# `xASP`: An Explanation Generation System for Answer Set Programming

Ly Ly Trieu[1](✉), Tran Cao Son[1], and Marcello Balduccini[2]

[1] New Mexico State University, New Mexico, USA
`lytrieu@nmsu.edu, tson@cs.nmsu.edu`
[2] Saint Joseph's University, Pennsylvania, USA
`mbalducc@sju.edu`

**Abstract.** In this paper, we present a system, called `xASP`, for generating explanations that explain *why* an atom belongs to (or does not belong to) an answer set of a given program. The system can generate all possible explanations for a query without the need to simplify the program before computing explanations, i.e., it works with non-ground programs. These properties distinguish `xASP` from existing systems such as `xClingo`, `DiscASP`, $exp(ASP^c)$, and *s(CASP)*, which also generate explanations for queries to logic programs under the answer set semantics but simplify and ground the programs (the three systems `xClingo`, `DiscASP`, $exp(ASP^c)$) or do not always generate all possible explanations (the system *s(CASP)*). In addition, the output of `xASP` is insensitive to syntactic variations such as the order conditions and the order of rules, which is also different from the output of *s(CASP)*.

**Keywords:** Explainable AI · Logic Programming · Answer Set Programming

## 1 Introduction

Recent interest in explainable artificial intelligence provided the impulse for the development of several systems capable of generating explanations for queries posed to a logic program under the answer set semantics such as `xClingo` [2], `DiscASP` [4], $exp(ASP^c)$ [7], and *s(CASP)* [1]. These systems can be characterized by three dimensions: (*i*) the strategy for computing the explanation (grounding vs. non-grounding), (*ii*) the types of queries that can be posed to the system (true atoms and false atoms), and (*iii*) the representation of the answers. Among these systems, only *s(CASP)* does not ground the program before computing the answers; both *s(CASP)* and $exp(ASP^c)$ generate explanations for atoms in an answer set (true atoms) and atoms not in an answer set (false atoms); while `xClingo` is not applicable to false atoms; and `DiscASP` currently only works for propositional answer set programs. *s(CASP)* generates a partial answer set supporting a query while others generate a full justification, represented by an explanation graph, given an answer set.

Partially supported by NSF grants 1914635, 1757207, 1812628. This contribution was also made possible in part through the support of NIST via cooperative agreement 70NANB21H167.

Grounding a program before computing an explanation comes at some costs. One of the most significant problems is that the grounding simplification techniques applied by answer set solvers tend to remove various pieces of information, resulting in explanations that are no longer faithful to the original program, or unable to even provide an explanation. This is illustrated in the next example.

*Example 1 (Limitation of Current Approaches).* Let $P$ be a program:

$$d :- b(X), a(X). \qquad b(1). \qquad a(4).$$

This program has a unique answer set $\{a(1), b(4)\}$. $d$ is false in this answer set. Suppose that we are interested in the question "*why is d false?*"

Among the four systems mentioned earlier, only *s(CASP)* is able to provide an explanation for this query. For other systems, no explanation for a false atom is provided, either by design or by the simplification process. *exp*(*ASP*$^c$) does not return an explanation graph for $d$ because $d$ is eliminated by the solver during the grounding phase.

In the above example, *s(CASP)* generates the following justification[1]

```
not d :- not b(Var0 | {Var0 \= 1}), b(1), not a(1).
```

This says that there is an answer set containing $b(1)$, that does not contain $a(1)$ and does not contain any other atom of the form $b(x)$ such that $x \neq 1$.

When we switch the position of $b(X)$ and $a(X)$ in the first rule, we receive a different justification:

```
not d :- not a(Var0 | {Var0 \= 4}),  a(4), not b(4).
```

The above example highlights the shortcomings of existing systems. For *s(CASP)*, even though both answers are correct, it is not ideal that a slight semantics-preserving change in the input results in a different justification.

