Explaining Answer-Set Programs with Abstract Constraint Atoms (Extended Abstract)

Thomas Eiter and Tobias Geibinger

Knowledge-based Systems Group, Institute of Logic and Computation, TU Wien,
Favoritenstraße 9-11, 1040 Vienna, Austria
{thomas.eiter, tobias.geibinger}@tuwien.ac.at

Abstract. Answer-Set Programming (ASP) is a popular declarative reasoning and problem solving formalism. Due to the increasing interest in explainability, several explanation approaches have been developed for ASP. However, support for commonly used advanced language features of ASP, as for example aggregates or choice rules, is still mostly lacking. We deal with explaining ASP programs containing Abstract Constraint Atoms, which encompass the above features and others. We provide justifications for the presence, or absence, of an atom in a given answer-set. To this end, we introduce several formal notions of justification in this setting based on the one hand on a semantic characterisation utilising minimal partial models, and on the other hand on a more ruled-guided approach. We provide complexity results for checking and computing such justifications, and discuss how the semantic and syntactic approaches relate and can be jointly used to offer more insight. Our results contribute to a basis for explaining commonly used language features and thus increase accessibility and usability of ASP as an AI tool.

1 Motivation

The growing pervasiveness of artificial intelligence (AI) in everyday life has led to concerns about the transparency of AI systems, and regulations against using "black-box" systems for sensitive tasks are under development.

Answer-Set Programming (ASP) is a symbolic, rule-based reasoning formalism that has been employed for various AI applications in numerous domains [6, 9], among them life sciences [7], health insurance [2], or psychology [12].

ASP allows for a declarative encoding of problems in a succinct manner. Solutions for them are obtained from *answer-sets*, which result from the evaluation of the encoding using an ASP solver. While ASP is a declarative AI approach, there is still need for providing concise and interpretable explanations as to why certain facts are, or are not, in a computed answer-set. For this reason, a number of explanation approaches for ASP have been developed; we refer to [10] for a comprehensive survey. However, most of the approaches in the literature do not support language extensions like aggregates [8] or choice rules [11]. As both features are frequently used in practice, the applicability of explanation approaches is limited and ASP can not live up to its full potential of a transparent AI tool.

This extended abstract encompasses work from a recent conference publication [5] tackling this shortcoming by introducing several formal notions of justification based on programs with *Abstract Constraint Atoms (c-atoms)* [13].

2 Background

We consider propositional *Answer-Set Programming* (ASP), assuming a denumberable set $\mathcal{A}$ of propositional *atoms*.

In particular, we consider programs consisting of *Abstract Constraint Atoms* (c-atoms) [13], which are defined as follows. A c-atom is a tuple $A = \langle D, C\rangle$, where $D \subseteq \mathcal{A}$ is the *domain* and $C \subseteq 2^D$ are the *satisfiers* of the c-atom.

The notion of c-atoms effectively generalises propositional atoms: any propositional atom a can be expressed by the c-atom $\langle\{a\}, \{\{a\}\}\rangle$. We call the latter *elementary* and whenever convenient, we will identify it with a.

We define the *complement* of a c-atom $A = \langle D, C\rangle$ as $\overline{A} = \langle D, \overline{C}\rangle$ where $\overline{C} = 2^D \setminus C$ are the *non-satisfiers* of A. Clearly, $A = \overline{\overline{A}}$ holds.

Following [14], we define *(disjunctive) logic programs* over c-atoms, which consist of a finite number of *rules* of the form

$$A_1 \vee \cdots \vee A_l \leftarrow A_{l+1}, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n,$$

where $A_1, \ldots, A_n$ are c-atoms, $l \geq 1$ and $m, n \geq 2$.

For a rule r of the form above, $H(r) := \{A_0, \ldots, A_l\}$ is the *head* of the rule, whereas $B(r) := \{A_{l+1}, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n\}$ is the *body*. The set of all propositional atoms appearing in program P is denoted by $\mathcal{A}_P$.

The semantics of logic programs is based on *interpretations*. Here, we also introduce them as potentially partial. A *partial interpretation* is a tuple $I = \langle I^+, I^-\rangle$, where $I^+, I^- \subseteq \mathcal{A}$ and $I^+ \cap I^- = \emptyset$. A partial interpretation $I = \langle I^+, I^-\rangle$ is *total on* set S if $I^+ \cup I^- \supseteq S$. When it is clear from context, we drop the set S and by default, we assume interpretations are total and identify them by the single set I^+. Furthermore, we say that I is finite if I^+ is.

