Reasoning in Highly Reactive Environments

Francesco Pacenza

Department of Mathematics and Computer Science
University of Calabria
Rende, Italy

pacenza@mat.unical.it

Advisor: Giovambattista Ianni

Department of Mathematics and Computer Science
University of Calabria
Rende, Italy

ianni@mat.unical.it

1   Introduction and problem description

The aim of my Ph.D. thesis concerns *Reasoning in Highly Reactive Environments*.

As *reasoning in highly reactive environments*, we identify the setting in which a knowledge-based agent, with given goals, is deployed in an environment subject to repeated, sudden and possibly unknown changes. This is for instance the typical setting in which, e.g., artificial agents for video-games (the so called "bots"), cleaning robots, bomb clearing robots, and so on are deployed. In all these settings one can follow the classical approach in which the operations of the agent are distinguished in "sensing" the environment with proper interface devices, "thinking", and then behaving accordingly using proper actuators [2].

In order to operate in an highly reactive environment, an artificial agent needs to be:

- *Responsive* → The agent must be able to react repeatedly and in a reasonable amount of time;

- *Elastic* → The agent must stay reactive also under varying workload;

- *Resilient* → The agent must stay responsive also in case of internal failure or failure of one of the programmed actions in the environment.

Nowadays, thanks to new technologies in the field of Artificial Intelligence, it is already technically possible to create AI agents that are able to operate in reactive environments. Nevertheless, several issues stay unsolved, and are subject of ongoing research.

2   Goal of the research

With the term *data streams* we identify an unbounded sequences of time-varying data elements, that means a "continuous" flow of information. The assumption is that recent information are more relevant as they describe the current state of a dynamic system. In the latest years, due to the increasing volume of data streams and the crucial time requirements of many applications, different real-time approaches have been studied in order to ensure that a query answer is pushed as soon as it becomes available to the consumers. Notably, in the field of stream reasoning, some work aim at introducing advanced reasoning capabilities under a well understood and formalized framework [3,4]. By the way, *Stream Reasoning* is a quite young and unexplored area, and therefore many different issues need to be solved. To this end we aim at extending the functionality of the already existing tools in order to give them the possibility to work also in highly reactive environment in a context in which the amount of data coming each second is relatively small, yet time requirements are very strict. In order to achieve our goal we focus on the usage of the pure *Answer Set Programming* (ASP) semantics in a repeated evaluation setting. In this

respect, we aim to study incremental techniques able to reuse previously computed inferences, allowing to knock down the time required during the reasoning phase. We aim to achieve this goal particularly focusing on the optimization of the grounding step. In addition to this, we would like to develop also an incremental answer set solver able to evaluate solutions at runtime in subsequent, similar, evaluation shots. Moreover, we are currently working on a framework able to ease the development of AI-agents in highly reactive contexts such as video-games and real robots. Some of these results have been already achieved and will be briefly described in sections 5.

3   Background and overview of the existing literature

During the last years, some different approaches have been explored in order to create responsive agents that are able to operate in highly reactive environments. Most of them make use of *deep learning techniques* in order to let the agent be able to emulate human learning. These techniques are applied in the field of video-games for automatic game playing accordingly to [19]. The same technique can be also used for predicting human behavior as discussed in [17]. This kind of approaches have their "limits" especially when one aims to *a)* generalize reasoning (indeed, deep learning techniques are usually built ad-hoc for a specific operating environment); *b)* create development and reasoning tools that can be "used" as middleware for the production of the so called "bots", in which "intelligent" capabilities can be tuned, refined and prototyped; and *c)* provide intelligible explanations of choices taken by the agent.

In the *Answer Set Programming (ASP)* community there have been some recent studies about reasoning in reactive environments; in particular, researchers recently proposed ASP-based forms of reactive reasoning. One of these (called "*oclingo*") allows to implement real-time dynamic systems running online in changing environments (as described in [15]). This new technology paves the way for applying ASP in many areas like robots, bots, and video-games due to the fact that now an ASP solver can be also adopted for online usage. However *oclingo* cannot be defined as totally "declarative". Indeed, the user has to specify how the environment will evolve and it is necessary to specify the operational design of some *base*, *cumulative* and *volatile* groups of rules, that should be respectively evaluated only one time, at each iteration and only in one iteration before being discarded.

