exp(ASP^c): Explaining ASP Programs with Choice Atoms and Constraint Rules[*]

Ly Ly Trieu Tran Cao Son Marcello Balduccini

New Mexico State University
New Mexico, USA

Saint Joseph's University
Pennsylvania, USA

lytrieu@nmsu.edu tson@cs.nmsu.edu mbalducc@sju.edu

We present an enhancement of `exp(ASP)`, a system that generates explanation graphs for a literal ℓ—an atom a or its default negation $\sim a$—given an answer set A of a normal logic program P, which explain why ℓ is true (or false) given A and P. The new system, `exp(ASP`^{c}`)`, differs from `exp(ASP)` in that it supports choice rules and utilizes constraint rules to provide explanation graphs that include information about choices and constraints.

1 Introduction

Answer Set Programming (ASP) [4, 5] is a popular paradigm for decision making and problem solving in Knowledge Representation and Reasoning. It has been successfully applied in a variety of applications such as robotics, planning, diagnosis, etc. ASP is an attractive programming paradigm as it is a declarative language, where programmers focus on the representation of a specific problem as a set of rules in a logical format, and then leave computational solutions of that problem to an answer set solver. However, this mechanism typically gives little insight into *why* something is a solution and *why* some proposed set of literals is not a solution. This type of reasoning falls within the scope of *explainable Artificial Intelligence* and is useful to enhance the understanding of the resulting solutions as well as for debugging programs. So far, only a limited number of approaches have been proposed [1, 6, 7]. To the best of our knowledge, no system deals directly with ASP programs with choice atoms.

In this paper, we present an improvement over our previous system, *exp(ASP)* [9], called `exp(ASP`^{c}`)`. Given an ASP program P, an answer set A, and an atom a, `exp(ASP`^{c}`)` is aimed at answering the question "why is a true/false in A?" by producing *explanation graphs* for atom a. The current system, `exp(ASP)`, does not consider programs with choice atoms and other constructs that extend the modeling capabilities of ASP. For instance, Fig. 1 shows an explanation graph for the atom $colored(1, red)$ given the typical encoding of the graph coloring problem that does not use choice rules. This explanation graph does provide the reason for the color assigned to node 1 by indicating that the node is red because it is not blue and not green. It is not obvious that this information represents the requirement that each node is colored with exactly one color. The improvement described here extends our approach with the ability to handle ASP programs containing choice rules and include constraint information in the explanation graphs.

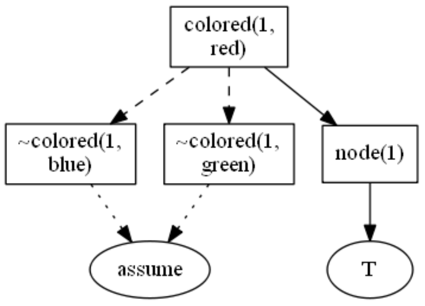


Figure 1: Explanation of $colored(1, red)$

[*]The second would like to acknowledge the partial support of the NSF 1812628 grant.

Submitted to:
ICLP 2021

2 Background: The `exp(ASP)` System

`exp(ASP)` deals with normal logic programs which are collection of rules of the form $head(r) \leftarrow body(r)$ where $head(r)$ is an atom and $body(r) = r^+, not\, r^-$ with r^+ and r^- collections of atoms in a propositional language and $not\, r^-$ denotes the set $\{\, not\, x \mid x \in r^-\}$ and *not* is the default negation.

`exp(ASP)` generates explanation graphs under the answer set semantics [2]. It implemented the algorithms proposed in [6] to generate explanation graphs of a literal ℓ (a or $\sim a$ for some atom a in the Herbrand base H of P), given an answer set A of a program P. Specifically, the system produces labeled directed graphs, called *explanation graphs*, for ℓ, whose nodes belong to $H \cup \{\sim x \mid x \in H\} \cup \{\top, \bot, \texttt{assume}\}$ and whose links are labeled with $+$, $-$ or $\circ$ (in Fig. 1, solid/dash/dot edges represent $+/-/\circ$ edges). Intuitively, for each node x, $x \not\in \{\top, \bot, \texttt{assume}\}$ in an explanation graph (E, G), the set of neighbors of x represents a support for x being true given A (see below).