A c-atom $A = \langle D, C\rangle$ is *satisfied* by a partial interpretation $I = \langle I^+, I^-\rangle$, denoted $I \models A$, if $S \in C$, where $S = I^+ \cap D$ and for each $U \in \overline{C}$ s.t. $S \subset U$, it holds that $U \cap I^- \neq \emptyset$.

The partial interpretation I satisfies $\mathit{not}\ A$ ($I \models \mathit{not}\ A$) if for each $S \in C$, either (a) $S \cap I^- \neq \emptyset$ or (b) $(U \setminus S) \cap I^+ \neq \emptyset$ for $U = \bigcup\{X \in \overline{C} \mid S \subseteq X\}$.

Intuitively, a c-atom is satisfied by an interpretation I if one of its satisfiers, which cannot be extended to an unsatisfier, is known to be true. The intuition behind satisfaction of a negative c-literal is more involved. However, we also give the following result.

Proposition 1. *Given a c-atom A, then for every partial interpretation I we have $I \models \mathit{not}\ A$ iff $I \models \overline{A}$.*

Satisfaction is generalised to sets of c-literals and rules as usual and answer-sets are defined as follows. Let P be a program and I be a total interpretation.

Then

$$P^I := \{HR(H(r), I) \leftarrow B(r) \mid r \in P, I \models B(r)\},$$

where $HR(M, I) := \{\langle D^A, \{I \cap D^A\}\rangle \mid A \in M, I \models A\}$ if $I \models H(r)$ and $HR(M, I) := \bot$ otherwise, is the *extended FLP reduct* of P with respect to I. A finite total interpretation I is then an *answer-set* of P if $I \models P^I$ and there is no total interpretation $I' \subset I$ such that $I' \models P^I$. The set of all answer-sets of P is denoted by $AS(P)$.

3 Contributions

The first contribution of the paper is a notion which we call *model-based justification* or *m-justification* for short. First, we define the following order over partial interpretations.

Definition 1. *Let J_1 and J_2 be partial interpretations. Then $J_1 \leq J_2$ if $J_1^+ \subseteq J_2^+$ and $J_1^- \subseteq J_2^-$. Furthermore, $J_1 < J_2$ if $J_1 \leq J_2$ and $J_1 \neq J_2$.*

An m-justification is then defined as follows.

Definition 2. *Let L be a c-literal and I be a total model of L. Then, a partial interpretation J is called a* (positive) m-justification *of $I \models L$ if (i) $J \leq I$, (ii) $J \models L$ and (iii) there is no $J' < J$ such that $J' \models L$.*

The intuition here is that an m-justification should highlight the parts of the model that are responsible for the satisfaction of the c-literal. The definition can be readily extended to non-satisfaction.

Definition 3. *Let L be a c-literal and I be a total countermodel of L. Then J is a* (negative) m-justification *of $I \not\models L$ if J is an m-justification of $I \models \overline{L}$.*

Let us look at an example.

Example 1. Consider the c-atom $A_1 = \langle D_1, C_1\rangle$ where $D_1 = \{a, b, c\}$ and $C_1 = \{\{a\}, \{b, c\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}$, which represents the aggregate $\#\texttt{sum}\{\texttt{2} : \texttt{a}, \texttt{1} : \texttt{b}, \texttt{1} : \texttt{c}\} > \texttt{1}$. Suppose we have $I_1 = \{a, b, c\}$ and $S_1 = I \cap D_1$. Clearly, $S_1 \in C_1$ and thus $I_1 \models A_1$. Furthermore, $\langle\{a\}, \emptyset\rangle$ and $\langle\{b, c\}, \emptyset\rangle$ are valid m-justifications. The former can be read as follows: the c-atom A_1 is satisfied by I because a is true. Consider $I_2 = \{b\}$. Since $I_2 \not\models A_1$, we may look for m-justifications of $\overline{A} = \langle D_1, \{\emptyset, \{b\}, \{c\}\}\rangle$. The partial interpretation $\langle\emptyset, \{a, c\}\rangle$ is the only m-justification. The intuition here is that $I_2 \not\models A_1$ holds because neither a nor c are satisfied by I_2, which, since b cannot be false in I_2, would be a requirement for $\overline{A}$ to be satisfied.