In this work, we describe `xASP`, a system capable of computing the explanation graphs of a ground atom $a$ w.r.t. an answer set $A$ of a non-ground program $P$. By working directly with programs including variables, `xASP` generates explanation graphs that are faithful to the program, thus distinguishing itself from `xClingo`, `DiscASP`, and *exp*(*ASP*$^c$), which simplify the program before computing an explanation. Different from *s(CASP)*, it generates all full explanation graphs for an atom given an answer set and its behavior is not affected by semantics-preserving changes in the program. To work with programs including variables, `xASP` uses the given atom and answer set to dynamically identify relevant ground rules for the construction of the answers. Again, the main purpose of `xASP` is to help respond to the need for explainable AI. However, by presenting the applicable rules, facts, and assumptions used in the derivation of a given atom, `xASP` could be useful for debugging as well. For example, if an atom $a$ is supposed to be false in all answer sets of a program $P$ but appears in some answer set $A$, the explanation graph of $a$ could be useful in figuring out which rule must not be applicable, etc.

[1] $p(V \mid V \neq v)$ represents the set of all atoms of the form $p(x)$ except for the atom $p(v)$.

## 2 The xASP System

xASP generates explanation graphs under the stable model semantics [3]. It deals with normal logic programs, which are collections of rules of the form $h \leftarrow b$ where $h$ is an atom and $b = p_1, \ldots, p_m,\ not\ \ n_1, \ldots,\ not\ \ n_s$, $p_i$ and $n_j$ are atoms, and $not$ is the default negation operator. For a rule $r$, $r^+$ and $r^-$ denote the set of positive atoms $\{p_1, \ldots, p_m\}$ and the set of negative atoms $\{n_1, \ldots, n_s\}$, respectively. xASP utilizes the notions of supported set and derivation path [7,8] and the concept of explanation graph [5] which are illustrated using the program $P_1$ consisting of the following rules:

$(r_1)$ `m :- l(X),` *not* `d,` *not* `h(X).`
$(r_2)$ `d :- b(X), a(X).` $(r_3)$ `h(X) :- k(X), p.`
$(r_4)$ `b(1).` $(r_5)$ `a(4).` $(r_6)$ `l(1).` $(r_7)$ `k(6).`

Given the answer set $A_1 = \{l(1), m, a(4), b(1), k(6)\}$, the explanation graphs of $m$ are shown in Fig. 1. Both indicate that $m$ is true in $A_1$ because of the existence of the rule $r_1$ and the following dependencies:

- $m$ depends positively on $l(1)$, which is a fact;
- $m$ depends negatively on $h(1)$, which is false, because there is only one instance of the rule $r_3$ with the head $h(1)$. In that instance, $h(1)$ depends positively on $k(1)$ (left) or $p$ (right) and both are false because there is no rule for deriving them;
- $m$ depends negatively on $d$, which is false. That is because there are two instances of rule $r_2$ supporting the derivation of $d$, but none of them is applicable in the given answer set. In fact, both $a(1)$ and $b(4)$ are false because there are no rules for deriving them.

In general, for a node $x$ in an explanation graph $G$, if $x$ is an atom $a$ then the set of nodes directly connected to $a$—the nodes $y$ such that $(a, y, \_)$ is an edge in $G$—represents the body of a rule whose head is $x$ and whose body is satisfied by $A$. If $x$ is $\sim a$ for some atom $a$, then the set of nodes directly connected to $\sim a$ represents a set of atoms whose truth values in $A$ are such to make each rule whose head is $a$ unsatisfied by $A$. In other words, the direct connections with a node represent the *support* [7] for the node being in (or not in) the answer set under consideration.

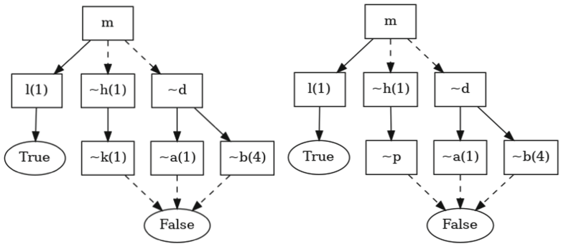

**Fig. 1.** Explanation graph of $m$

The three types of links connecting a node $x$ (corresponding to an atom $a$) to a node $y$ (for an atom $b$) in explanation graphs are as follows:

- $+$ (represented by a `solid link`) demonstrates that the truth value of $a$ depends *positively* on the truth value of $b$ w.r.t. $A$. If node $y$ is $\top$ ( $True$), atom $a$ is a fact.