The main components of `exp(ASP)` are:

1. **Preprocessing:** This component produces an *aspif* representation [3] of P that will be used in the reconstruction of ground rules of P. It also computes supported sets for atoms (or its negations) in the Herbrand base of P and stored in an associative array E.
2. **Computing minimal assumption set:** This calculates a minimal assumption set U given the answer set A and P according to the definition in [6].
3. **Computing explanation graphs:** This component uses the supported sets in E and constructs e-graphs for atoms in H (or their negations) under the assumption that each element $u \in U$ is assumed to be false.

We note that `exp(ASP)` does not deal with choice atoms [8]. The goal of this paper is to extend `exp(ASP)` to deal with choice atoms and utilize constraint information.

3 Explanation Graphs in Programs with Choice Atoms

`exp(ASP)` employs the notion of a *supported set* of a literal in a program in its construction. Given a program P, an answer set A of P, and an atom c, if $c \in A$ and r is a rule such that (*i*) $head(r) = c$, (*ii*) $r^+ \subseteq A$, and (*iii*) $r^- \cap A = \emptyset$, $support(c, r) = r^+ \cup \{\sim n \mid n \in r^-\}$; and refer to this set as a *supported set* of c for rule r. If $c \not\in A$, for every rule r such that $head(r) = c$, then $support(\sim c, r) \in \{\{p\} \mid p \in A \cap r^-\} \cup \{\{\sim n\} \mid n \in r^+ \setminus A\}$.

To account for choice atoms[1] in P, the notion of supported set needs to be extended. For simplicity of the presentation, we assume that any choice atom x is of the form $l\ \{p_1 : q_1, \ldots, p_n : q_n\}\ u$ where[2] p_i's and q_i's are atoms. Let x_l and x_u denote l and u, respectively. Furthermore, we write $c \in x$ to refer to an element in $\{p_1, \ldots, p_n\}$. For $c \in x$, $q_i \cong c$ indicates that $c : q_i$ belongs to $\{p_1 : q_1, \ldots, p_n : q_n\}$.

In the presence of choice atoms, an atom c can be true because c belongs to a choice atom that is a head of a rule r and $body(r)$ is true in A. In that case, we say that c is chosen to be true and extend $support(c, r)$ with a special atom $+choice$ to indicate that c is chosen to be true. Likewise, c can be false even if it belongs to a choice atom that is a head of a rule r and $body(r)$ is true in A. In that case, we say that c is chosen to be false and extend $support(\sim c, r)$ with a special atom $-choice$ to indicate that c is chosen to be false. Also, $q \cong c$ will belong to the support set of $support(c, r)$ and $support(\sim c, r)$.

[1]We use choice atoms synonymous with weight constraints.

[2]As we employ the *aspif* representation, this is a reasonable assumption.

The above extension only considers the case c belongs to the head of a rule. $support(c,r)$ also needs to be extended with atoms corresponding to choice atoms in the body of r. Assume that x is a choice atom in r^+. By definition, if $body(r)$ is true in A then $x_l \leq |S| \leq x_u$ where $S = \{(c,q) \mid c \in x, q \cong c, A \models c \wedge q\}$. For this reason, we extend $support(c,r)$ with x. Because x is not a standard atom, we indicate the support of x given A by defining $support(x,r) = \{S\}$. Furthermore, for each $s \in S$, $support(s,r) = \{*True\}$. When $S = \emptyset$, we write $support(x,r) = \{*Empty\}$. Similar elements will be added to $support(c,r)$ or $support(\sim c,r)$ in other cases (e.g., the choice atom belongs to r^-) or has different form (e.g., when $l = 0$ or $u = \infty$). We omit the precise definitions of the elements that need to be added to $support(c,r)$ for brevity.

