Explaining Answer-Set Programs with Abstract Constraint Atoms

Thomas Eiter and **Tobias Geibinger**
Knowledge-based Systems Group, Institute of Logic and Computation, TU Wien, 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 Introduction

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 [Erdem *et al.*, 2016; Falkner *et al.*, 2018], among them life sciences [Erdem and Oztok, 2015], health insurance [Beierle *et al.*, 2005], or psychology [Inclezan, 2015], to mention a few.

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 [Fandinno and Schulz, 2019] for a comprehensive survey and to the Related Work section for more discussion. However, most of the approaches in the literature do not support language extensions like aggregates [Faber *et al.*, 2004] or choice rules [Gebser *et al.*, 2012]. 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.

We tackle this shortcoming by introducing several formal notions of justification based on programs with *Abstract Constraint atoms (c-atoms)* [Marek and Truszczynski, 2004]. The choice of the latter is motivated by the fact that such programs embrace many of the common ASP language extensions. Our main contributions are briefly summarized as follows.

- We introduce formal notions aiming to give reasons as to why an atom is true, respectively, false in a given answer-set. To this end, we present both model-based and rule-based notions of justification to answer these questions, guided by the basic principles of sufficiency, coherence, and conciseness.
- Specifically, the model-based notion (m-justification) resorts to minimal partial model semantics, and aims to reveal what parts of the answer-set are responsible for the (non-)satisfaction of possibly negated c-atoms and, in further generalisation, to the firing or non-firing of single rules.
- The rule-based notion (r-justification) takes the whole program into account and shows in a more operational view how an atom is derived or excluded from the answer-set in terms of an r-justification chain, which is a suitable sequence of r-justifications. Different from some other proposals in the literature, such chains proceed top-down and expand on false atoms, revealing that their derivation is blocked by missing support in a non-constructive manner (cf. Section 7).
- We analyse the computational complexity of problems related to recognising and computing the novel notions of justifications on an abstract basis for representing c-atoms. The complexity ranges for the recognition problems from NP/coNP to Σ_2^p, while computing justifications is feasible in polynomial time with an NP oracle.

Our notions of justifications provide succinct reasons for the presence or absence of an atom in an answer-set. They can serve as building blocks for more comprehensive justifications, and may be used in tandem to obtain more insight on answer-sets and make them better understandable. In particular, we imagine that our notions can be utilised in an interactive fashion where the user has control over which rules get expanded and explained. This top-down approach is also more akin to query answering than the common bottom-up approach and thus more aligns with the user perspective. This is especially the case when one considers interactive justification finding, where the user starts with the atom that should be justified and proceeds by finding supporting rules (r-justifications) and looking at what is needed to fire those rules (m-justifications).

Proofs of the results will be included in an extended version.

2 Preliminaries

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) [Marek and Truszczynski, 2004], 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. Note that we require that C is *smooth*, i.e., for every $S \in C$ there is some $S' \subseteq S$ such that S' is $\subseteq$-minimal in C. For a c-atom A, we will use D^A to denote its domain and C^A to denote its satisfiers. A *c-literal* is either a c-atom A (*positive c-literal*) or its (default) negation $not\ A$ (*negative c-literal*). Similar to [Shen *et al.*, 2009], we use $\bot$ to denote $\langle D, \emptyset\rangle$ for any domain D and $\top$ for $\langle\emptyset, \{\emptyset\}\rangle$.

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.

Furthermore, $A = \langle D, C\rangle$ is called (a) *monotone* if for every $S \in C$ it holds that $S' \in C$ for every S' such that $S \subseteq S' \subseteq D$, (b) *convex* if for every $S \in C$ it holds that for every S' and S'' such that $S \subseteq S' \subseteq S'' \subseteq D$, $S'' \in C$ implies $S' \in C$, and (c) *bi-smooth* if $\overline{C}$ is also *smooth*. A c-literal $not\ A$ is called *monotone* (respectively *convex* or *bi-smooth*) if $\overline{A}$ is. This is naturally extended to sets of c-literals. Clearly, monotonicity implies convexity and any c-atom over a finite domain is bi-smooth.

Following [Oetsch *et al.*, 2018], we define *(disjunctive) logic programs* over c-atoms, which consist of a finite number of *rules* of the form

$$A_1 \vee \cdots \vee A_l \leftarrow A_{l+1}, \dots, A_m, not\ A_{m+1}, \dots, not\ A_n, \quad (1)$$

where $A_1, \dots, A_n$ are c-atoms, $l \geq 1$ and $m, n \geq 2$. We call a rule *normal* if $l = 1$; a *fact* if $l = 1$, $m = n = 2$ and $A_m = \top$, a *constraint* if $l = 1$ and $A_l = \bot$; *positive* if $n = m$; *definite* if it is normal and $|A_1^C| = 1$; *basic* if it is normal and A_1 is elementary; and *convex* if every A_i, for $l < i \leq n$, is convex. We call a program *normal*, (*positive, definite, basic, convex*) if all of its rules are. Note that, different to most of the literature, we do not allow empty rule heads or bodies, because we can express constraints with $\bot$ and facts using $\top$.

For a rule r of the form (1), $H(r) := \{A_0, \dots, A_l\}$ is the *head* of the rule, whereas $B^+(r) := \{A_{l+1}, \dots, A_m\}$ and $B^-(r) := \{A_{m+1}, \dots, A_n\}$ are the *positive* and, respectively, *negative body*. Furthermore, $B(r) := B^+(r) \cup not\ B^-(r)$, where $not\ S := \{not\ A \mid A \in S\}$ for any set of c-atoms S. The set of all propositional atoms appearing in program P is denoted by $\mathcal{A}_P$.

Sometimes we will also prefix rules with labels, i.e., write $l : r$ where l is the label and r is a rule as defined above, to give proper names to rules and ease readability.

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 $I^+ \cup I^- \supseteq S$. Intuitively, this indicates that no atoms in the set S are undefined by I. 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.

Satisfaction of c-atoms is defined as follows. This definition, which is mostly based on [Son and Pontelli, 2007], is semantically equivalent to the one of [Wang *et al.*, 2012]. However, unlike them, we do not require prefixed powersets and thus needless auxiliary definitions.