A second promising ASP extension like *HEX* [1] provides the possibility of a bidirectional access to external sources of knowledge and/or computation using the concept of external atoms; the extension *ActHEX* [14] of HEX programs introduces the notion of action atoms, which are associated to corresponding functions capable of actually changing the state of external environments. Using this ASP extension it is in practice possible to realize artificial agents able to operate in many different settings (for instance, an HEX extension has been successfully used for implementing an AI agent operating in video-games scenarios as described in [7]). However, ActHex does not make explicitly possible to check and act if the planned actions in the schedule have failed at each shot. Moreover, in very complex scenarios, the implementation of ActHex knowledge bases could become very hard and unhandy due to the complex ways for programming the order of actions composing a scheduled plan. Furthermore, we found in literature a large variety of languages designed for programming logical Agents and Multi-Agent Systems in contexts very similar to our setting. One of these is Jason [5]; Jason is a platform for developing agents based on the Agent-Speak logic language. A Jason agent consists of a set of plans, each of which has a triggering event, a context, and a body of actions. Moreover also other languages like IMPACT [13], DALI [10], ALP [12] and ASTRA [11] can be used to design logical Agents and Multi-Agent Systems.

4   Current status of the research

In the latest years, also the *Answer Set Programming* community focused on *Stream Reasoning*. Implementing stream reasoning with a typical ASP system requires to start each computation from scratch again and again at each evaluation shot. Indeed, the typical structure of an ASP system mimics such way of defining the semantics by relying on a grounding module that takes in input a non-ground logic program P and produces an equivalent propositional theory $gr(P)$; the grounder is coupled with a subsequent solver module that applies proper resolution techniques on $gr(P)$ for computing the actual semantics in form of *answer sets $AS(gr(P))$* [20].

In the context of stream reasoning and multi-shot reasoning engines [4, 6, 16] based on answer set programming, a quite typical setting is when the grounding step is repeatedly executed on slightly different input data, while a short computation time window is allowed. In this respect, we decided to focus our attention on the following open issues:

1. reuse as much as possible of the previously computed knowledge in logic programs;

2. knocking down the time required for the evaluation of logic programs in subsequent and similar evaluation shots;

3. stop and restart the computation of the answer sets when new facts are fed in input.

Concerning points 1 and 2 we focused on the usage of the pure ASP semantics in a repeated evaluation setting. In particular, we aimed at closing some of the mentioned gaps in the current state-of-the-art by investigating the possibility of generating incrementally larger ground programs for a fixed non-ground logic program. In our recent experiments, we showed that this approach, which we called "overgrounding", pays off in terms of performance. Overgrounded programs can be reused in combination with deliberately many different sets of inputs; this gives less control on the computational procedure easing the burden of taking care of it: indeed, the overgrounding process and the multi-shot machinery can be completely transparent to the user. We expect this setting to be particularly favourable when non-ground input knowledge bases are constituted of small set of facts, typical of declaratively programmed video game agents, or robots. Part of this work has been published in [18] while its extension will be published in [9].

5   Preliminary results

5.1   First year

In order to achieve our goal, we started to investigate the literature [1,3,4,7,14,15,17,19,21,22]. Then, we focused our research on artificial agents for video-games (also called "bots"). We worked on embedding rule-based *Reasoning Modules* into the well-known Unity[1] game development engine. To this end, we presented an extension of EMBASP[2], a framework to ease the integration of declarative formalisms with generic applications. Finally, we proved the viability of our approach[3] by developing a proof-of-concept Unity game that makes use of ASP-based AI modules [8].

---

[1] https://unity3d.com/unity

[2] https://www.mat.unical.it/calimeri/projects/embasp

[3] All the development material, including logic programs, source code and a fully playable version of the game are available at
https://github.com/DeMaCS-UNICAL/Pac-Man-Unity-EmbASP.

