Explainable ASP

Jérémie Dauphin[1]* and Ken Satoh[2]**

[1] CSC, University of Luxembourg, Esch-sur-Alzette, Luxembourg
jeremie.dauphin@uni.lu
[2] National Institute of Informatics, Tokyo, Japan
ksatoh@nii.ac.jp

Abstract. Despite its proven relevance, ASP (answer set programming) suffers from a lack of transparency in its outputs. Much like other popular artificial intelligence systems such as deep learning, the results do not come with any explanation to support their derivation. In this paper, we use a given answer set as guidance for a simplified top-down procedure of answer set semantics developed by Satoh and Iwayama to provide not only an explanation for the derivation (or non-derivation) of the atoms, but also an explanation for the consistency of the whole answer set itself. Additionally, we show that a full use of the Satoh-Iwayama procedure gives an explanation of why an atom is not present in any answer set.

1 Introduction

The ASP reasoning system has proved useful in many situations [Erdem2016]. However, much like AI paradigms such as deep learning, it fails to provide explanations for any of its outputs. In critical domains such as self-driving vehicles and legal reasoning, providing such an explanation to justify the results of systems of increasing complexity is crucial for well-informed decision-making and future improvements of the systems.

In this paper, we treat a proof sequence of top-down proof of the Satoh-Iwayama procedure [Satoh93] as such an explanation. We additionally simplify the procedure by considering a given answer set as input to guide the procedure. This mechanism is based on the notion of "well-supportedness" [Fages1991], which identifies a sequence of rules in derivation of an atom in the answer set.

Additionally, the procedure also produces an explanation not only for the presence of a certain literal in a given answer set, but also for the consistency of said answer set (we call this a "credulous explanation"). Indeed, while providing justifications for how a literal is being derived in a given answer is important, it is also crucial to show how the given answer set avoids deriving inconsistencies and thus satisfies integrity constraints. For debugging purposes, issues might sometimes arise in this part of the logic program, rather than in the derivation of a literal of interest.

* The work of Jérémie Dauphin was supported by the H2020 Marie Skłodowska-Curie grant number 690974 for the project MIREL.

** This work was partially supported by JSPS KAKENHI Grant Number 17H06103.

Consider the following example logic program: $\bot \leftarrow q$
$p \leftarrow \sim q$
$q \leftarrow \sim p$
Suppose that $\bot \leftarrow q$ has been added by mistake instead of $\bot \leftarrow p$. Then, we have an extension $\{p\}$. We would like to know why p is derived but without considering the integrity constraint of $\bot \leftarrow p$, we cannot detect the mistake.

2 Preliminaries

In this section, we briefly present some definitions from the state of the art. The basic building blocks are *atoms*, elementary propositions which may be true or false. A *literal* is either an atom a or its weak negation $\sim a$. Weak negation means that $\sim a$ holds iff there is no derivation of a.

Definition 1. *Let l be a literal. We denote the* inverse *of the literal as $\tilde{l}$ where*

- *if l is a positive literal then $\tilde{l} = \sim l$ (negation of l);*
- *if l is a negative literal of the form $\sim l'$, then $\tilde{l} = l'$.*

Definition 2. *A* rule *R is an expression of the form:*

$$H \leftarrow B_1, B_2, ..., B_k, \sim A_1, ..., \sim A_m$$

Where B_i and A_j are all atoms, and H is either an atom or $\bot$ (meaning contradiction*). We call H the* head *of the rule denoted as $head(R)$ and a set of literals, $B_1, B_2, ..., B_k, \sim A_1, ..., \sim A_m$: the* body *of the rule denoted as $body(R)$. We also denote a set of positive literals in the rule, $B_1, B_2, ..., B_k$ as $pos(R)$ and a set of literals appearing negatively in the rule, $A_1, ..., A_m$ as $neg(R)$.*

If $H = \bot$, we call the rule an integrity constraint*:*

$$\bot \leftarrow B_1, B_2, ..., B_k, \sim A_1, ..., \sim A_m.$$

A *logic program T* is a set of such rules and integrity constraints.