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 $not\ A$ ($I \models not\ A$) 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. In contrast, a c-atom is falsified by I, i.e., $I \models not\ A$, if for each of its satisfiers at least one of (a) or (b) hold. Intuitively, (a) L implies that the satisfier is known to be false, whereas (b) ensures that something is already known to be true which, in addition with the satisfier, makes $I^+ \cap D$ a non-satisfier. In other words, (b) states that even if the satisfier would be true, the resulting extension of the interpretation would not satisfy the c-atom.

For a set of c-literals $\mathcal{L}$, we say a (partial) interpretation I satisfies $\mathcal{L}$, denoted $I \models \mathcal{L}$, if $I \models L$ for every $L \in \mathcal{L}$; satisfies a rule r, denoted $I \models r$, if $I \models L$ for some $L \in H(r)$ whenever $I \models B(r)$. A (partial) interpretation is a *model* of $\mathcal{L}$ if $I \models \mathcal{L}$ and a *countermodel* otherwise.

Example 1. *Consider the c-atom $A = \langle D, C\rangle$, where $D = \{a, b\}$ and $C = \{\{a\}\}$, and the partial interpretation $J_1 = \langle\{a\}, \emptyset\rangle$. Now, even though $a \in J_1^+$ is known to be true, we have $J_1 \not\models A$, since $\{a, b\} \in \overline{C}$ and $b \notin J_1^-$. Furthermore, $J_1 \not\models not\ A$ because neither condition (a) nor (b) of the definition above holds. The latter does not hold, since for the only satisfier $S = \{a\}$, $U = \{a, b\}$ and thus $(U \setminus S) \cap J^+ = \emptyset$.*

Now, consider the partial interpretation $J_2 = \langle\{a\}, \{b\}\rangle$. Clearly, now $J_2 \models A$ as well as $J_2 \not\models not\ A$ hold, the latter for the same reason as above.

A seminal concept in ASP is the notion of reduct. The original *GL reduct* [Gelfond and Lifschitz, 1988] is the most known, but it is only defined over propositional programs without c-atoms. Several reducts have been introduced to handle them. Here, we use the following, which is a slight variation of the so called *FLP reduct* [Faber *et al.*, 2004].

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 $AS(P)$. Note that Oetsch et al. 2012 gave a similar definition using a different version of the FLP reduct. However, their notion of answer-set seems to coincide with our. The introduced semantics neatly generalise aggregates and choice atoms as implemented in the ASP solver DLV [Leone *et al.*, 2006] and, with restrictions, in the solver clingo [Gebser *et al.*, 2014].

3 Basic Properties of c-Literals

Our definitions allow us to derive some useful properties of (sets of) c-literals.

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

This proposition, which was used implicitly in [Son and Pontelli, 2007], allows us to use complements of c-atoms and their negation interchangeably. Furthermore, it turns out that a set of c-atoms, and thus c-literals by the above proposition, can be merged into a single c-atom.

Theorem 1. *For every set $\mathcal{L} = \{L_1, \ldots, L_n\}$ of c-literals there is a c-atom $A^{\mathcal{L}}$ such that for every partial interpretation J, we have $J \models \mathcal{L}$ iff $J \models A^{\mathcal{L}}$. Furthermore, if $\mathcal{L}$ is a set of convex c-literals, then $A^{\mathcal{L}}$ is convex as well.*

The intuition here is as follows. The domains of the c-atoms can simply be merged and the satisfiers are then all subsets of the resulting domain that are satisfiers of all c-atoms in the original when intersected with the respective domain.

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 $J_1 \neq J_2$.*

We have an intuitive result on the monotonicity of partial interpretations.

Proposition 2. *Let J_1 and J_2 be partial interpretations s.t. $J_1 \leq J_2$. Then, for every c-literal L, $J_1 \models L$ implies $J_2 \models L$.*

4 Model-based Justifications

Inspired by the philosophical study of explanation and justification, c.f. [Miller, 2019], we argue that a justification for the inclusion, or non-inclusion, of an atom in an answer-set should obey the following properties, which are intuitively described as:

Sufficiency: The justification should give *sufficient* reason to why the atom is or is not contained in the answer-set.

Coherence: The justification should be *coherent*, i.e., the justification contains nothing which is in contradiction with the answer-set.

Conciseness: The justification should be *concise*, meaning that it contains no superfluous information.

Note that the last property is not the same as the following, which we do not require.

Necessity: The justification is *necessary*, i.e., the inclusion, or non-inclusion, of the atom follows from the knowledge given in the justification. If the content of the latter were false, then the atom would not, or would respectively, be contained in the answer-set.

The reason why we do not demand necessity will become clear later on. In this section, we will introduce formal notions of justification for (non)satisfaction of c-atoms w.r.t. a given answer-set without taking the respective program into account.

4.1 Justifications for c-literals

We now come to our notion of what justifies the satisfaction of a c-atom.

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 set of all m-justifications of L w.r.t. I is denoted by $J(L, I)$.*

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 condition (i) enforces coherence, (ii) ensures that the m-justification is sufficient and (iii) describes conciseness. Furthermore, we have the following property.

Definition 3. *Let L be a c-literal and J be an* m-justification *of $I \models L$. Then we call J* definite *if $J(L, I) = \{J\}$.*

Generally, c-atoms will not have a definite m-justification, which is why we do not require this property in Definition 2.

In the case of a definite m-justification, we also have an important theorem, which shows that definite m-justifications satisfy our informal notion of "necessity".

Theorem 2. *Suppose $J = \langle J^+, J^- \rangle$ is a definite m-justification of $I \models L$. Then, for every $X \neq \emptyset$, (i) $I \setminus X \not\models L$ if $X \subseteq J^+$, and (i) $I \cup X \not\models L$ if $X \subseteq J^-$.*

Definition 2 can be readily extended to non-satisfaction.

Definition 4. *Let L be a c-literal and I be a total counter-model of L. Then J is a* (negative) m-justification *of $I \not\models L$ if J is an m-justification of $I \models \overline{L}$.*

As stated, m-justifications are not unique, but there is always at least one as we are going to show now. First, we establish the following result.

Proposition 3. *Let I be a total interpretation. Then for every c-atom A, either $I \models A$ or $I \models$ not A.*

Intuitively, the proposition states that for a total interpretation, the truth value of a c-literal cannot be undefined.