As a side research topic we investigated to what extent the ASP-based approach is scalable enough for industrial contexts in the field of video games, by proposing a Unity extension capable to automatically generate dungeons maps[4]. In this context we first investigated over the usage of a partition-based generation technique [25], then we proposed a multiple step-generation approach, set in the context of the 2-D caves generation domain, where each step is declaratively controlled by an ASP specification. With respect to existing literature [23, 26], our approach promises to be better scalable to real contexts with higher size mazes; experiments aimed at confirming that are currently ongoing. Finally we developed two plugins based on our generation technique, which were respectively deployed as an asset available in the Unity development and in the GVGAI [24] frameworks [8].

5.2 Second year

During the currently ongoing second year of studies, we continued to work on the lines of research described above. In particular we extended the integrated rule-based *Reasoning Module* framework by adding some new functionalities in order to ease the integration of declarative formalisms within the typical game development workflow in Unity. We solved some of the issues encountered in the previous work as, for example, the possibility to execute asynchronous tasks in a Unity game and the possibility to share the internal Unity data structure with the AI module. Thanks to these improvements we give the possibility to developers to directly attach a reasoning task to a trigger condition on an object property or at scheduled times.

After some preliminary studies we focused our attention on the usage of the pure ASP semantics in a repeated evaluation setting. We characterized a class of ground programs called *embeddings* or *embedding programs* which introduce a model-theoretic-like notion of ground programs; second, we proposed an incremental grounding strategy able to reuse previously grounded programs. This technique allows to knock down the time necessary for performing the instantiation of logic programs in subsequent, similar, evaluation shots. In particular, we maintain a stored ground logic program which grows monotonically from one shot to another; such *overgrounded programs* are series of embedding programs that become more and more general while elaborating consecutive shots, increasingly adding potentially useful rules. Cached rules can be reused in combination with deliberately many different sets of inputs. Then we developed a first implementation of an intelligent grounder able to apply the new introduced differential algorithm and finally tested our architecture over some applicative domains. We obtained very promising results.

Currently we are performing other tests using the benchmark suite used during the ASP Competition. Part of this work has been published in [18] while its extension will likely be published in [9].

6 Open issues and expected achievements

In the future work we aim at solving some of the issues encountered during the last 2 years. Some of them are reported in the following:

1. consider simplified ground programs;

2. introduction of memory constraints during the grounding step;

---

[4]Both versions of our prototypes, together with logic program specifications and source code are fully available online at
`https://github.com/DeMaCS-UNICAL/DCS-Maze_Generator-GVGAI` and
`https://github.com/DeMaCS-UNICAL/DCS-Maze_Generator-Unity`

3. development of an incremental solver considering also the model generation phase;

4. possibility to stop and restart the computation of an answer set.

Studying the first issue is of great importance because including simplifications in ground programs would knock down the usage of computational resources in terms of memory consumption; indeed, storing a non simplified ground program is more expensive than storing a simplified yet general ground program due to its larger and increasing size in terms of number of stored rules. Our next step is to add the simplification step in our incremental grounder framework.

After the implementation of the simplification, another issue we will deal with is the possibility to add a memory constraint during the grounding stage. Thanks to this possible improvement, one could set an upper bound to memory consumption. One possibility could be to remove the less triggered rules (or simply the oldest ones) when the memory limit has been exceeded.

Concerning the third issue, this is one of the crucial points that we need to solve in order to release a monolithic system able to directly produce incremental answer sets. At this stage of research activities, we are performing benchmarks also on the solving time of the overgrounded programs that are obviously greater than the solving time of the simplified instances. This time could be knocked down if a solver could be fed at each shot with only the newly computed ground rules. In this case it is not necessary to read at each iteration the entire ground program but only the new incremental rules.

Finally, we aim at adding the possibility to stop and restart the computation of the answer sets when the system understands that the current knowledge base has been changed by adding or removing assertions, so the set of candidate solutions is no more the same and it is needed to restart the computation over the new set of facts.

7   Conclusions

In this work we are investigating the field of stream reasoning with specific focus on achieving high performance. Our goal is to extend the currently developed framework to ease the generation of responsive, elastic and resilient artificial agents. Here we reported on how we are proceeding in order to make this goal possible and so we have presented our current state of the work. In the last year we worked on the implementation of an incremental grounder able to apply a differential algorithm in subsequent and similar shots over different input data; currently we are working on an improvement of the incremental grounder in order to take in consideration simplifications rules that can allow to produce smaller programs. Moreover, we have presented some of the research themes that we would like to deepen (add a memory consumption limit, develop of a full incremental solver, introduce the possibility to restart a computation at runtime).