The introduction of different elements in supported sets of literals in a program necessitates the extension of the notion of explanation graph. Due to the space limitation, we introduce its key components and provide the intuition behind each component. The precise definition of an explanation graph is rather involved and is included in the appendix for review. First, we introduce additional types of nodes. Besides $+$`choice`, $-$`choice`, $*$`True`, and $*$`Empty`, we consider the following types of nodes:

- *Tuples* are of the form $(x_1,\ldots,x_m,\ not\ y_1,\ldots,\ not\ y_n)$ to represent elements belonging to choice atoms (e.g., $(colored(1,blue),color(blue))$ representing an element in $1\{(colored(N,C) : color(C)\}1$). $\mathscr{T}$ denotes all tuple nodes in program P.
- *Choices* are of the form $l \leq T \leq u$ or $\sim (l \leq T \leq u)$ where $T \subseteq \mathscr{T}$. Intuitively, when $l \leq T \leq u$ (resp. $\sim (l \leq T \leq u)$) occurs in an explanation graph, it indicates that $l \leq T \leq u$ is satisfied (resp. not satisfied) in the given answer set A. $\mathscr{O}$ denotes all choices.
- *Constraints* are of the form $triggered\_constraint(x)$ or $triggered\_constraint(\sim x)$. The former (resp. latter) indicates that x is (resp. is not) true in A and satisfies all the constraints r such that $x \in r^+$ (resp. $x \in r^-$). The set of all constraints is denoted with $\mathscr{C}$.

As defined the nodes of the graph, we introduce the new types of links in explanation graphs as follows:

– $\bullet$ is used to connect literals c and $\sim c$ to $+$`choice` and $-$`choice`, respectively, where $c \in x$ and x is a choice atom in the head of a rule.
– $\diamond$ is used to connect literals c and $\sim c$ to $triggered\_constraint(c)$ and $triggered\_constraint(\sim c)$, respectively.
– $\oplus$ is used to connect a tuple $t \in \mathscr{T}$ to $*$`True`.
– $\oslash$ is used to connect a choice $n \in \mathscr{O}$ to $*$`Empty`.

4 The `exp(ASP`^{c}`)`ystem

In this section, we will focus on describing how the three main tasks in Sec. 2 are implemented. `exp(ASP`^{c}`)` uses a data structure, associative array, whose keys can be choices, tuples, constraints, or literals. For an associate array D, we use $D.keys()$ to denote the set of keys in D and $k \mapsto D[k]$ to denote that k is associated to $D[k]$. To illustrate the different concepts, we will use the program P_1 that contains a choice atom and a constraint rule as follows:

```
(r1) a :- not b, not c.    (r3) c :- not a.        (r5) 1 {m(X) : n(X)} 1 :- c.
(r2) b :- c,a.             (r4)   :- b,m(1).       (r6) n(1..2).
```

4.1 Preprocessing

Similar to `exp(ASP)`, a program is preprocessed to maintain facts and as many ground rules as possible by using the `--text` and `--keep-facts` options and replacing facts with the external statements. The *aspif* representation of the program is then obtained and processed, together with the given answer set, for generating explanation graphs. The *aspif* statements of P_1 is given in Listing 1. Let us briefly discuss the *aspif* representation before continuing with the description of other components.

Listing 1: *aspif* Representation of P_1

```
1  asp 1 0 0
2  5 1 2
3  5 2 2
4  1 0 1 3 0 1 -4
5  1 0 1 4 0 2 -5 -3
6  1 0 1 5 0 2 4 3
7  1 0 1 6 0 1 3
8  1 0 0 0 2 7 5
9  1 1 1 7 0 2 6 1
10 1 1 1 8 0 2 6 2
11 1 0 1 9 0 2 1 7
12 1 0 1 10 0 2 2 8
13 1 0 1 11 1 1 2 9 1 10 1
14 1 0 1 12 1 2 2 9 1 10 1
15 1 0 1 13 0 2 11 -12
16 1 0 0 0 2 6 -13
17 4 4 n(1) 1 1
18 4 4 n(2) 1 2
19 4 1 b 1 5
20 4 1 c 1 3
21 4 1 a 1 4
22 4 4 m(1) 1 7
23 4 4 m(2) 1 8
24 0
```