Proposition 4. *Let L be a bi-smooth c-literal and I be a total interpretation. Then there exists some $J \leq I$ such that J is an m-justification either for $I \models L$ or for $I \not\models L$.*

Example 2. *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 $\#\mathtt{sum}\{\mathtt{2 : a, 1 : b, 1 : c}\} > \mathtt{1}$. 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.

Example 3. *Consider the aggregate atom $\#\mathtt{max}\{\mathtt{1:a}, \mathtt{1:b},\ \mathtt{2:c}\} < 2$ and let $A_2 = \langle D_2, C_2\rangle$ be the corresponding c-atom, i.e., $D_2 = \{a,b,c\}$ and $C_2 = \{\emptyset, \{a\}, \{b\}, \{a,b\}\}$. Suppose we have model $I_3 = \{a,b\}$. Then, $\langle\emptyset, \{c\}\rangle$ is a definite m-justification.*

Example 4. *Consider the aggregate atom $\#\mathtt{sum}\{\mathtt{1:a}, \mathtt{1:b}, \mathtt{1:c}\} = 1$ and let $A_3 = \langle D_3, C_3\rangle$ be the corresponding c-atom, i.e., $D_3 = \{a,b,c\}$ and $C_3 = \{\{a\},\{b\},\{c\}\}$. Suppose we have a model $I_4 = \{a\}$. The only valid m-justification is $\langle\{a\},\{b,c\}\rangle$, which intuitively says that $I_4 \models A_3$ holds, i.e., the sum is exactly 1, because a is true and b and c are false.*

4.2 Justifications for Sets of c-literals

We extend our notion of justifications from c-literals to sets of c-literals in the following way.

Definition 5. *Let $\mathcal{L} = \{L_1, \dots, L_n\}$ be a set of c-literals and I be a total model of $\mathcal{L}$. Then, a partial model J of $\mathcal{L}$ is called*

(i) a locally concise (positive) m-justification *if there is a sequence of m-justifications $J_1, \dots, J_n$ such that $J_i \in J(L_i, I)$ and $J = \langle\bigcup_{1\le i\le n} J_i^+, \bigcup_{1\le i\le n} J_i^-\rangle$, and*

(ii) a globally concise (positive) m-justification *if there is no partial interpretation $J' < J$ such that $J' \models \mathcal{L}$.*

Clearly, every m-justification for a set is sufficient and coherent. Notably, according to our definition, an m-justification for a set of c-literals is only the $\leq$-minimal partial model if it is globally concise. The basic idea is that global conciseness might be too strong in some cases, since it prefers m-justifications that are concise and sufficient for multiple c-literals. In contrast, local conciseness only requires the m-justification to be the union of m-justifications for all c-literals.

Example 5. *Consider the set of c-literals $\mathcal{L}_1 = \{A_4, A_5\}$, where A_4 represents the $\#\mathtt{sum}\{\mathtt{2:a}, \mathtt{1:b}, \mathtt{1:c}\} > 1$ and A_5 represents $\#\mathtt{max}\{\mathtt{2:a}, \mathtt{1:b}\} > 1$. Furthermore, let $I_5 = \{a,b,c\}$ an interpretation. We know from previous examples that $J(A_4, I_5) = \{\langle\{a\},\emptyset\rangle, \langle\{b,c\},\emptyset\rangle\}$ and we clearly have $J(A_5, I_5) = \{\langle\{a\},\emptyset\rangle\}$. We have one globally concise m-justification for $\mathcal{L}_1$, $\langle\{a\},\emptyset\rangle$, while there are two locally concise m-justifications, namely, $\langle\{a\},\emptyset\rangle$ and $\langle\{a,b,c\},\emptyset\rangle$. Global conciseness would ignore one of the two m-justifications for A_4. Of course, it can be argued that $\langle\{a\},\emptyset\rangle$ is in fact a stronger m-justification for $\mathcal{L}_1$ than $\langle\{a,b,c\},\emptyset\rangle$ as it is non-redundant. However, depending on the context, the additional m-justifications obtained through local conciseness provide broader, more inclusive explanations and more diversity.*

We can again easily adopt our notions to non-satisfaction.

Definition 6. *Let $\mathcal{L}$ be a set of c-literals and I be a total countermodel of $\mathcal{L}$. Then, a partial interpretation J is a* locally concise (negative) m-justification *of $I \not\models \mathcal{L}$ if J is an m-justification of $\overline{L}$ for some $L \in \mathcal{L}$. Furthermore, J is* globally concise *if there is no partial interpretation $J' < J$ such that J' is an m-justification of $I \models \overline{L}$ for some $L \in \mathcal{L}$.*

For convenience, given a total interpretation I, a set $\mathcal{L}$ of c-literals and a partial interpretation J, we say that J is an m-justification for $\mathcal{L}$ if it is an m-justification for $I \models \mathcal{L}$ or $I \not\models \mathcal{L}$. Similarly, for single c-literals.

5 Rule-based Justifications

We now come to giving rule-based justifications, i.e., we aim to explain why atoms are in a given answer-set, but in addition to the previous section, we take the program into account as well. First, we give the following definitions.

Definition 7 (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 $I \models A$.*

The idea behind Presumptuous entailment is that we want to define inferences under certain preconditions; viz., when we have some information about what is, or is not, true.

The next definition is based the known concept of *support* in answer-set semantics [Lifschitz, 2010].

Definition 8. *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 $I' \models B(r)$.*

The intuition behind failed supports is that they are rules whose body is not satisfied, but if they would be active, then the respective atom would also have to be true.

We now come to the main notions of this section.

Definition 9. *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 annoted 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. Recalling our informal notions of what properties a justification should have, we see that "sufficiency" and "coherence" are defined by reflection. Furthermore, both conditions (a) and (b) ensure that the set of rules Q is minimal and thus contains no redundant rules.

Example 6. *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 $(d^+, \{r_1\}, \langle\emptyset, \{c\}\rangle)$.

The next theorem shows that r-justifications always exist.

Theorem 3. *Let P be a program and $I \in AS(P)$ be an answer-set of P. Then for every atom a, there is an r-justification (a°, Q, J) w.r.t. P and I.*

We want to utilise this notion to justify atoms step by step. We do this by chaining r-justifications providing something akin to rule derivation, but with negative atoms mixed in.