References

[1] Selen Basol, Ozan Erdem, Michael Fink & Giovambattista Ianni (2010): *HEX Programs with Action Atoms*, pp. 24–33. doi:10.4230/LIPIcs.ICLP.2010.24.

[2] Nick Bassiliades, Antonis Bikakis, Stefania Costantini, Enrico Franconi, Adrian Giurca, Roman Kontchakov, Theodore Patkos, Fariba Sadri & William Van Woensel, editors (2017): *Proceedings of the Doctoral Consortium @ RuleML+RR 2017*. CEUR Workshop Proceedings 1875, CEUR-WS.org.

[3] Harald Beck, Minh Dao-Tran & Thomas Eiter (2018): *LARS: A Logic-based framework for Analytic Reasoning over Streams*. Artificial Intelligence 261, pp. 16–70, doi:10.1016/j.artint.2018.04.003.

[4] Harald Beck, Thomas Eiter & Christian Folie (2017): *Ticker: A system for incremental ASP-based stream reasoning*. TPLP 17(5-6), pp. 744–763, doi:10.1017/S1471068417000370.

[5] Rafael H. Bordini, Jomi Fred Hübner & Michael Wooldridge (2007): *Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology)*. doi:10.1002/9780470061848.

[6] Gerhard Brewka, Stefan Ellmauthaler, Ricardo Gonçalves, Matthias Knorr, João Leite & Jörg Pührer (2018): *Reactive multi-context systems: Heterogeneous reasoning in dynamic environments*. Artif. Intell. 256, pp. 68–104, doi:10.1016/j.artint.2017.11.007.

[7] Francesco Calimeri, Michael Fink, Stefano Germano, Giovambattista Ianni, Christoph Redl & Anton Wimmer (2013): *AngryHEX: an Artificial Player for Angry Birds Based on Declarative Knowledge Bases*. In: *Proceedings of the Workshop Popularize Artificial Intelligence co-located with the 13th Conference of the Italian Association for Artificial Intelligence (AI*IA 2013), Turin, Italy, December 5, 2013.*, pp. 29–35. Available at http://ceur-ws.org/Vol-1107/paper10.pdf.

[8] Francesco Calimeri, Stefano Germano, Giovambattista Ianni, Francesco Pacenza, Armando Pezzimenti & Andrea Tucci (2018): *Answer Set Programming for Declarative Content Specification: A Scalable Partitioning-Based Approach*. In: *AI*IA 2018 - Advances in Artificial Intelligence - XVIIth International Conference of the Italian Association for Artificial Intelligence, Trento, Italy, November 20-23, 2018, Proceedings*, pp. 225–237, doi:10.1007/978-3-030-03840-3_17.

[9] Francesco Calimeri, G Ianni, Francesco Pacenza, Simona Perri & Jessica Zangari: *Incremental answer set programming with overgrounding*.

[10] Stefania Costantini & Arianna Tocchio (2004): *The DALI Logic Programming Agent-Oriented Language*. In: *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30, 2004, Proceedings*, pp. 685–688, doi:10.1007/978-3-540-30227-8_57.

[11] Akshat Dhaon & Rem W. Collier (2014): *Multiple Inheritance in AgentSpeak(L)-Style Programming Languages*. In: *Proceedings of the 4th International Workshop on Programming Based on Actors Agents & Decentralized Control*, AGERE! '14, ACM, New York, NY, USA, pp. 109–120, doi:10.1145/2687357.2687362.

[12] Conrad Drescher, Stephan Schiffel & Michael Thielscher (2009): *A Declarative Agent Programming Language Based on Action Theories*. In: *Frontiers of Combining Systems, 7th International Symposium, FroCoS 2009, Trento, Italy, September 16-18, 2009. Proceedings*, pp. 230–245, doi:10.1007/978-3-642-04222-5_14.