The above notions can naturally be extended towards sets of c-literals. The reader is referred to the paper [5] for details.

An issue with m-justification is that they do not take the rules of the program into account. This motivates our second contribution. Namely, *rule-based justifications (r-justifications)*. First, we give some auxiliary definitions.

Definition 4 (Presumptuous Entailment). *Let P be a program, J be a partial interpretation and A be a c-atom. Then we define $P \models_J A$ if for every total model I of P s.t. $I \geq J$, it holds that $I \models A$.*

Definition 5 (Failed Support). *Let r be a rule and I be a total model of r. Then, r is a* failed support *for atom a w.r.t. I if $I \not\models B(r)$ and for every I', $I' \models a$ if $I' \models r$ and $I' \models B(r)$.*

Now we can define r-justifications.

Definition 6. *Let P be a program, $I \in AS(P)$ be an answer-set and a be an atom. Then, a triple (a°, Q, J), where $\circ \in \{+, -\}$, $Q \subseteq P$ is a set of rules, and $J \leq I$ is a partial interpretation, is an* r-justification *for a w.r.t. P and I if the following conditions hold:*

(a) If $a \in I$, then $\circ = +$, $Q^I \models_J a$ and there is no $R \subset Q$ such that $R^I \models_J a$.
(b) If $a \notin I$, then $\circ = -$, $Q = \{r \in P \mid r$ is a failed support of a w.r.t. $I\}$, and for every $r \in Q$, $J \models \overline{A}$ for some $A \in B(r)$.

We say that the r-justification is concise *if J is $\leq$-minimal.*

An r-justification is essentially composed of three things: (1) an annotated atom indicating what is justified, (2) the set of rules needed to do so, and (3) a partial interpretation explaining why the rules do, or respectively do not, yield the atom. Furthermore, both conditions (a) and (b) ensure that the set of rules Q is minimal and thus contains no redundant rules.

Example 2. Consider the program $P_1 = \{r_1 : d \leftarrow \langle\{a, b, c\}, \{\emptyset, \{a\}, \{b\}, \{a, b\}\}\rangle$, $r_2 : a \leftarrow not\ c,\ r_3 : c \leftarrow not\ a\ \}$ and one of its answer-sets $\{a, d\}$. Intuitively, the body of rule r_1 is true whenever c is false and the other encode a choice between a and c. The concise r-justifications for a, b, c and d are then $(a^+, \{r_2\}, \langle\emptyset, \{c\}\rangle)$, $(b^-, \emptyset, \langle\emptyset, \emptyset\rangle)$, $(c^-, \{r_3\}, \langle\{a\}, \emptyset\rangle)$ and $(d^+, \{r_1\}, \langle\emptyset, \{c\}\rangle)$.

In the paper, we additionally discuss chains of r-justifications, which exhaustively try to justify atoms, an their *elaboration*, i.e., minimising the rules contained in an element of the chain. We will not provide the formal definitions here, but give the following example.

Example 3. For the program $P_5 = \{r_{10} : a \leftarrow b,\ r_{11} : b \leftarrow not\ c,\ r_{12} : \bot \leftarrow c\}$, the set $\{a, b\}$ is the unique answer-set and $\mathcal{J} = (a^+, \{r_{10}, r_{11}\}, \langle\emptyset, \{c\}\rangle)$, $(c^-, \emptyset, \langle\emptyset, \emptyset\rangle)$ is an r-justification chain for a. Furthermore, $\mathcal{J}$ can be elaborated through the first element yielding $(a^+, \{r_{10}\}, \langle\{b\}, \emptyset\rangle)$, $(b^+, \{r_{11}\}, \langle\emptyset, \{c\}\rangle)$, $(c^-, \emptyset, \langle\emptyset, \emptyset\rangle)$ as a (concise) chain.

We also studied the computational complexity of recognising and computing model-based and rule based justifications. The complexity ranges for the recognition problems from $\mathsf{NP}/\mathsf{coNP}$ to Σ_2^p, while computing justifications is feasible in polynomial time with an NP oracle. For the detailed results, we again refer to the paper.