Definition 10. *We call a sequence of r-justifications w.r.t. P and I, $\mathcal{J} = (a_1^\circ, Q_1, J_1), \dots, (a_n^\circ, Q_n, J_n)$ an* r-justification chain *for a_1 if for every $(a_i^\circ, Q_i, J_i) \in \mathcal{J}$, it holds that*

(a) for each $a \in J_i^+$ some (a_j^+, Q_j, J_j) exists such that $a_j = a$ and either $j > i$ or $a_j = a_1$ and there is no subsequence $(a_{k_1}^+, Q_{k_1}, J_{k_1}), \dots, (a_{k_\ell}^+, Q_{k_\ell}, J_{k_\ell})$ of $\mathcal{J}$ where $k_1 = 1$, $k_\ell = i$, and $a_{k_{h+1}} \in J_{k_h}^+$, for all $1 < h < \ell$;

(b) for each $a \in J_i^-$ some (a_j^-, Q_j, J_j) exists s.t. $a_j = a$;

(c) if $i > 1$, then there is some (a_j°, Q_j, J_j) such that $a_i \in J_j^\circ$ and $j < i$; and

(d) $a_i = a_j$ for $1 \leq j \leq n$ implies $i = j$.

The intuition here is that positive information must be justified recursively, which eventually bottoms out, while negative information can be cyclic. Condition (a) ensures that each positive premise of a positive r-justification has some positive r-justification that comes later, with an exception for the case when the premise in question is the starting atom a_1 to be positively justified: if there is no chain as described, the r-justification (a_i^+, Q_i, J_i) is definitely not needed to derive the atom a_1, but only to "block" the derivation of some other atom. Example 9 shows such a case. Condition (b) enforces that negative premises are justified somewhere in the chain, but not necessarily after their usage. Condition (c) merely ensures that the chain does not contain unnecessary r-justifications, whereas (d) enforces that each atom appears at most once.

Example 7. *Consider $P_2 = \{r_4 : \langle\{a, b\}, \{\{a\}, \{b\}\}\rangle \leftarrow a\}$. Clearly, the only answer-set of P_2 is $\emptyset$ and the r-justifications for a and b not being true are $(a^-, \{r_4\}, \langle\emptyset, \{a\}\rangle)$ and $(b^-, \emptyset, \langle\emptyset, \{a\}\rangle)$. In fact the former is also an r-justification chain, since a not being true is justified by itself and the non-firing of rule r_4.*

Note that for negative r-justifications, we do not require that the assumptions are justified by further r-justification to the right, but rather that they appear somewhere in the chain.

Example 8. *Let us consider $P_3 = \{r_5 : a \leftarrow \langle\{b\}, \emptyset\rangle, r_6 : b \leftarrow \langle\{a\}, \emptyset\rangle\}$. Now, P_3 has two answer-sets $\{a\}$ and $\{b\}$. The justification chain for a w.r.t $\{a\}$ is then $(a^+, \{r_5\}, \langle\emptyset, \{b\}\rangle), (b^-, \{r_6\}, \langle\{a\}, \emptyset\rangle)$.*

The intuition here is that negative atoms have no constructive role in the answer-set and thus do not have to be built "bottom-up". There can be situations where a positive r-justification requires a positive atom that has already been justified.

Example 9. *Consider the program $P_4 = \{r_7 : a \leftarrow not\ b$, $r_8 : b \leftarrow not\ c$, $r_9 : c \leftarrow a\}$. Now, P_4 has answer-set $\{a, c\}$ and an r-justification chain for a is $(a^+, \{r_7\}, \langle\emptyset, \{b\}\rangle)$, $(b^-, \{r_8\}, \langle\{c\}, \emptyset\rangle), (c^+, \{r_9\}, \langle\{a\}, \emptyset\rangle)$.*

In the last element of the chain c depends on a being true which has already been justified in the beginning.

For an answer-set, r-justification chains can always be found for every atom.

Theorem 4. *Let P be a program and $I \in AS(P)$ be an answer-set of P. Then for every atom a, there is an r-justification chain $\mathcal{J}$ for a w.r.t. P and I.*

However, not all r-justification chains will offer the most insight.

Example 10. *For the program $P_5 = \{r_{10} : a \leftarrow b,\ r_{11} : b \leftarrow not\ c,\ r_{12} : \perp \leftarrow c\}$, 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.*

In the above example, the derivation of b through r_{11} is hidden in the first element of the chain. Ideally we want to decompose such an r-justification and put an r-justification for b before the one for a. This idea motivates the next definition.

Definition 11. *Given an r-justification (a^+, Q, J) we call r-justifications $(a_1^+, Q_1, J_1), (a_2^+, Q_2, J_2)$ an* elaboration *of it if $a = a_1$, $Q_1, Q_2 \subset Q$, $J_1^+ = J^+ \cup \{a_2\}$, $J_2^+ = J^+$ and $J_1^-, J_2^- \subseteq J^-$. We call an r-justification* elaborated *if there is no elaboration for it, and an r-justification chain* elaborated *if all its elements are elaborated.*

Example 11. *Reconsider program P_5 from Example 10. Then $\mathcal{J}$ in the example 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.*

Example 12. *We have the program $P_6 = \{r_{13} : a \vee b \leftarrow \top,\ r_{14} : c \leftarrow a,\ r_{15} : c \leftarrow b,\ r_{16} : \langle\{a, b\}, \{\{a, b\}\}\rangle \leftarrow c\}$.*

Now, an r-justification for a is $(a^+, P_6, \langle\emptyset, \emptyset\rangle)$ which is not elaborated. It might be tempting to try to elaborate it using b, but $(a^+, \{r_{15}, r_{16}\}, \langle\{b\}, \emptyset\rangle), (b^+, P_6, \langle\emptyset, \emptyset\rangle)$ is not a valid elaboration since $P_6 \not\subset P_6$ and b can only be derived using the whole program. However, $(a^+, \{r_{15}, r_{16}\}, \langle\{c\}, \emptyset\rangle), (c^+, \{r_{13}, r_{14}, r_{15}\}, \langle\emptyset, \emptyset\rangle)$ is in fact an elaboration for a. This can be motivated by highlighting that while both b and c can be used to derive a, the latter atom requires less rules to do so.