Definition 3. *Let I be a set of atoms. We say that I satisfies the body of a rule R (denoted as $I \models body(R)$) iff $pos(R) \subseteq I$ and $neg(R) \cap I = \emptyset$.*

Definition 4. *Given a logic program T, a* model *of T is a set of atoms M s.t. $p \in M$ iff $\exists r \in T$ s.t. $head(R) = p$ and $M \models body(R)$.*
If a model of T is minimal in the set inclusion sense, we call it a minimal model*.*
If a minimal model is unique, we call it the least model*.*

Definition 5. *Given a logic program T, an* answer set *M is a set of atoms s.t. $\bot \notin M$ and*

$$M = min(T^M)$$

where

- $T^M = \{head(R) \leftarrow pos(R) | R \in T$ *and* $neg(R) \cap M = \emptyset\}$*, which we also call the* reduct *of* T *w.r.t.* M*;*
- $min(T)$*: the least model of* T*.*

So the answer sets are the models which are exactly the least models of the program reduct in their respect. Note that since the reduct is a positive logic program, it is guaranteed to have a least model.

We now introduce the notion of *resolution*.

Definition 6. *Given a logic program* T *and a literal* l*, the* resolution *of* T *w.r.t.* l *is:*

- *If* l *is a positive literal:*
  $resolve(l, T) = \{H \leftarrow B_1, ..., B_{i-1}, B_{i+1} ..., B_k, \sim A_1, ..., \sim A_m | H \leftarrow B_1, ..., B_k, \sim A_1, ..., \sim A_m \in T$ *and* $l = B_i\}$
- *If* l *is a negative literal of the form* $\sim l'$*:*
  $resolve(l, T) = \{H \leftarrow B_1, ..., B_k, \sim A_1, ..., \sim A_{i-1}, A_{i+1}, ..., \sim A_m | H \leftarrow B_1, ..., B_k, \sim A_1, ..., \sim A_m \in T$ *and* $l' = A_i\} \cup \{\bot \leftarrow B_1, ..., B_k, \sim A_1, ..., \sim A_m | H \leftarrow B_1, ..., B_k, \sim A_1, ..., \sim A_m \in T$ *and* $l' = H\}$

In the cases where l is a positive literal, the resolution is the set of rules containing l in the positive part of the body, but with l removed from the body. In the cases where l is a negative literal, the resolution is the set of rules containing l either in the negative part of the body of the head, but with it again removed. The aim is to identify the rules affected by the fixing of l's truth value and consider their simplified version, where l's truth value has been taken into account.

3 Overview of the Full Version of the Satoh-Iwayama Procedure

We reproduce the Satoh-Iwayama top-down procedure [Satoh93] as a basis for producing explanations. The procedure has the following characteristics:

- the procedure answers whether there is an answer set which satisfies a query.
- the procedure is correct for a consistent logic program.
- for a finite and consistent logic program, if there is an answer set which satisfies a query, the procedure always answers "yes".

It consists of the subprocedures *derive*, *literal_con*, *rule_con* and *delete_con*. Note that Δ in the procedure expresses the literals derived by the recursive subprocedure calls. So Δ is used in order to ensure that literals are only computed once, thus avoiding redundant computations.

For *derive*(p, Δ), given an atom p and a set of literals Δ, we do the following:

1. Check whether p has already been computed, in which case we can stop.

2. Otherwise, select a rule with p as its head:

$$p \leftarrow p_1, ..., p_m, \sim q_1, ..., \sim q_n$$

3. Check if every positive literal p_i in its body can be derived by calling *derive*.
4. Check if every negative literal $\sim q_j$ in its body can be consistently assumed to be false by calling *literal_con*.
5. Check that the head becoming true does not lead to contradiction by calling *literal_con*.