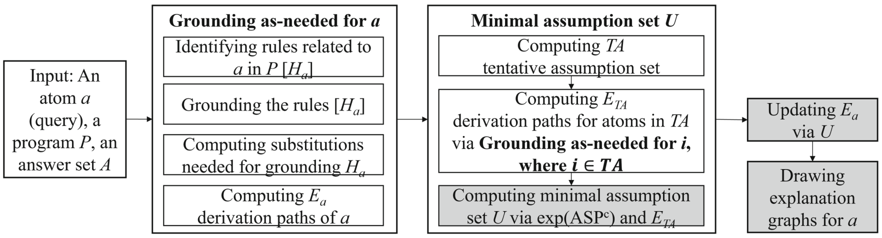
Input: An atom $a$ (query), a program $P$, an answer set $A$
Grounding as-needed for $a$
Identifying rules related to $a$ in $P$ [$H_a$]
Grounding the rules [$H_a$]
Computing substitutions needed for grounding $H_a$
Computing $E_a$ derivation paths of $a$
Minimal assumption set $U$
Computing $TA$ tentative assumption set
Computing $E_{TA}$ derivation paths for atoms in $TA$ via Grounding as-needed for $i$, where $i \in TA$
Computing minimal assumption set $U$ via exp(ASP^c) and $E_{TA}$
Updating $E_a$ via $U$
Drawing explanation graphs for $a$

**Fig. 2.** The overview of xASP system

- $-$ (represented by a `dashed link`) demonstrates that the truth value of $a$ depends *negatively* on the truth value of $b$ w.r.t. $A$. If node $y$ is $\perp$ ($False$), it means that atom $a$ is always false. Note that in our prior system *exp*($ASP^c$), this link does not exist because the atoms, that are always false, have been simplified during the grounding process of `clingo`.
- $\circ$ (represented by a `dotted link`) is used in the case in which node $y$ is $assume$, which means that atom $a$ is assumed to be false (see in examples in [5,7]).

## 2.1 Overview of xASP

Figure 2 shows the overview of xASP. The large two boxes represent the two main phases of xASP, *grounding as-needed* and *computing a minimal assumption set*. The grey boxes are implemented via *exp*($ASP^c$).
*Grounding as-needed*: xASP computes the set of ground rules that are necessary for the construction of explanation graphs of $a$ and the set of derivation paths of $a$ given $A$. It starts by identifying the rules related to $a$, e.g., rules whose head is $a$ or an atom that $a$ depends on. Afterwards, these rules are grounded, taking into consideration the given answer set. Finally, the derivation paths of $a$ and its dependencies are obtained via the computation of supported sets which is the focus of this paper (Sect. 2.2).
*Computing a minimal assumption set*: xASP computes $E_{TA}$, the set of derivation paths of all atoms in the tentative assumption set $TA$ and a minimal assumption set $U$ of $A$. It then utilizes *exp*($ASP^c$) [7] to construct explanation graphs for $a$.

## 2.2 Computing Derivation Paths of *a*

This section presents a key algorithm for computing $E_a$, an *associative array* whose keys are $a$ or atoms that $a$ depends on, directly or indirectly, as defined via the dependency graph [6]. $E_a.keys()$ denotes the set of keys in $E_a$. For each $x \in E_a.keys()$, $E_a[x]$ is the value associated with $x$ in $E_a$ and contains the supported sets of $x$ [7].