It is of course interesting to consider how m-justification from the previous section and r-justification relate. For a subclass of programs, namely normal and convex programs, we have the following result showing a strong correspondence.

Theorem 5. *Let P be a normal, convex program and $I \in AS(P)$. For each elaborated, concise r-justification (a^+, Q, J), it holds that $Q = \{r\}$ is a singleton set and J is a globally concise m-justification for $I \models B(r)$ w.r.t. P and I.*

The reason for this result is that for such programs we can define an immediate consequence operator for definite programs. Now, the restriction to normal programs is necessary because otherwise the reduct defined in Section 2 would not yield a definite program. In contrast convexity is required to ensure that the fixpoint of the consequence operator is indeed an answer-set. The next examples also show that we do not have such elaborate r-justifications in general.

Example 13. *Suppose we have the program* $Q = \{r_{17} : p \leftarrow q,\ r_{18} : q \leftarrow p\}$ *which we use to define* $P_7 = Q \cup \{r_{19} : p \vee q \leftarrow \top\}$ *and* $P_8 = Q \cup \{r_{20} : p \leftarrow \langle\{p, q\},\ \{\emptyset, \{p, q\}\}\rangle\}$. *The former is clearly convex but disjunctive, whereas the latter is normal but not convex. Now,* $\{p, q\}$ *is the sole answer-set for both programs but the elaborated r-justification for* p *w.r.t.* P_7 *is* $(p^+, \{r_{17}, r_{19}\}, \langle\emptyset, \emptyset\rangle\})$ *and the one w.r.t.* P_8 *is* $(p^+, \{r_{17}, r_{20}\}, \langle\emptyset, \{q\}\rangle\})$. *In both cases it is not possible to use a single rule to derive* p.

In the final part of this section, we will show how m-justification and r-justification can be jointly used.

Example 14. *Consider the program*

$$\begin{aligned} P_9 = \{r_{21} : a \leftarrow \#\mathtt{sum}\{1 : \mathtt{b}, 2 : \mathtt{d}\} > 2,\\ \#\mathtt{sum}\{3 : \mathtt{d}, 2 : \mathtt{c}\} < 5,\\ r_{22} : d \leftarrow\ b,\ r_{23} : \{\mathtt{b}; \mathtt{c}\} \leftarrow\ \top\} \end{aligned}$$

where $\{\mathtt{b}; \mathtt{c}\}$ *represents a choice between some subset of* b *and* c. *Given the answer-set* $\{a, b, d\}$, *a possible r-justification chain for* a *is* $(a^+, \{r_{21}\}, \langle\{b, d\}, \{c\}\rangle\})$, $(d^+, \{r_{22}\}, \langle\{b\}, \emptyset\rangle\})$, $(b^+, \{r_{23}\}, \langle\emptyset, \{c\}\rangle\})$, $(c^-, \emptyset, \langle\emptyset, \emptyset\rangle)$. *Looking at the first element of the chain, we see that rule* r_{21} *derives* a *based on* $\langle\{b, d\}, \{c\}\rangle$. *If we now look at the m-justifications for the body atoms of* r_{21}, *i.e.,* $\#\mathtt{sum}\{1 : \mathtt{b}, 2 : \mathtt{d}\} > 2$ *and* $\#\mathtt{sum}\{3 : \mathtt{d}, 2 : \mathtt{c}\} < 5$, *we obtain* $\langle\{b, d\}, \emptyset\rangle$ *and* $\langle\emptyset, \{c\}\rangle$. *In other words, the truth of* b *and* d *satisfies the first aggregate, whereas the second aggregate is true because* c *is false in the answer-set.*

The interplay between m-justifications and r-justifications is can especially be beneficial in an interactive setting as the next example shows.

Example 15. *Assume we have the following in the common ASP-Core-2 syntax [Calimeri* et al.*, 2020]:*

$$\begin{aligned} P_{10} = \{r_{24} : top(1) \leftarrow emp(1),\\ \#\mathtt{max}\{V, M : sold(1, M, V)\} > 10,\\ \#\mathtt{sum}\{V, M : sold(1, M, V)\} \geq 20,\\ r_{25} : emp(1),\ r_{26} : sold(1, 1, 12),\\ r_{27} : sold(1, 2, 8),\ r_{28} : sold(1, 3, 1)\} \end{aligned}$$

Intuitively, the program contains facts about an employee and their sales for the last three months. Rule r_{24} *states that employee 1 is a top earner if she sold more than 10 units in at least one month and the total units she sold is at least 20. The single answer-set of* P_{10} *contains the atom* $top(1)$ *indicating that employee 1 is a top earner. Suppose now the user has access to an interactive explanation system and asks why employee 1 is a top earner. The system can start the explanation process by giving an r-justification for* $top(1)$, *for instance* $(top(1), \{r_{24}, r_{25}, r_{26}, r_{27}\}, \langle\emptyset, \emptyset\rangle)$. *Elaboration of this r-justification would not really yield additional insight, but the user can instead opt to view an m-justification for the body of* r_{24}. *The unique m-justification is* $\langle\{emp(1), sold(1, 1, 12), sold(1, 2, 8)\}, \emptyset\rangle$ *indicating that employee 1 was designated as a top earner because she sold 12 units in the first month and 8 in the second. The process can even be further fine-grained – if the user wishes – by giving the m-justifications of the individual body literals. For example, the first aggregate has the m-justification* $\langle\{sold(1, 1, 12)\}, \emptyset\rangle$, *which tells the user that the sales in the first month were necessary to satisfy this part of the body of the rule.*

6 Complexity

We now consider complexity and assume basic familiarity with complexity theory, cf. [Papadimitriou, 1994].

6.1 Computational Basis

Before we present our study of the complexity of the problems emerging from what we have discussed so far, we first have to give some assumptions on the underlying representation and evaluation of c-atoms. We do not commit to a specific formalism, but rather request that any representation $R(A) = (D, R(C))$ of a c-atom A satisfies the following property.

Definition 12. *Let* $A = \langle D, C\rangle$ *be a c-atom and* J *be a partial interpretation. Then* $A_J = \langle D, C_J\rangle$ *where* $C_J = \{S \in C \mid S \supseteq J^+, S \cap J^- = \emptyset\}\rangle$ *is the* fixing *of* J *in* A*; it is* total, *if* J *is total on* D.

