b(X)$ in which '$\bullet$' denotes the "previous" operator, or in *telingo* syntax: `a(X) :- 'b(X)`, that is finally encoded as `a(X,t) :- b(X,t-1)` where

[2] Another good example for this are (arithmetic) CSPs, nicely modeled by expressions like $x + y < 7$ but usually best implemented in ASP (and SAT) via an order encoding [23] treating integer variables by inequalities like, $x \leq 1, x \leq 2, \ldots$ (rather than a direct encoding using equalities $x = 1, x = 2, \ldots$).

the parameter '`t`' is handled by multi-shot solving [11]. Obviously, modeling is ideally more abstract than encoding by dropping aspects like the implementation of time by increasing integers. Also, the targeted implementation using parameters '`t`' (instead of variables '`T`') remains hidden. But apart from this abstraction, no real gain is obtained as regards the elimination of redundancy. Unlike this, multi-shot solving cuts back redundancies by avoiding repeated grounding and solving efforts.

Much more is possible. Encoding-wise an exemplar is the *parallel* representation of sequential plans which has been investigated in SAT planning [7, 21]. A first attempt to transfer this to ASP is given in [6].[3] Another example is multi-path planning in logistics warehouses, where a two-step abstraction encoding technique was used [17] to scale up to state of the art algorithms. Certainly, many more such principled techniques exist but are no matter of common knowledge. Solving-wise, the semantic links between the aforementioned fluent copies need to be exploited. Again, SAT planning serves us as an exemplar, where heuristics were used in [20] to provide such links. This idea has led to the heuristic directives in *clingo* [12]. Another solving technique was put forward in [10], where ground multi-state invariants are extracted and generalized in order to be fed back into the solving process, thus extending their scope to all similar state combinations.[4] And much more can and needs to be done!

4 Benchmarking

The upgrade of ASP is moreover threatened by a lack of complex benchmark scenarios mimicking the needs of dynamic real-world applications. In contrast to many available benchmark suites, often supplied by automatic instance generators, real-world applications are rarely disseminated, either because they are classified or come only with a handful of instances. Another commonality of existing benchmark suites is that they are kept simple, stick to basic ASP, and usually feature at most a single specifics, so that they can be processed by as many systems as possible. However, this is in contrast to many real-world applications whose solution requires the integration of multiple types of knowledge and forms of reasoning. Last but not least, a feature distinguishing ASP from all other solving paradigms is its versatility, which is best put in perspective by solving multi-faceted problems.

The fear is thus that the lack of complex benchmark scenarios becomes a major bottleneck in ASP's progression towards real-world applications, and hence that more and more should be made available to our community. As a first step to overcome this problem, we propose in [13] the domain of *robotic intra-logistics*, a key domain in the context of the fourth industrial revolution, as witnessed by Amazon's Kiva, GreyOrange's Butler, and Swisslog's CarryPick systems.[5]

[3] This is implemented in the *plasp* system available at `https://github.com/potassco/plasp`.

[4] This is implemented in the *ginkgo* system available at `https://github.com/potassco/ginkgo`.

[5] `www.amazonrobotics.com`, `www.greyorange.com/products/butler`, `www.swisslog.com/`

All of them aim at automatizing warehouse operations by using robot vehicles that drive underneath mobile shelves and deliver them to picking stations. From there, workers pick and place the requested items in shipping boxes. Apart from the great significance of this real-world domain, our choice is motivated by several aspects. First of all, the domain is highly dynamic. At the same time, the warehouse layout is grid-based and thus provides a suitable abstraction for modeling robot movements in ASP. Moreover, the domain offers a great variety of manifold problem scenarios that can be put together in an increasingly complex way. For instance, one may start with single or multi-robot path-finding scenarios induced by a set of orders that are accomplished by using robots for transporting shelves to picking stations. This can be extended in various ways, for example, by adding shelf handling and delivery actions, considering order lines with multiple product items, keeping track of the number of ordered and/or stored product items, modeling energy consumption and charging actions, taking into account order frequencies, supplies, and priorities, striving for effective layouts featuring dedicated locations, like highways or storage areas, and so on. Finally, the domain is extremely well-suited for producing scalable benchmarks by varying layouts, robots, shelves, orders, product items, etc. Inspired by this, we have developed the benchmark environment *asprilo* [13] consisting of four parts (i) a benchmark generator, (ii) a solution checker, (iii) a benchmark and solution visualizer, and (iv) a variety of reference encodings. The design of *asprilo* was driven by the desire to create an easily configurable and extensible framework that allows for generating scalable, standardized benchmark suites that can be visualized with and without a corresponding solution. Correctness can be established via a modular solution checker. The accompanying reference encodings may serve as exemplary bases for extended encodings addressing more complex scenarios. The *asprilo* framework is freely available at `https://potassco.org/asprilo`.

5 Summary

Many people well beyond our community get interested by the modeling and solving capabilities of ASP, its elegance, succinctness, transparency, and last but not least its effectiveness. We attract them by showcasing *our* exemplary problems and solutions. But at the end of the day, if we want to keep them interested in ASP, we have to solve *their* problems.