For *literal_con*$(l, \varDelta)$, given a literal l and a set of literals $\varDelta$, we do the following:

1. Check if l has already been computed, in which case we can stop.
2. Otherwise, add l to $\varDelta$.
3. Check the consistency of rules which are obtained by the resolution of l and the program by calling *rule_con*. This is used to check that l being true does not lead to any contradiction.
4. Check the consistency of deleted rules in the program by l by calling *deleted_con*. This is used to check consistency of implicit deletion of a rule by assuming l which might lead to contradiction.

For *rule_con*$(R, \varDelta)$, given a rule R and a set of literals $\varDelta$, we show one of the following:

Case 1 A positive literal in the body of R can be consistently assumed to be false by calling *literal_con*.
Case 2 A negative literal in the body of R is derived by calling *derive*.
Case 3 Every positive literal in the body of R is derived and every negative literal in the body of R can be consistently assumed to be false and the assumption that the head becomes true does not lead to contradiction by calling *derive* and *literal_con*.

Cases 1 and 2 are for when the rule is not applicable, while case 3 ensure that in the cases where the rule is applicable, it does not lead to any contradiction.

For *deleted_con*$(R, \varDelta)$, given a rule R and a set of literals $\varDelta$, we show one of the following:

Case 1 The head of a deleted rule R is derived by another rule by calling *derive*.
Case 2 The head of a deleted rule R can be consistently assumed to be false by calling *literal_con*.

This subprocedure simply ensures that the rule R being no-longer applicable, due to the inverse of a literal in its body having been set to true, does not cause any issues.

For a detailed account of the procedure, we refer the reader to the original work [Satoh93].

4 Answer Set Guided Method of Producing Explanation why an Atom is Derived in a given Answer Set

In our work, we focus on causal explanations, and thus consider a causal trace to be an explanation. We base this on the following quote:

> To explain an event is to provide some information about its causal history. In an act of explaining, someone who is in possession of some information about the causal history of some event - *explanatory information*, I shall call it - tries to convey it to someone else. - Lewis [Lewis1986]

We first give a definition of well-supportedness from [Fages1991]. Identifying supporting rules provides us with a first step towards explanation, and allows us to speed up the second part of the explanation process.

Definition 7. *Let T be a logic program. We say that a set of atoms I is* well-supported *if there exists a strict well-founded partial order $\prec$ on I such that for any atom $A \in I$ there exists a rule $R \in T$ called* a supporting rule *for A in T s.t. $head(R) = A$ and $I \models body(R)$ and for every $B \in pos(R)$, $B \prec A$.*

We call such a set of atoms I a *well-supported model.*

Theorem 1. *([Fages1991]) Let T be a logic program. A set of atoms I is an answer set iff I is a well-supported model.*

Since we consider only positive literals in the body of a supporting rule in the definition of a well-supported model M for a logic program T, we can instead look at the corresponding rules from T's reduct with respect to M, and then return their original equivalents. Given an answer set, we can compute the set of corresponding supporting rules in linear time by slightly adapting the algorithm 2 of [Dowling1984]. By first taking the reduct according to the given answer set, we are able to apply their algorithm for checking the satisfiability of positive logic programs. The algorithm works by marking edges as they are visited, and we can re-use this marking to identify the supporting rules.

Now, we give an answer set guided method of producing an explanation of why an atom is derived in a given answer set. We can use the knowledge of having the answer set known in advance in order to skip some selection operations in the original procedure. This allows us to speed up the process and save a considerable amount of computations.

It consists of simplified versions of the above four subprocedures, with the main difference being that we now have a global variable M representing the answer set of interest. This allows many branching choices to be reduced to single paths in the proof tree, saving many unnecessary computations. We briefly sketch the improvements from the original procedure.

For *simple_derive*(p, Δ), we can now select a supporting rule with respect to M instead of trying all rules with p as their heads. Since it is a supporting rule, we are guaranteed to be able to apply it, and hence won't need to backtrack and restart the process. This reduce a potential exponential branching to a single operation guaranteed to succeed.