Each line encodes a statement in *aspif*. Lines starting with 4, 5, and 1 are output, external, and rule statements, respectively. Atoms are associated with integers and encoded in output statements (e.g., Line 17: 1 is the identifier of $n(1)$). External statements help us to recognize the facts in P, e.g. atom $n(1)$ ($ID = 1$) is a fact (Line 2). A rule statement r is of the form: 1 H B, where H and B are the encoding of the head and body of r, respectively. Because of page limitation, we focus on describing the rule statement whose head is a choice atom or whose body is a weight body. If the head is a choice, its encoding H has the form: $1\ n\ i_{c_1}\ \dots\ i_{c_n}$, where n is the number of head atoms and i_c is an integer identifying the atom c. E.g. $m(1)$ ($ID = 7$) and $m(2)$ ($ID = 8$) are the head choices in Lines 9 and 10, respectively, which represents rule r_5. If the body of a rule is a weight body, its encoding B has the form: $1\ l\ n\ i_{a_1}\ w_{a_1}\ \dots\ i_{a_n}\ w_{a_n}$, where $l > 0, l \in \mathbb{N}$ is the lower bound, $n > 0$ is the number of literal a_i's with $ID = i_{a_i}$ and weight w_{a_i}. E.g. Lines 13-14 contain weight body. Given an ID i that does not occur in any output statement [3, 9], we use $l(i)$ to denote the corresponding literal. Constraint r_4 is shown in Line 8. It is interesting to observe that there is one additional constraint in Line 16. By tracking integer identifiers, one can notice that Line 16 states that it cannot be the case that c is true (via Line 7) and $l(13)$ cannot be proven to be true. Lines 13-15 ensure that $l(13)$ is true if $1\{l(9);l(10)\}$ is true and $2\{l(9);l(10)\}$ cannot be proven to be true. Note that $l(9)$ and $l(10)$ are the same weight, so we ignore their weight. Line 11 states that $l(9)$ is true if $m(1)$ and $n(1)$ are true. Line 12 states that $l(10)$ is true if $m(2)$ and $n(2)$ are true. Thus, the new constraint is generated from the semantics of choice rule r_5, which is added to aspif representation.

Given the *aspif* representation P' of a program P, an associate array D_P is created where $D_P = \{(t,h) \mapsto B \mid t \in \{0,1\}, h \in H, B = \{body(r) \mid r \in P', head(r) = h\}\}$. Here, for an element $(t,h) \mapsto B$ in D_p, t is the type of head h, either disjunction ($t = 0$) or choice ($t = 1$). Furthermore, for an answer set A of P, $E_{r(P)} = \{k \mapsto V \mid k \in \{a \mid a \in A\} \cup \{\sim a \mid a \notin A\}, V = \{support(k,r) \mid r \in P\}\}$ [9].

Algorithm 1 shows how constraints are processed given the program P and its answer set A. The outcome of this algorithm is an associated array E_c. First, V_c—the set of constraints (the bodies of constraints)—is computed. Afterwards, for each body B of a constraint r in V_c, $violation$ and $support$ are computed. Each element in $violation$ requires some trigger constraint to falsify the body B, which are those in $support$. Each $choice\_support$ encodes a support for a choice atom. $triggered\_constraint(v)$, where $v \in violation$, is assigned to support the explanation of v (Line 23), and $support$ is used for the explanation of $triggered\_constraint(v)$ to justify the satisfaction of constraints containing v (Line 24). For $\{p_1 : q_1, \dots, p_n : q_n\}$ in a choice atom x, we write $\mathscr{X} = \{(c,q) \mid c \in x, q \cong c\}$ (e.g., Line 7).

During the preprocessing, the set of all negation atoms in P, $NANT(P) = \{a \mid a \in r^- \wedge r \in P\}$ [6, 9], is computed. For P_1 and the answer set $A = \{n(1), n(2), c, m(1)\}$, we have $NANT(P_1) = \{a, b, c\}$.