Acknowledgments I am grateful to Martin Gebser, Roland Kaminski, and the four reviewers for their constructive comments! As well, I would like to mention that many aspects arose in very fruitful collaborations with Mutsunori Banbara, Pedro Cabalar, Vladimir Lifschitz, Orkunt Sabuncu, and Tran Can Son, not to mention the terrific Potassco people! Thank you all!!

`carrypick`

References

[1] F. Aguado, P. Cabalar, M. Diéguez, G. Pérez, and C. Vidal. Temporal equilibrium logic: a survey. *Journal of Applied Non-Classical Logics*, 23(1-2):2–24, 2013.

[2] M. Banbara, K. Inoue, B. Kaufmann, T. Okimoto, T. Schaub, T. Soh, N. Tamura, and P. Wanko. teaspoon: Solving the curriculum-based course timetabling problems with answer set programming. *Annals of Operations Research*, 2018. To appear.

[3] L. Bozzelli and D. Pearce. On the complexity of temporal equilibrium logic. In *Proceedings of the Thirtieth Annual Symposium on Logic in Computer Science (LICS'15)*, pages 645–656. IEEE Computer Society Press, 2015.

[4] P. Cabalar, R. Kaminski, T. Schaub, and A. Schuhmann. Temporal answer set programming on finite traces. *Theory and Practice of Logic Programming*, 2018. To appear.

[5] G. De Giacomo and M. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In Rossi [22], pages 854–860.

[6] Y. Dimopoulos, M. Gebser, P. Lühne, J. Romero, and T. Schaub. plasp 3: Towards effective ASP planning. In M. Balduccini and T. Janhunen, editors, *Proceedings of the Fourteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'17)*, volume 10377 of *Lecture Notes in Artificial Intelligence*, pages 286–300. Springer-Verlag, 2017.

[7] Y. Dimopoulos, B. Nebel, and J. Köhler. Encoding planning problems in nonmonotonic logic programs. In S. Steel and R. Alami, editors, *Proceedings of the Fourth European Conference on Planning*, volume 1348 of *Lecture Notes in Artificial Intelligence*, pages 169–181. Springer-Verlag, 1997.

[8] E. Erdem, M. Gelfond, and N. Leone. Applications of ASP. *AI Magazine*, 37(3):53–68, 2016.

[9] A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe, and E. Teppan. Industrial applications of answer set programming. *Künstliche Intelligenz*, 2018. To appear.

[10] M. Gebser, R. Kaminski, B. Kaufmann, P. Lühne, J. Romero, and T. Schaub. Answer set solving with generalized learned constraints. In M. Carro and A. King, editors, *Technical Communications of the Thirty-second International Conference on Logic Programming (ICLP'16)*, volume 52, pages 9:1–9:15. Open Access Series in Informatics (OASIcs), 2016.

[11] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming*, 2018. To appear.

[12] M. Gebser, B. Kaufmann, R. Otero, J. Romero, T. Schaub, and P. Wanko. Domain-specific heuristics in answer set programming. In M. desJardins and M. Littman, editors, *Proceedings of the Twenty-Seventh National Conference on Artificial Intelligence (AAAI'13)*, pages 350–356. AAAI Press, 2013.

[13] M. Gebser, P. Obermeier, T. Otto, T. Schaub, O. Sabuncu, V. Nguyen, and T. Son. Experimenting with robotic intra-logistics domains. *Theory and Practice of Logic Programming*, 2018. To appear.

[14] M. Gelfond and V. Lifschitz. Action languages. *Electronic Transactions on Artificial Intelligence*, 3(6):193–210, 1998.

[15] A. Heyting. Die formalen Regeln der intuitionistischen Logik. In *Sitzungsberichte der Preussischen Akademie der Wissenschaften*, page 42–56. Deutsche Akademie der Wissenschaften zu Berlin, 1930. Reprint in Logik-Texte: Kommentierte Auswahl zur Geschichte der Modernen Logik, Akademie-Verlag, 1986.

[16] J. Lee, V. Lifschitz, and F. Yang. Action language BC: Preliminary report. In Rossi [22], pages 983–989.

[17] V. Nguyen, P. Obermeier, T. Son, T. Schaub, and W. Yeoh. Generalized target assignment and path finding using answer set programming. In C. Sierra, editor, *Proceedings of the Twenty-sixth International Joint Conference on Artificial Intelligence (IJCAI'17)*, pages 1216–1223. IJCAI/AAAI Press, 2017.

[18] A. Pnueli. The temporal logic of programs. In *Proceedings of the Eighteenth Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.

[19] F. Ricca, G. Grasso, M. Alviano, M. Manna, V. Lio, S. Iiritano, and N. Leone. Team-building with answer set programming in the gioia-tauro seaport. *Theory and Practice of Logic Programming*, 12(3):361–381, 2012.

[20] J. Rintanen. Planning as satisfiability: heuristics. *Artificial Intelligence*, 193:45–86, 2012.

[21] J. Rintanen, K. Heljanko, and I. Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12-13):1031–1080, 2006.

[22] F. Rossi, editor. *Proceedings of the Twenty-third International Joint Conference on Artificial Intelligence (IJCAI'13)*. IJCAI/AAAI Press, 2013.

[23] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling finite linear CSP into SAT. *Constraints*, 14(2):254–272, 2009.