Given two atoms $a = p(t_1, t_2, .., t_n)$ and $b = q(t'_1, t'_2, .., t'_m)$, we write $pu(a, b)$ to denote that $a$ and $b$ have the same predicates ($p = q$) and arities ($n = m$), i.e., $a$ and $b$ are possibly unifiable.

```
Algorithm 1: $PartialGrounding(a, P, A)$
    Input: $a$-a ground atom; $P$-program; $A$-an answer set
    Output: $E_a$ - set of derivation paths of $a$ and $a$'s dependencies
 1  Let $E_a[a] = []$ if $a \in A$ or let $E_a[\sim a] = []$ if $a \notin A$
 2  Let $H_a = \{(h, r^+, r^-) \mid r = h \leftarrow b \in P \wedge pu(a, h)\}$
 3  for $(h, r^+, r^-) \in H_a$ do
 4      if $\exists! \theta$ such that $\theta$ be the unifier of $\{a, h\}$ then
 5          $L = \{p \mid p \in r^+ \wedge p\theta$ is not ground$\}$
 6          for $\theta' \in \omega(L, A)$ do
 7              $\theta' \leftarrow \theta' \circ \theta$                // Composition of substitutions
 8              $D \leftarrow \{d\theta' \mid d \in r^+\}, N \leftarrow \{n\theta' \mid n \in r^-\}$
 9              if $a \in A$ then
10                  $T \leftarrow \{D \cup \{\sim n \mid n \in N\} \mid D \subseteq A, N \cap A = \emptyset\}$
11                  Append $T$ to $E_a[a]$
12              else
13                  $T \leftarrow \{\{d\} \mid d \in A \cap N\} \cup \{\{\sim n\} \mid n \in D \setminus A\}$
14                  $E_a[\sim a] \leftarrow [X \cup L \mid X \in E_a[\sim a], L \in T]$
15  $E_a[\sim a] \leftarrow [\{\perp\}] \mid \nexists (h, r^+, r^-) \in H_a$ such that $a$ is unifiable with $h$
16  $PartialGrounding(c, P, A)$ where either $c \in C$ or $\sim c \in C$, $C \in E_a[a] \cup E_a[\sim a]$
17  return $E_a$
```

**Algorithm** 1 takes a grounded atom $a$ and an answer set $A$ of program $P$ as inputs and computes $E_a$ for the construction of the explanation graph for $a \in A$ (true atom) or $a \notin A$ (false atom). $E_a$ is initialized with the empty array (line 1). Only rules in $H_a$ whose head could be unified with $a$ are involved in the partial grounding process (lines 2-14). For each $(h, r^+, r^-) \in H_a$, the grounding process starts with a unifier $\theta$ of $a$ and $h$ (line 4). $L$ is the set of positive atoms that are not grounded after substituting with $\theta$. $p\theta$ denotes that variables in atom $p$ are substituted by elements in $\theta$. Due to the restriction that variables occurring in negative atoms must appear in positive atoms, ground substitutions of atoms in $r^+$ are ground substitutions for the negative atoms in $r^-$. We define $\omega(L, A)$ as the set of potential substitutions for variables in $L$ given $A$, in which each element $\theta'$ of $\omega(L, A)$ is a set of substitutions of the form $v/t$ such that for some $x \in L$ and $x\theta' \in A$, and $\{v \mid v/t \in \theta'\} = V$ where $V$ is the set of variables in $L$. Note that $\theta'$ must be composed to a valid substitution for variables in $L$, e.g., it must not specify two different values for a variable (called conflict in the variable). In addition, some atoms in $L$ cannot be unified with any atoms in $A$ and hence are false w.r.t. $A$. Therefore, those atoms are not grounded (see Sect. 2.3). If $L = \emptyset$, $\omega(L, A)$ is empty. After obtaining substitutions, the positive and negative atoms are grounded via $\theta'$ (line 7) and supported sets of $a$ are computed which depend on the

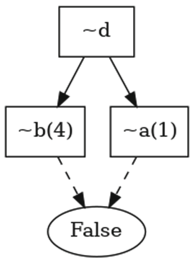

**Fig. 3.** Explanation graph of $d$

truth value of $a$ in $A$ (lines 6-14). Note that, in line 10, if $D = \emptyset$ and $N = \emptyset$, then $T = \{\top\}$, denoting that atom $a$ is a fact. Observe that if there are no rules whose head can be unified with $a$, then the atom is false (no rule in $P$ supports $a$). As such, $E_a[\sim a]$ is set to $[\{\bot\}]$, i.e. atom $a$ is false in $P$ (line 15). Unlike *exp*$(ASP^c)$, Algorithm 1 is recursively called only on atoms in supported set of $a$ (line 16).

*Example 2.* Let us reconsider program $P_1$ and compute the derivation paths for $m$. Given a ground atom $m$, there is only rule $r_1$ whose head is unifiable with $m$ where $\theta = \emptyset$. $\theta$ is not a ground substitution for positive atoms in $r_1$. Thus, answer set $A_1$ is utilized to obtain a unifier $\{X/1\}$ to substitute for atoms in the body of $r_1$, resulting in $E_m[m] = \{l(1), \sim d, \sim h(1)\}$. Algorithm 1 is called recursively on atoms $l(1)$, $d$ and $h(1)$.