4 Conclusion & Future Work

We introduced and studied complementary notions of justification for ASP programs built over Abstract Constraint atoms, which encompass various ASP language extensions, among them aggregates and choice rules. Our m-justifications use a semantic approach based on minimal partial models, while r-justification chains are a more syntactically minded, rule-based notion. We have provided several examples and showed how the notions can be jointly used. Besides some basic properties of justifications, we provided complexity results for emerging computational tasks. While they are often intractable in general, we identified relevant tractable cases.

Related work includes the explanation systems `xclingo` [3, 4] and `xASP2` [1], which both have recently begun to support programs with aggregates. However, both approaches currently do not give minimal justifications for aggregates and thus incorporating our notion of m-justification in those systems might be an interesting topic.

Our ongoing and future work aims to extend this theoretical study of justification, and to utilise it in a fully-fledged and interactive explanatory system. Specifically, a user may opt in a session for the model- or the rule-based approach at the current stage to obtain more insight, control which rules get expanded, and drive possible elaboration of justifications. Investigating how our justifications can be used or extended to offer contrastive justifications, which inform about changes to flip the membership of atoms in an answer-set, is another worthwhile endeavour.

References

1. Alviano, M., Trieu, L.L.T., Son, T.C., Balduccini, M.: Explanations for answer set programming. In: Technical Communications of the 39th International Conference on Logic Programming (ICLP 2023) (2023). https://doi.org/https://dx.doi.org/10.4204/EPTCS.385.4
2. Beierle, C., Dusso, O., Kern-Isberner, G.: Using answer set programming for a decision support system. In: Proceedings of the 8th International Conference (LPNMR 2005). LNCS, vol. 3662, pp. 374–378. Springer (2005). https://doi.org/10.1007/11546207_30
3. Cabalar, P., Fandinno, J., Muñiz, B.: A system for explainable answer set programming. In: Technical Communications of the 36th International Conference on Logic Programming (ICLP 2020). EPTCS, vol. 325, pp. 124–136 (2020). https://doi.org/10.4204/EPTCS.325.19
4. Cabalar, P., Muñiz, B.: Explanation graphs for stable models of labelled logic programs. In: Proceedings of the International Conference on Logic Programming 2023 Workshops co-located with the 39th International Conference on Logic Programming (ICLP 2023). CEUR Workshop Proceedings, vol. 3437. CEUR-WS.org (2023), https://ceur-ws.org/Vol-3437/paper3ASPOCP.pdf
5. Eiter, T., Geibinger, T.: Explaining answer-set programs with abstract constraint atoms. In: Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI 2023). pp. 3193–3202. ijcai.org (2023). https://doi.org/10.24963/ijcai.2023/356

6. Erdem, E., Gelfond, M., Leone, N.: Applications of answer set programming. AI Magazine **37**(3), 53–68 (2016). https://doi.org/10.1609/aimag.v37i3.2678
7. Erdem, E., Oztok, U.: Generating explanations for biomedical queries. Theory and Practice of Logic Programming **15**(1), 35–78 (2015). https://doi.org/10.1017/S1471068413000598
8. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004). LNCS, vol. 3229, pp. 200–212. Springer (2004). https://doi.org/10.1007/978-3-540-30227-8_19
9. Falkner, A., Friedrich, G., Schekotihin, K., Taupe, R., Teppan, E.C.: Industrial applications of answer set programming. KI - Künstliche Intelligenz **32**(2), 165–176 (2018). https://doi.org/10.1007/s13218-018-0548-6
10. Fandinno, J., Schulz, C.: Answering the "why" in answer set programming - A survey of explanation approaches. Theory and Practice of Logic Programming **19**(2), 114–203 (2019). https://doi.org/10.1017/S1471068418000534
11. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers (2012)
12. Inclezan, D.: An application of answer set programming to the field of second language acquisition. Theory and Practice of Logic Programming **15**(1), 1–17 (2015). https://doi.org/10.1017/S1471068413000653
13. Marek, V.W., Truszczynski, M.: Logic programs with abstract constraint atoms. In: Proceedings of the 19th National Conference on Artificial Intelligence (AAAI 2004). pp. 86–91. AAAI Press / The MIT Press (2004)
14. Oetsch, J., Pührer, J., Tompits, H.: Stepwise debugging of answer-set programs. Theory and Practice of Logic Programming **18**(1), 30–80 (2018). https://doi.org/10.1017/S1471068417000217