Algorithm 1: *constraint_preprocessing*(D, A)

```
   Input: $D$ - associative array of rules (this is $D_P$), $A$ - an answer set
1  $V_c = \{B \mid D[(0,h)] = B \wedge h = \emptyset\}$
2  $E_c \leftarrow \{\emptyset \mapsto \emptyset\}$        // Initialize an empty associative array $E_c$:  $E_c.key() = \emptyset$
3  for $B \in V_c$ do
4      $violation \leftarrow \{a \mid a \in r^+ \wedge a \in A\} \cup \{\sim a \mid a \in r^- \wedge a \notin A\}$
5      $support \leftarrow \{\sim a \mid a \in r^+ \wedge a \notin A\} \cup \{a \mid a \in r^- \wedge a \in A\}$
6      if $choice\ atom\ x\ in\ B\ and\ S = \{(c,q) \mid c \in x, q \cong c, A \models c \wedge q\}$ then
7          $\mathscr{X} = \{(c,q) \mid c \in x, q \cong c\}$
8          if $x = l\ \{p_1 : q_1, \dots, p_n : q_n\}\ u \in r^+\ and\ |S| < l\ or\ |S| > u$ then
9              $choice\_support \leftarrow \{``\sim(l <= \mathscr{X} <= u)"\}$
10         if $x = l\ \{p_1 : q_1, \dots, p_n : q_n\}\ u \in r^-\ and\ l \leq |S| \leq u$ then
11             $choice\_support \leftarrow \{``l <= \mathscr{X} <= u"\}$
12         if $x = l\ \{p_1 : q_1, \dots, p_n : q_n\} \in r^+\ or\ x = \{p_1 : q_1, \dots, p_n : q_n\}\ l-1 \in r^-\ and\ |S| < l$ then
13             $choice\_support \leftarrow \{``\sim(l <= \mathscr{X})"\}$
14         if $x = l\ \{p_1 : q_1, \dots, p_n : q_n\} \in r^-\ or\ x = \{p_1 : q_1, \dots, p_n : q_n\}\ l-1 \in r^+\ and\ |S| \geq l$ then
15             $choice\_support \leftarrow \{``l <= \mathscr{X}"\}$
16     $support \leftarrow support \cup choice\_support$
17     if $S \neq \emptyset$ then
18         $E_c[choice\_support] \leftarrow [S]$
19         $E_c[p_i] \leftarrow [\{``*True"\}]\ such\ that\ p_i \in S$
20     else
21         $E_c[choice\_support] \leftarrow [\{``*Empty"\}]$
22     for $v \in violation$ do
23         Append $\{triggered\_constraint(v)\}$ to a list $E_c[v]$
24         $E_c[triggered\_constraint(v)] \leftarrow [c \cup \{s\} \mid s \in support, c \in E_c[triggered\_constraint(v)]]$
25 return $E_c$
```

Example 1 *Let us reconsider the program P_1 and its answer set $A = \{n(1), n(2), c, m(1)\}$. The output of the preprocessing, $E_{r(P_1)}$ (left) and $E_{c(P_1)}$ (right), are as follows:*

$E_{r(P_1)}$
```
   = {
c     : [{~a}],
~a    : [{c}],
~b    : [{~a}],
m(1)  :
[{c, +choice, n(1)}],
~m(2):
[{c, -choice, n(2)}],
n(1)  : [{T}],
n(2)  : [{T}]
}
```

$E_{c(P_1)}$
```
   = {
m(1)  : [{triggered_constraint(m(1))}],
triggered_constraint(m(1))  : [{~b}],
c  : [{triggered_constraint(c)}],
triggered_constraint(c)  :
[{1<={(m(1), n(1)), (n(2), m(2))}<=1}],
1<={(m(1), n(1)), (n(2), m(2))}<=1  :
[{(m(1), n(1))}],
(m(1), n(1))  : [{*True}]
}
```

$E_{r(P_1)}$ shows that the supported set of two choice heads $m(1)$ and $m(2)$ contains $+choice$ and $-choice$, respectively, which depends on their truth values and the value of their bodies.

$E_{c(P_1)}$ shows that atom b in r_4 makes the constraint satisfied while $m(1)$ does not support the con-

straint. Thus, $\{\sim b\}$ is the support set of $triggered\_constraint(m(1))$, and $\{triggered\_constraint(m(1))\}$ is the support set of $m(1)$. For the additional constraint of P_1, $l(9)$ is true (encoded in $(m(1),n(1))$) w.r.t. A, resulting the constraint is satisfied. The truth value of c does not contribute to making the constraint satisfied. Thus, $triggered\_constraint(c)$ is added to the explanation of c.

4.2 Minimal assumption set

The idea of the algorithm for computing minimal assumption sets is as follows.