Assertion 1. *Given a c-atom* $A = \langle D, C\rangle$ *(i.e.,* $R(A)$*) and a total fixing* I *of* A, *deciding whether* $C_I \neq \emptyset$ *is in* P.

In other words, checking whether a set S is a satisfier (seen as a total fixing) of a c-atom A can be done in polynomial time. For example, using for the representation $R(A)$ of A simply C as an explicit collection, a Boolean formula, some (perhaps restricted) CNF/DNF, a BDD etc. would fulfill this elementary property. We then easily obtain

Lemma 1. *Given a set* $\mathcal{L}$ *of c-literals and a partial interpretation* J, *deciding* $J \models \mathcal{L}$ *is* coNP*-complete resp.* $J \not\models \mathcal{L}$ *is* NP*-complete. Both tests are in* P, *if* J *is a total fixing of each c-literal in* $\mathcal{L}$ *(in particular, if* J *is a total interpretation).*

Both deciding $I \models \mathcal{L}$ and $I \models \mathcal{L}$ are in P, if $R(\cdot)$ fulfills stronger properties; e.g., if deciding $C \neq \emptyset$ (satisfiability) from $R(A)$ is in P and $R(\cdot)$ is polynomially closed under fixing, i.e., some representation $R(A_I)$ in the formalism is computable in polynomial time from $R(A)$ and I. E.g., DNFs, Horn CNFs, and many other representations have this property.

Furthermore, if $R(\cdot)$ is polynomially closed under conjunctions, i.e., from $R(L_i)$ of c-literals L_i (remember that negative c-literals can be replaced by complemented c-atoms), $i = 1, \ldots, n$, some $R(A)$ in the formalism is computable in polynomial time for $A = A^{\mathcal{L}}$ as in Theorem 1, we can merge all c-literals into a single c-atom without an exponential blowup. This extends to merging of c-literals if $R(\cdot)$ is also polynomially closed under negation, i.e., $R(\overline{A})$ is computable in polynomial time from $R(A)$. Clearly, Boolean formulas have these properties.

6.2 Complexity Results

We now come to our complexity results for m-justification as introduced in Section 4. We start with the problem dealing with the recognition of m-justifications.

Theorem 6. *Given a c-literal L, a total interpretation I, and a partial interpretation $J \leq I$, deciding whether J is an m-justification is* DP*-complete.*

The class DP informally contains the problems amounting to the ”conjunction” of a problem in NP and a problem in coNP; the canonical DP-complete problem is SAT-UNSAT, i.e., to decide, given Boolean formulas F, G whether F is satisfiable and G is unsatisfiable. In Theorem 6, the coNP-check is required by condition (i) of Definition 2 as partial model checking is coNP-complete. The NP-check stems from condition (ii) which requires minimality of the partial interpretation.

Now, by the computational assumptions in the previous subsection (yielding Lemma 1), we can readily obtain:

Corollary 1. *Given a set $\mathcal{L}$ of c-literals, a total interpretation I, and a partial interpretation $J \leq I$, deciding whether J is a globally concise m-justification for $\mathcal{L}$ is* DP*-complete.*

However, under local conciseness, the complexity of recognizing positive and negative m-justifications diverges.

Theorem 7. *Given a set $\mathcal{L}$ of c-literals, a total interpretation I of $\mathcal{L}$, and a partial interpretation $J \leq I$, deciding whether J is a locally concise m-justification for $\mathcal{L}$ is (a) Σ_2^{P}-complete if $I \models \mathcal{L}$ and (b) Θ_2^{P}-complete otherwise.*

The case (a) can be solved by guessing a sequence of potential m-justifications whose union is J and then checking whether each candidate is in fact an m-justification for the corresponding c-literal in $\mathcal{L}$. As this check is in DP, we obtain that the problem is in $\mathsf{P}^{\mathsf{NP}} = \Sigma_2^{\mathsf{P}}$. In case (b), deciding negative m-justification amounts to checking whether the given partial interpretation J is a model of the complement of some c-literal in $\mathcal{L}$. This can be done with parallel DP-checks thus NP checks, which puts the problem into $\mathsf{P}_{||}^{\mathsf{NP}} = \Theta_2^{\mathsf{P}}$;matching hardness is by a reduction from evaluating a disjunction of DP instances. This means lower complexity than Σ_2^P but higher than DP (under common complexity hypotheses).

If we restrict ourselves to convex c-literals, most of the problems we have discussed so far become tractable.

Theorem 8. *Given a set $\mathcal{L}$ of convex c-literals, a total interpretation I of $\mathcal{L}$ and a partial interpretation $J \leq I$, deciding whether (a) J is a locally concise m-justification for $\mathcal{L}$ is* NP*-complete if $I \models \mathcal{L}$ and (b) J is a globally, respectively locally, concise m-justification is in* P *otherwise.*

This result is due to Theorem 1, which allows us to merge a set of convex c-literals semantically into a single convex c-atom. Recognising locally concise positive m-justifications is still intractable, as the guess as in Theorem 8.(a) is needed. In fact, the NP-hardness holds even for monotone c-literals.

Now we turn to the complexity of computing justifications.

Theorem 9. *Given a set $\mathcal{L}$ of c-literals and a total interpretation I, computing an m-justification for $\mathcal{L}$ is in* $\mathsf{FP}^{\mathsf{NP}}$ *and* $\mathsf{FP}_{||}^{\mathsf{NP}}$*-hard even if $\mathcal{L}$ is a singleton set.*

Indeed, with an NP oracle, we can just minimize I atom by atom. This result is not very surprising, as m-justifications resemble *prime implicants* sharing this complexity (cf. Sec. 7).

Our final complexity results for m-justifications concern definite m-justifications.

Theorem 10. *Given a c-literal L and a total interpretation I, deciding whether a definite m-justification for L exists is* coNP*-complete, and computing the latter* $\mathsf{FP}^{\mathsf{NP}}[1]$*-complete.*

Note that $\mathsf{FP}^{\mathsf{NP}}[1]$ is the class of function problems computable in polynomial time having access to exactly one NP-oracle call. Here, coNP membership holds as we only need to check for each atom in, respectively not in, the interpretation, whether it is required for the truth of L. This can be done in polynomial time and yields a partial interpretation J. If J is an m-justification, which can be verified in coNP, then it is also definite. Otherwise, there is no definite m-justification.

