Intrinsic Prioritization in Answer Set Programming Based on an Adapted Notion of Tolerance

Marco Wilhelm[1] and **Lars-Phillip Spiegel**[2] and **Gabriele Kern-Isberner**[1]
[1]Dept. of Computer Science, TU Dortmund University, Dortmund, Germany
[2]Dept. of Mathematics and Computer Science, University of Hagen, Hagen, Germany

Abstract

Answer set programming (ASP) is a declarative programming language suited to solve complex combinatorial search problems. Prioritized ASP is the subdiscipline of ASP which aims at prioritizing the models (answer sets) of ASP programs. Common approaches to prioritized ASP require an additional input, sometimes expressed as special literals within the ASP program, that guides the prioritization of the answer sets. In this paper, to the best of our knowledge, we propose the first approach that is able to prioritize answer sets solely based on the original ASP program. For this, we adapt the notion of tolerance from conditional reasoning according to System Z and establish tolerance partitions of ASP programs which can be used for prioritization.

Introduction

Answer set programming (ASP) (Baral 2010; Gelfond and Lifschitz 1988; Gelfond 2008) is a declarative programming language and a well-established formalism in the AI area of knowledge representation and reasoning (KR). As a most important feature it provides *default negation* in rules like "$B \leftarrow A, \mathsf{not}\ C$." stating that B follows from A unless C is known. Therewith, ASP implements uncertainty into logic programming with the consequence that ASP programs can have several models (*answer sets*) constituting alternative solutions to the declared problem. In real world applications, ASP programs may lead to a vast amount of answer sets, though (cf. e.g., (Thevapalan et al. 2023)), and the need to select most suitable models among them has emerged. There are several approaches to *prioritized* resp. *preferred* ASP which tackle this problem, most notably (Brewka and Eiter 1999), but all of them have in common that they need an additional input to guide the prioritization process.

In this paper, we propose an approach to intrinsically prioritizing the answer sets of ASP programs without any additional input. For this, we borough the idea of specificity from conditional reasoning, which is based on the notion of *tolerance,* and apply it to ASP rules. A *conditional* $\delta = (B|A)$ ("If A holds, then usually B follows.") is *tolerated* by a finite set of conditionals Δ if there is a possible world ω which verifies δ ($\omega \models A \wedge B$) and does not falsify any conditional from Δ ($\omega \not\models A' \wedge \neg B'$ for all $(B'|A') \in \Delta$). Therewith, the conditionals in Δ can be categorized into an ordered tolerance partition expressing the specificity resp. normality of the conditionals, which can then be used to define a preferential model on the possible worlds as is done in *System Z* (Pearl 1990). Here, we construct tolerance partitions of ASP programs and rate answer sets with regard to the positions of the rules in these tolerance partitions that are used to generate the answer sets. We show that our approach produces the expected preference orderings in typical examples (cf. Examples 8 and 9) and prove that it satisfies a formal principle from (Brewka and Eiter 1999) conceived as a basic requirement to well-behaved prioritization in rule-based systems.

The paper is organized as follows: First, we recall some basics on ASP and conditional reasoning, particularly on System Z, and discuss related work on prioritized ASP. Then, we adapt the notion of tolerance to ASP and show how it can be used to prioritize answer sets intrinsically. Finally, we conclude with a discussion of our approach.

Preliminaries

We consider a *relational language* $\mathcal{L}$ which is defined over a *signature* $\Sigma = (\mathsf{Const}, \mathsf{Pred})$ consisting of finite sets of *constants* Const and *predicates* Pred. An *atom* is a predicate of arity n together with an n-tuple of constants or *variables*. Constants are denoted with lowercase letters, variables with uppercase letters. *Formulas* in $\mathcal{L}$ are either atoms or they are *negations* ($\neg A$), *conjunctions* ($A \wedge B$), or *disjunctions* ($A \vee B$) of formulas A and B. A *literal* L is an atom or its negation. With $\overline{L}$ we denote the *complementary literal* of L, i.e., $\overline{L} = \neg A$ if $L = A$ and $\overline{L} = A$ if $L = \neg A$ for an atom A. An expression (atom, literal, formula, etc.) is *ground* if it does not mention variables. A ground formula is called a *sentence*. The semantics of sentences is given by *interpretations* based on a truth assignment to the ground atoms in $\mathcal{L}$ as usual. We abbreviate $A \wedge B$ with AB.