The subprocedure *literal_con*$(l, \varDelta)$ remains identical to the original one, as in this case the knowledge on the answer set does not provide any help.

For *simple_rule_con*$(R, \varDelta)$, we can guide the procedure towards the relevant case from the three possibilities, since we know from the answer set which of the three cases is the applicable one.

The case of *simple_deleted_con*$(R, \varDelta)$ is similar to the previous one, in the sense that the answer set tells us which of the two cases to pursue.

5 Examples of Explanations

We show examples of explanations. For a more user-friendly interface, we translate the subprocedure calls into natural language sentences.

Let T be: $\{p \leftarrow {\sim}q; q \leftarrow {\sim}p; r \leftarrow q; r \leftarrow {\sim}r\}$.

We show a credulous explanation for the derivation of q in the answer set $\{q, r\}$. To justify q, we first check which rule should be applied and since $q \leftarrow {\sim}p$ is a supporting rule for q given the answer set M, we choose the rule without making a selection of other rules[3]. Then, to derive q from the rule $q \leftarrow {\sim}p$, we first check the consistency of ${\sim}p$. To do this consistency check, we check the consistency of the resolvents of ${\sim}p$ w.r.t. T, which are $0 \leftarrow {\sim}q$[4] and $q \leftarrow$.

1. To show the consistency of $0 \leftarrow {\sim}q$, we show a derivation of q[5]. Then,
   (a) we can identify the supporting rule, $q \leftarrow {\sim}p$, for q given M and show the consistency of ${\sim}p$. This is done since ${\sim}p$ is already assumed
   (b) we check the consistency of assuming q as follows:
      i. we check the consistency of resolvent of q w.r.t. T, which is $r \leftarrow$. Then, we check the consistency of assuming r. This is done by consistency check of deleting $r \leftarrow {\sim}r$. This is successful since r is already assumed.
      ii. we check the consistency of deleted rule, $p \leftarrow {\sim}q$. In this case, thanks to M, we can determine that p should be false and it is proved since p is already assumed to be false.
2. the consistency of $q \leftarrow$ is proved since q is already assumed.

And finally, we check the consistency of assuming q, which is immediate since q is already assumed.

[3] In this example, $q \leftarrow {\sim}p$ is the only rule for deriving q but even if there were other rules, we can identify this rule given M

[4] We display 0 instead of $\perp$ for notational consistency with the meta-interpreter output of the Satoh-Iwayama procedure.

[5] Since we know that $q \in M$, we can speed up the process by skipping the other checks.

We now show a cautious explanation for why there is no answer set including p for a logic program T. To show this, we need to show that every derivation of p leads to a contradiction. Since there is only one rule to derive p, that is $p \leftarrow \sim q$, we show consistency of assuming $\sim q$ as follows:

1. we need to check for consistency of the resolvents for $\sim q$ w.r.t. T, that is, $p \leftarrow$ and $0 \leftarrow \sim p$, as follows:
   (a) for $p \leftarrow$, deriving p does not lead to contradiction so p is assumed.
   (b) for $0 \leftarrow \sim p$, we already assume p so it is consistent.
2. we need to check consistency of a deleted rule by $\sim q$. w.r.t. T, that is, $r \leftarrow q$. To do so, we need to check whether r is either true or false. However, both checks fail in the following reasons:
   (a) in order to show r, we could use either $r \leftarrow q$ or $r \leftarrow \sim r$. But the first rule cannot be used since $\sim q$ is already assumed. For the second rule, we need to assume $\sim r$ to derive r. Then, we need to check resolvents of $\sim r$ w.r.t. T, that is $0 \leftarrow q$ and $0 \leftarrow \sim r$ and $r \leftarrow$:
      i. $0 \leftarrow q$ is consistent since $\sim q$ is already assumed.
      ii. $0 \leftarrow \sim r$ leads to contradiction since $\sim r$ is already assumed, therefore, showing r is failed.
   (b) in order to show $\sim r$, we should show that $\sim r$ is consistently assumed but then we iterate a part of the above checking to show r, the procedure is failed.
3. Therefore, r is neither true nor false and so derivations of $\sim$ and p are failed.