We now turn to r-justifications as introduced in Section 5.

Theorem 11. *Given a program P and an answer-set $I \in AS(P)$. Then deciding whether a triple (a°, Q, J) is an r-justification w.r.t. P and I is (a)* DP*-complete if $\circ = +$, and (b)* coNP*-complete if $\circ = -$. Deciding whether the triple is a concise r-justification is* DP*-complete.*

Essentially, the complexity in (a) stems from checking presumptuous entailment, which is coNP-complete, and minimality of the rules, which requires an NP-complete check. In case (b), the complexity is slightly lower. The underlying reason is that we simply have to check whether the set Q contains exactly the failed support w.r.t. P and I. This amounts to checking for each rule in Q whether it is a failed support. As all checks are in coNP, so is their conjunction. Conciseness requires an additional NP-check.

Finally, we consider problems for r-justification chains.

Theorem 12. *Given a program P, an answer-set $I \in AS(P)$ and an r-justification chain $\mathcal{J}$, deciding whether $\mathcal{J}$ is elaborated is* NP*-complete.*

Informally, we can check for an atom b in coNP whether it can be used for some elaboration. By combining the checks for all b into a single coNP-check, deciding whether some elaboration exists is in coNP, and thus the negation is in NP.

Theorem 13. *Given a program P, $I \in AS(P)$, and an atom a, computing an elaborated r-justification chain for a is in* $\mathsf{FP}^{\mathsf{NP}}$ *and* $\mathsf{FP}_{||}^{\mathsf{NP}}$*-hard even if P is normal and convex. If P is normal and all c-atoms are elementary, the problem is in* P.

Informally, we can set up an r-justification chain with positive and negative justifications of the form $(b^+, Q_b, \langle \emptyset, I^- \rangle)$ and $(b^-, Q_b, \langle I_b^+, I^- \rangle)$, resp., and repeatedly replace non-elaborated r-justifications with some elaboration. The number of repetitions is polynomial, hence this is feasible in $\mathsf{FP}^{\mathsf{NP}}$. The $\mathsf{FP}_{||}^{\mathsf{NP}}$-hardness is by a reduction from computing some m-justification (cf. Theorem 9).

The complexity results given in this section are of course for the worst case. Since we already have an answer-set, related work [Wang *et al.*, 2022] suggests that finding r-justification chains should be practically feasible. However, such investigations are subject to future work.

7 Related Work

Our m-justifications naturally link to *prime implicants*, cf. [Umans *et al.*, 2006], which are smallest formulas (in the context of propositional logic, a non-redundant conjunction of literals) that imply a given formula. and they were used to offer explanations for machine learning classifiers [Shih *et al.*, 2018; Ignatiev *et al.*, 2019]. Our m-justifications J of c-atoms $A = \langle D, C \rangle$ amount to prime implicants of a Boolean formula $F(A)$ associated with A, while negative and locally concise m-justifications of (sets of) c-literals have no immediate counterpart. Nonetheless, the link may be fruitfully exploited for efficient computation of m-justifications [Jackson, 1992].

While there are many works on explainability in ASP [Fandinno and Schulz, 2019], few address programs with c-atoms. Notable ones are `stepping` [Oetsch *et al.*, 2018], which is a debugging approach, and *computations* [Liu *et al.*, 2010], which is a definition of answer-set semantics through constructive computation. However, the goals of these works differ from ours. Namely, `stepping` aims for debugging at discrepancies between the actual and an expected answer-set, considering counterfactual situations; Liu *et al.*'s computations aim to generalise answer-set semantics to programs with c-atoms. Both works essentially pursue a bottom-up construction of answer-sets, while our r-justification chains proceed top-down and are thus more goal oriented and leaner. Furthermore, Liu *et al.*'s computations give no justifications for atoms not in an answer-set, do not support disjunction, and they have slightly different semantics. E.g., program P_8 in Example 13 (which is not convex) would have the unique answer-set $\emptyset$, which differs from ASP solvers like DLV [Leone *et al.*, 2006] and clingo [Gebser *et al.*, 2014] that also yield $\{p, q\}$ as the only answer-set.

The notions for programs most similar to our r-justification chains are *β-witnesses* [Wang *et al.*, 2022] and *offline justification graphs* [Pontelli *et al.*, 2009], which recently were extended to programs with choice rules and constraints [Trieu *et al.*, 2021], respectively with variables [Trieu *et al.*, 2022]. The main difference between our approach and theirs on choice is that offline justification graphs can designate an atom to be true by the choice rule and the other atoms in the choice do not appear in the justification at all. In an r-justification, those atoms always appear as negative prerequisites. This has the benefit that in an r-justification chain, such atoms can also be constructively falsified. Furthermore, justification graphs currently neither handle disjunction nor aggregates. In contrast to justification graphs and r-justification chains, β-witnesses only encompass positive information about what is in an answer-set. The main similarity between our r-justification chains and β-witnesses is that they also use sets of rules to iteratively justify atoms, but bottom-up. Offline justification graphs do incorporate information about the atoms absent from an answer-set. Intuitively, they aim at a constructive proof of an atom using the well-founded model of the program. Atoms not contained in that model can be assumed to be false, and thus cycles in the graph prevented. This is in stark contrast to our r-justification chains, which do not require such constructive proofs for false atoms, but confine to blocked support. Furthermore, while well-founded semantics for disjunctive programs is available, cf. [Wang *et al.*, 2012], generalising justification graphs from normal propositional to disjunctive programs with c-atoms is non-trivial.

Another related notion is the *formal theory of justification* [Denecker *et al.*, 2015; Marynissen *et al.*, 2022], which aims to characterise different semantics of logic programs in terms of "justifications". Intuitively, different semantics can be formalised by which justifications are accepted. In contrast to our work, the justifications of the formal theory are potentially infinite and describe dependencies between atoms containing no direct information about rules. Furthermore, disjunction and c-atoms in the rule heads are not supported.

8 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.

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.

Acknowledgments

Tobias Geibinger is a recipient of a DOC Fellowship of the Austrian Academy of Sciences at the Institute of Logic and Computation at the TU Wien. Furthermore, this work has benefited from Humane AI Net ICT-48-2020-RIA / 952026.