[13] Thomas Eiter, V.S. Subrahmanian & T.J. Rogers (2000): *Heterogeneous active agents, III: Polynomially implementable agents*. Artificial Intelligence 117(1), pp. 107 – 167, doi:10.1016/S0004-3702(99)00104-6. Available at http://www.sciencedirect.com/science/article/pii/S0004370299001046.

[14] Michael Fink, Stefano Germano, Giovambattista Ianni, Christoph Redl & Peter Schüller (2013): *ActHEX: Implementing HEX Programs with Action Atoms*, pp. 317–322. doi:10.1007/978-3-642-40564-8_31.

[15] Martin Gebser, Torsten Grote, Roland Kaminski & Torsten Schaub (2011): *Reactive Answer Set Programming*. In: *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning*, LPNMR'11, Springer-Verlag, Berlin, Heidelberg, pp. 54–66, doi:10.1016/S0004-3702(02)00187-X. Available at http://dl.acm.org/citation.cfm?id=2010192.2010201.

[16] Martin Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2019): *Multi-shot ASP solving with clingo*. TPLP 19(1), pp. 27–82, doi:10.1017/S1471068418000054.

[17] Jason S Hartford, James R Wright & Kevin Leyton-Brown (2016): *Deep Learning for Predicting Human Strategic Behavior*. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon & R. Garnett, editors: *Advances in Neural Information Processing Systems 29*, Curran Associates, Inc., pp. 2424–2432. Available at http://papers.nips.cc/paper/6509-deep-learning-for-predicting-human-strategic-behavior.pdf.

[18] Giovambattista Ianni, Francesco Pacenza & Jessica Zangari (2019): *An Infrastructure for Multi-Shot Reasoning with Incremental Grounding*. In: *Proceedings of the 34th Italian Conference on Computational Logic*, CILC2019.

[19] Niels Justesen, Philip Bontrager, Julian Togelius & Sebastian Risi (2017): *Deep Learning for Video Game Playing*. *CoRR* abs/1708.07902. Available at `http://arxiv.org/abs/1708.07902`.

[20] Benjamin Kaufmann, Nicola Leone, Simona Perri & Torsten Schaub (2016): *Grounding and Solving in Answer Set Programming*. *AI Magazine* 37(3), pp. 25–32, doi:10.1609/aimag.v37i3.2672. Available at `http://www.aaai.org/ojs/index.php/aimagazine/article/view/2672`.

[21] Sven Koenig & Maxim Likhachev (2002): *Improved Fast Replanning for Robot Navigation in Unknown Terrain*. In: *Proceedings of the 2002 IEEE International Conference on Robotics and Automation, ICRA 2002, May 11-15, 2002, Washington, DC, USA*, pp. 968–975, doi:10.1109/`ROBOT`.2002.1013481.

[22] Marco Maratea, Luca Pulina & Francesco Ricca (2013): *Automated Selection of Grounding Algorithm in Answer Set Programming*. In: *Proceeding of the XIIIth International Conference on AI*IA 2013: Advances in Artificial Intelligence - Volume 8249*, Springer-Verlag New York, Inc., New York, NY, USA, pp. 73–84, doi:10.1007/978-3-319-03524-6_7.

[23] Mark J. Nelson & Adam M. Smith (2016): *ASP with Applications to Mazes and Levels*, pp. 143–157. Springer International Publishing, Cham, doi:10.1007/978-3-319-42716-4_8.

[24] Diego Pérez-Liébana, Jialin Liu, Ahmed Khalifa, Raluca D. Gaina, Julian Togelius & Simon M. Lucas (2018): *General Video Game AI: a Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms*. *CoRR* abs/1802.10363. Available at `http://arxiv.org/abs/1802.10363`.

[25] Noor Shaker, Antonios Liapis, Julian Togelius, Ricardo Lopes & Rafael Bidarra (2016): *Constructive generation methods for dungeons and levels*, pp. 31–55. Springer International Publishing, Cham, doi:10.1007/978-3-319-42716-4_3.

[26] A. M. Smith & M. Mateas (2011): *Answer Set Programming for Procedural Content Generation: A Design Space Approach*. *IEEE Transactions on Computational Intelligence and AI in Games* 3(3), pp. 187–200, doi:10.1109/TCIAIG.2011.2158545.