- $E_m[l(1)] = [\{\top\}]$ because of rule $r_6$.
- Similar to $m$, unifier $\theta = \emptyset$ for atom $d$ is not a ground substitution for positive atoms in $r_2$. However, given $A_1$, we can conclude that there are two possible substitutions, $\{X/1\}$ and $\{X/4\}$, for $r_2$. We have $E_m[\sim d] = [\{\sim a(1), \sim b(4)\}]$. Algorithm 1 is then called for atoms $a(1)$ and $b(4)$.
  - Although the head of rule $r_5$ has the same predicate and arity as $a(1)$, $a(1)$ and $a(4)$ are not unifiable. Thus, $E_m[\sim a(1)] = [\{\bot\}]$. Similar to $a(1)$, $E_m[\sim b(4)] = [\{\bot\}]$.
- Similaly, we have $E_m[\sim h(1)] = [\{\sim k(1)\}, \{\sim p\}]$, $E_m[\sim k(1)] = [\{\bot\}]$ and $E_m[\sim p] = [\{\bot\}]$.

## 2.3 Illustrations

Figure 3 shows the explanation graph for $d$ in the case of Example 1. Unlike *s(CASP)*, in xASP the explanation for $d$ being false is that all possible ground rules whose head is $d$ have an unsatisfied body, in this case because $a(1)$ and $b(4)$ are false.

Consider another program $P'$ containing the rules:

```
(r1) m     :-  not q.          (r2) q     :-  d(X).
(r3) d(X)  :-  a(X), l.        (r4) a(1).
```

An explanation graph of $m$ w.r.t. $P'$ and answer set $A' = \{m, a(1)\}$ is shown in Fig. 4. It contains non-ground atom $d(X)$ because no atoms formed by $d/1$ occur in $A'$.

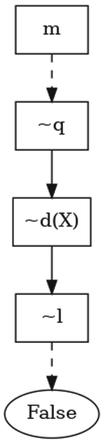

**Fig. 4.** Explanation graph of $m$

## 3 Conclusion

We presented xASP, a system for computing explanation graphs of true and false atoms w.r.t. an answer set of a program. xASP does not simplify the program before finding the explanations, thus providing faithful explanations for the truth value of the given atom. This is important to form a correct understanding of programs. Future work includes testing xASP on realistic debugging tasks and supporting the full language of clingo.

## References

1. Arias, J., Carro, M., Chen, Z., Gupta, G.: Justifications for goal-directed constraint answer set programming. Electron. Proc. Theor. Comput. Sci. **325**, 59–72 (2020)
2. Cabalar, P., Fandinno, J., Muñiz, B.: A system for explainable answer set programming. Electron. Proc. Theor. Comput. Sci. **325**, 124–136 (2020)
3. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R., Bowen, K. (eds.) Logic Programming: Proceedings of the Fifth International Conference and Symposium, pp. 1070–1080 (1988)
4. Li, F., Wang, H., Basu, K., Salazar, E., Gupta, G.: Discasp: a graph-based asp system for finding relevant consistent concepts with applications to conversational socialbots. arXiv preprint arXiv:2109.08297 (2021)
5. Pontelli, E., Son, T., El-Khatib, O.: Justifications for logic programs under answer set semantics. TPLP **9**(1), 1–56 (2009)
6. Przymusinski, T.C.: On the declarative semantics of deductive databases and logic programs. In: Foundations of Deductive Databases and Logic Programming (1988)
7. Trieu, L.L., Son, T.C., Balduccini, M.: exp(aspc): explaining asp programs with choice atoms and constraint rules. Electron. Proc. Theor. Comput. Sci. **345**, 155–161 (2021)
8. Trieu, L.L., Son, T.C., Pontelli, E., Balduccini, M.: Generating explanations for answer set programming applications. In: Artificial Intelligence and Machine Learning for Multi-domain Operations Applications III, pp. 390–403. International Society for Optics and Photonics, SPIE (2021). https://doi.org/10.1117/12.2587517