- A tentative assumption set TA [6, 9] is computed, which is the superset of minimal assumption sets.
- $E_{r(P)}$ in Sec. 4.1 is utilized to compute all derivation paths M of $a \in TA$. Then, the derivation paths M are examined whether they satisfy the cycle condition in the definition of explanation graph. During the examination of a derivation path $N \in M$, the other tentative assumption atoms derived from a are stored in a set D, which is appended to $DA[a]$ (DA is an associative array) if N is satisfied. If a is derivable from other atoms in TA, then the relation of a will be checked later in the next step and a is stored in a set T'. A set $T = TA \setminus T'$ contains atoms that must assume to be false.
- We calculate a set of minimal atoms, $min(B)$, that breaks all cycles among tentative assumption atoms via DA. Note that the number of set $min(B)$ can be more than one.
- Finally, the minimal assumption set $U = T \cup min(B)$.

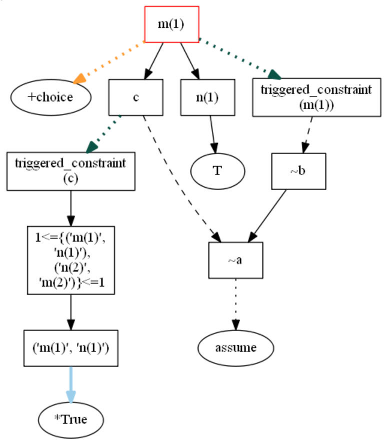


Figure 2: Explanation of $m(1)$

Example 2 *Let us reconsider the program P_1 and Example 1.*

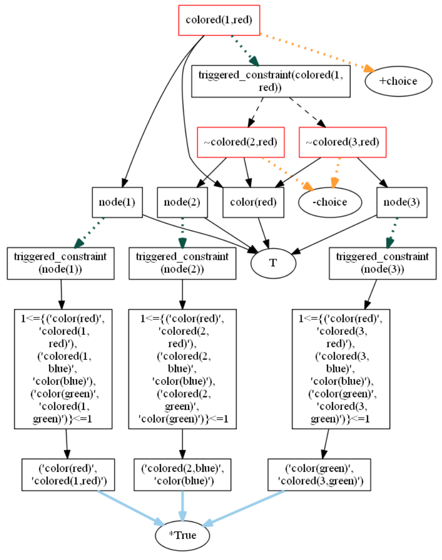
- *For the program P_1, we have: $TA = \{a,b\}$*
- *From $E_{r(P_1)}$ in Example 1, atom a is not derivable from other atoms in TA while atom b is derivable from an atom in $\{a\}$. Thus, we have $T' = \{b\}$, $T = \{a\}$ and $DA = b : [\{a\}]$. Also, there is no cycle between a and b, so $min(B) = \emptyset$. As a result, the minimal assumption set is $U = \{a\}$.*

4.3 ASP-based explanation system

In this section, we describe how the explanation graph is generated by utilizing E_r, E_c from Sec. 4.1 and the minimal assumption set U from Sec. 4.2. Let $E = \{k \mapsto V \mid k \in E_r.keys() \cup E_c.keys(), V = [r \cup c]$ *such that* $r \in E_r[k]$ *and* $c \in E_c[k]\}$. Note that $r = \emptyset$ if $\nexists k \in E_r.keys()$ and $c = \emptyset$ if $\nexists k \in E_c.keys()$. Given E, the algorithm from [9] will find the explanation graph of literal in P, taking into consideration the additional types of nodes and links.

Example 3 *For program P_1, the explanation graph of $m(1)$ is shown in Fig. 2.*

In Fig. 2, a justification for $m(1)$ depends positively on c and $n(1)$. A choice head $m(1)$ is chosen to be true. The constraint containing $m(1)$ is satisfied by A because of the truth value of b. The constraint containing c is satisfied by A because the conjunction of $n(1)$ and $m(1)$ is true.


colored(1,red)
triggered_constraint(colored(1, red))
+choice
~colored(2,red)
~colored(3,red)
node(1)
node(2)
color(red)
-choice
node(3)
triggered_constraint (node(1))
triggered_constraint (node(2))
T
triggered_constraint (node(3))
1<={('color(red)', 'colored(1, red)'), ('colored(1, blue)', 'color(blue)'), ('color(green)', 'colored(1, green)')}<=1
1<={('color(red)', 'colored(2, red)'), ('colored(2, blue)', 'color(blue)'), ('colored(2, green)', 'color(green)')}<=1
1<={('color(red)', 'colored(3, red)'), ('colored(3, blue)', 'color(blue)'), ('color(green)', 'colored(3, green)')}<=1
('color(red)', 'colored(1,red)')
('colored(2,blue)', 'color(blue)')
('color(green)', 'colored(3,green)')
*True
Figure 3: Explanation graph of $colored(1,red)$

4.4 Illustration

We illustrate the application of our updated system, `exp(ASP`^{c}`)`, to the graph coloring problem. We use a solution of the problem where each node is assigned a unique color by the choice rule: $1\{colored(X,C) : color(C)\}1 \leftarrow node(X)$.