Thus, there is no answer set including p.

6 Related Work

In this section, we compare our approach with works mentioned in the excellent survey of explanations in ASP made by Fanndinno and Schulz [Fandinno2019].

[Pontelli2009] gives a method of producing a graph-based explanation (called off-line justification) of the truth value of an atom w.r.t. a given answer set and extends the method to give a justification of atoms during the computation of an answer set (called on-line justification). In off-line justification, all the rule application steps used to derive an atom is included in the graph. On the other hand, [Schulz2016] decomposes each derivation of an atom into a part whether only assumptions to derive an atom are considered. Then, they define attack tree justification to give an overall explanation. [Cabalar2014] gives a method of giving an explanation by causal graphs. They give an algebraic characterization of combining causal graphs.

These approaches represent a graph (or a tree) for an explanation about derivation of a literal whereas our approach not only gives a derivation of a literal but also gives a derivation for the consistency of assuming the literal. Moreover, our approach is more procedural in which we give a credulous explanation by proof sequence of why a literal is derived in a given answer set and maintain consistency. We also give a cautious explanation why a literal is not included in any answer set. In the cautious explanation, we could use the previous methods above by giving an explanation why an atom is not included in each answer set after computing all the answer sets. However, this would cause a complex explanation whereas our proof procedure could share a common derivation to simplify the explanation.

7 Conclusion

We show methods of giving credulous and cautious explanations in ASP. We modify the Satoh-Iwayama's top-down procedure for answer set to produce a credulous explanation using a guidance of a given answer set. We also show a full use of the Satoh-Iwayama procedure to produce a cautious explanation.

We would like to apply these methods for applying ASP in a critical domain and show usefulness of our approach as a future work. Additionally, we would like to consider Miller's work on what constitutes an explanation in AI [Miller2018] in order to produce multiple kinds of explanations.

References

[Cabalar2014] Cabalar, P., Fandinno, J., and Fink, M., "Causal Graph Justifications of Logic Programs", Theory and Practice of Logic Programming 14, (4-5), 603-618 (2014).

[Dowling1984] Dowling, W. F., and Gallier, J., "Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae", The Journal of Logic Programming 1(3), 267-284 (1984).

[Erdem2016] Erdem, E., Gelfond, M., and Leone, N., "Applications of Answer Set Programming", AI Magazine 37(3): 53-68 (2016).

[Fages1991] Fages, F., "A new fixpoint semantics for general logic programs compared with the well-founded and the stable model semantics", New Generation Computing, Volume 9, Issue 3–4, 425–443 (1991).

[Fandinno2019] Fandinno, J., and 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).

[Pontelli2009] Pontelli, E., Son, T. C., and El-Khatib, O., "Justifications for Logic Programs under Answer Set Semantics", Theory and Practice of Logic Programming 9 (1), 1-56 (2009).

[Satoh93] Satoh, K. and Iwayama, N., "A Correct Goal-Directed Proof Procedure for a General Logic Program with Integrity Constraints", E. Lamma and P. Mello (eds.), Extensions of Logic Programming, LNAI 660, pp. 24 – 44, Springer-Verlag (1993).

[Schulz2016] Schulz, C., and Toni, F., "Justifying Answer Sets using Argumentation", Theory and Practice of Logic Programming 16 (1), 59-110 (2016).

[Miller2018] Miller, Tim, "Explanation in artificial intelligence: Insights from the social sciences", Artificial Intelligence (2018).

[Lewis1986] Lewis, David, "Causal explanation", Philosophical Papers 2, 214-240 (1986).