References

[Beierle *et al.*, 2005] Christoph Beierle, Oliver Dusso, and Gabriele Kern-Isberner. Using answer set programming for a decision support system. In *Proceedings of the 8th International Conference (LPNMR 2005)*, volume 3662 of *LNCS*, pages 374–378. Springer, 2005.

[Calimeri *et al.*, 2020] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca, and Torsten Schaub. Asp-core-2 input language format. *Theory Practice of Logic Programming*, 20(2):294–309, 2020.

[Denecker *et al.*, 2015] Marc Denecker, Gerhard Brewka, and Hannes Strass. A formal theory of justifications. In *Proceedings of the 13th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2015)*, volume 9345 of *LNCS*, pages 250–264. Springer, 2015.

[Erdem and Oztok, 2015] Esra Erdem and Umut Oztok. Generating explanations for biomedical queries. *Theory and Practice of Logic Programming*, 15(1):35–78, 2015.

[Erdem *et al.*, 2016] Esra Erdem, Michael Gelfond, and Nicola Leone. Applications of answer set programming. *AI Magazine*, 37(3):53–68, 2016.

[Faber *et al.*, 2004] Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004)*, volume 3229 of *LNCS*, pages 200–212. Springer, 2004.

[Falkner *et al.*, 2018] Andreas Falkner, Gerhard Friedrich, Konstantin Schekotihin, Richard Taupe, and Erich C. Teppan. Industrial applications of answer set programming. *KI - Künstliche Intelligenz*, 32(2):165–176, 2018.

[Fandinno and Schulz, 2019] Jorge Fandinno and Claudia Schulz. Answering the "why" in answer set programming - A survey of explanation approaches. *Theory and Practice of Logic Programming*, 19(2):114–203, 2019.

[Gebser *et al.*, 2012] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[Gebser *et al.*, 2014] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo = ASP + control: Preliminary report. *CoRR*, abs/1405.3694, 2014.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP 2012)*, pages 1070–1080. MIT Press, 1988.

[Ignatiev *et al.*, 2019] Alexey Ignatiev, Nina Narodytska, and João Marques-Silva. Abduction-based explanations for machine learning models. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, pages 1511–1519. AAAI Press, 2019.

[Inclezan, 2015] Daniela Inclezan. An application of answer set programming to the field of second language acquisition. *Theory and Practice of Logic Programming*, 15(1):1–17, 2015.

[Jackson, 1992] Peter Jackson. Computing prime implicates. In *Proceedings of the 20th ACM Annual Conference on Computer Science (CSC 1992)*, pages 65–72. ACM, 1992.

[Leone *et al.*, 2006] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[Lifschitz, 2010] Vladimir Lifschitz. Thirteen definitions of a stable model. In *Fields of Logic and Computation, Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*, volume 6300 of *LNCS*, pages 488–503. Springer, 2010.

[Liu *et al.*, 2010] Lengning Liu, Enrico Pontelli, Tran Cao Son, and Miroslaw Truszczyński. Logic programs with abstract constraint atoms: The role of computations. *Artificial Intelligence*, 174(3):295–315, 2010.

[Marek and Truszczynski, 2004] Victor W. Marek and Miroslaw Truszczynski. Logic programs with abstract constraint atoms. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI 2004)*, pages 86–91. AAAI Press / The MIT Press, 2004.

[Marynissen *et al.*, 2022] Simon Marynissen, Jesse Heyninck, Bart Bogaerts, and Marc Denecker. On nested justification systems. *Theory Practice of Logic Programming*, 22(5):641–657, 2022.

[Miller, 2019] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, 2019.

[Oetsch *et al.*, 2012] Johannes Oetsch, Jörg Pührer, and Hans Tompits. An FLP-style answer-set semantics for abstract-constraint programs with disjunctions. In *Technical Communications of the 28th International Conference on Logic Programming (ICLP 2012)*, volume 17 of *LIPIcs*, pages 222–234. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[Oetsch *et al.*, 2018] Johannes Oetsch, Jörg Pührer, and Hans Tompits. Stepwise debugging of answer-set programs. *Theory and Practice of Logic Programming*, 18(1):30–80, 2018.

[Papadimitriou, 1994] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Pontelli *et al.*, 2009] Enrico Pontelli, Tran Cao Son, and Omar Elkhatib. Justifications for logic programs under answer set semantics. *Theory and Practice of Logic Programming*, 9(1):1–56, 2009.

[Shen *et al.*, 2009] Yi-Dong Shen, Jia-Huai You, and Li-Yan Yuan. Characterizations of stable model semantics for logic programs with arbitrary constraint atoms. *Theory and Practice of Logic Programming*, 9(4):529–564, 2009.

[Shih *et al.*, 2018] Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining bayesian network classifiers. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 5103–5111. ijcai.org, 2018.

[Son and Pontelli, 2007] Tran Cao Son and Enrico Pontelli. A constructive semantic characterization of aggregates in answer set programming. *Theory Practice of Logic Programming*, 7(3):355–375, 2007.

[Trieu *et al.*, 2021] Ly Ly T. Trieu, Tran Cao Son, and Marcello Balduccini. exp(aspc) : Explaining ASP programs with choice atoms and constraint rules. In *Technical Communications of the 37th International Conference on Logic Programming (ICLP 2021)*, volume 345 of *EPTCS*, pages 155–161, 2021.

[Trieu *et al.*, 2022] Ly Ly T. Trieu, Tran Cao Son, and Marcello Balduccini. xasp: An explanation generation system for answer set programming. In *Proceddings of the 16th*

International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2022), volume 13416 of *LNCS*, pages 363–369. Springer, 2022.

[Umans *et al.*, 2006] Christopher Umans, Tiziano Villa, and Alberto L. Sangiovanni-Vincentelli. Complexity of two-level logic minimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(7):1230–1246, 2006.

[Wang *et al.*, 2012] Yisong Wang, Fangzhen Lin, Mingyi Zhang, and Jia-Huai You. A well-founded semantics for basic logic programs with arbitrary abstract constraint atoms. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*. AAAI Press, 2012.

[Wang *et al.*, 2022] Yisong Wang, Thomas Eiter, Yuanlin Zhang, and Fangzhen Lin. Witnesses for answer sets of logic programs. *ACM Transactions on Computational Logic*, 2022.