Fig. 3 shows the explanation graph of $colored(1,red)$. Unlike Fig. 1, Fig. 3 shows that a choice head $colored(1,red)$ is chosen to be true while two choice heads, $colored(3,red)$ and $colored(2,red)$, are chosen to be false, which are represented via orange dotted links (link •). Fig. 3 displays the constraint that $node(1)$ must assign a different color with $node(3)$ and $node(2)$. This shows via the links from $colored(1,red)$ to $\sim(colored(2,red)$ and $\sim(colored(3,red)$ connected through $triggered\_constraint$ $(colored(1,red))$ (green dotted link ⋄). Also, the triggered constraints of each $node(1)$, $node(2)$ and $node(3)$, each node is assigned exactly one color, are shown via the aggregate functions in the node labels (blue solid link ⊕).

5 Conclusion

In this paper, we proposed an extension of our explanation generation system for ASP programs, `exp(ASP`^{c}`)`, which supports choice rules and includes constraint information. At this stage, we have focused on developing an approach that would simplify the program debugging task and have left a systematic investigation of its scalability for a later phase. Nonetheless, in preliminary evaluations, we successfully tested our approach on programs of growing complexity, including one from a practical application that contains 421 rules, 657 facts and has a tentative assumption set of size 44. An additional future goal is to extend `exp(ASP`^{c}`)` so that it can deal with other `clingo` constructs like the aggregates #*sum*, #*min*, etc.

References

[1] Pedro Cabalar, Jorge Fandinno & Brais Muñiz (2020): *A System for Explainable Answer Set Programming*. *Electronic Proceedings in Theoretical Computer Science* 325, p. 124–136, doi:10.4204/eptcs.325.19.

[2] M. Gelfond & V. Lifschitz (1988): *The stable model semantics for logic programming*. In R. Kowalski & K. Bowen, editors: *Logic Programming: Proc. of the Fifth International Conf. and Symp.*, pp. 1070–1080.

[3] Roland Kaminski, Torsten Schaub & Philipp Wanko (2017): *A tutorial on hybrid answer set solving with clingo*. In: *Reasoning Web International Summer School*, Springer, pp. 167–203.

[4] V. Marek & M. Truszczyński (1999): *Stable models and an alternative logic programming paradigm*. In: *The Logic Programming Paradigm: a 25-year Perspective*, pp. 375–398, doi:10.1007/978-3-642-60085-2_17.

[5] I. Niemelä (1999): *Logic programming with stable model semantics as a constraint programming paradigm*. *Annals of Mathematics and Artificial Intelligence* 25(3,4), pp. 241–273, doi:10.1023/A:1018930122475.

[6] E. Pontelli, T.C. Son & O. El-Khatib (2009): *Justifications for logic programs under answer set semantics*. *TPLP* 9(1), pp. 1–56. Available at `http://dx.doi.org/10.1017/S1471068408003633`.

[7] Claudia Schulz & Francesca Toni (2016): *Justifying answer sets using argumentation*. *Theory and Practice of Logic Programming* 16(1), pp. 59–110, doi:10.1017/S1471068414000702.

[8] Patrik Simons, Ilkka Niemelä & Timo Soininen (2002): *Extending and implementing the stable model semantics*. *Artificial Intelligence* 138(1-2), pp. 181–234, doi:10.1016/S0004-3702(02)00187-X.

[9] Ly Ly Trieu, Tran Cao Son, Enrico Pontelli & Marcello Balduccini (2021): *Generating explanations for answer set programming applications*. In: *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*, International Society for Optics and Photonics, SPIE, pp. 390 – 403. Available at `https://doi.org/10.1117/12.2587517`.