Answer Set Programming (ASP). ASP (Baral 2010; Gelfond 2008) is a logic-based declarative programming language with *default negation* (Clark 1977) as its main feature. An *ASP program* is a finite set of rules

$$r\colon \quad H \leftarrow A_1, \ldots, A_n, \mathsf{not}\ B_1, \ldots, \mathsf{not}\ B_m., \qquad (1)$$

where H and all the A_i's and B_j's are literals from $\mathcal{L}$. Rules

Copyright © 2024 by the authors.
This open access article is published under the Creative Commons Attribution-NonCommercial 4.0 International License.

without literals in the *body* ($n = m = 0$) are called *facts*. A rule which mentions variables is understood as a schema that has to be grounded before solving the program. The *grounding* of a rule r means the substitution of each variable in r by a constant. The *grounding* $\mathcal{P}^g$ of an ASP program $\mathcal{P}$ is $\mathcal{P}$ in which all rules that mention variables are replaced by all of their proper groundings. A rule of the form (1) which is ground can be read as "if $A_1, \ldots, A_n$ hold and it can be assumed that $B_1, \ldots, B_m$ do not hold, then H follows." The assumption that B_j does not hold does not necessarily mean that its complement $\overline{B_j}$ is true; it is also appropriate that the truth value of B_j is unknown which differentiates the *default negation* "not" from the strict negation "¬".

Example 1. *Let* $\mathsf{Const} = \{a, b\}$*, and* $\mathcal{P}_{\mathsf{smk}} = \{r_1, \ldots, r_5\}$*,*

$r_1\colon s(b) \leftarrow .$, $\quad r_3\colon f(Y,X) \leftarrow f(X,Y).$,

$r_2\colon f(a,b) \leftarrow .$, $\quad r_4\colon s(Y) \leftarrow f(X,Y), s(X), \mathsf{not}\ \overline{s(Y)}.$,

$r_5\colon \overline{s(X)} \leftarrow \mathsf{not}\ s(X).$,

state that Bob (b) is a smoker (r_1), Alice (a) and Bob are friends (r_2), being friends is a symmetric relation (r_3), friends of smokers usually smoke, too, unless proven wrong (r_4), and usually someone is not a smoker (r_5). With $r_3^{ij}\colon f(j,i) \leftarrow f(i,j).$, $r_4^{ij}\colon s(j) \leftarrow f(i,j), s(i), \mathsf{not}\ \overline{s(j)}.$, *and* $r_5^i\colon \overline{s(i)} \leftarrow \mathsf{not}\ s(i)$*. for* $i, j \in \{a, b\}$*, we have*

$$\mathcal{P}^g_{\mathsf{smk}} = \{r_1, r_2, r_3^{aa}, r_3^{ab}, r_3^{ba}, r_3^{bb}, r_4^{aa}, r_4^{ab}, r_4^{ba}, r_4^{bb}, r_5^a, r_5^b\}.$$

The formal semantics of ground ASP programs is defined via the *Gelfond-Lifschitz-reduct* (Gelfond and Lifschitz 1991). Let $\mathcal{S}$ be a *state* which is a consistent set of ground literals, i.e., there is no ground atom $A \in \mathcal{L}$ such that $\{A, \overline{A}\} \subseteq S$. Then, the *Gelfond-Lifschitz-reduct* $\mathcal{P}^S$ of a ground ASP program $\mathcal{P}$ is the smallest set such that

if $H \leftarrow A_1, \ldots, A_n, \mathsf{not}\ B_1, \ldots, \mathsf{not}\ B_m. \in \mathcal{P}$ and, for $j \in [m]$,[1] $B_j \notin \mathcal{S}$, then $H \leftarrow A_1, \ldots, A_n. \in \mathcal{P}^{\mathcal{S}}$.

That is, $\mathcal{P}^{\mathcal{S}}$ is $\mathcal{P}$ after removing (1) all rules from $\mathcal{P}$ which are *blocked* by $\mathcal{S}$ (i.e., $\exists j \in [m]\colon B_j \in \mathcal{S}$) and (2) all default negated literals from the remaining rules. As a result, $\mathcal{P}^{\mathcal{S}}$ is a classic logic program free of default negation. For such programs a unique minimal model exists that can be obtained by recursively applying the rules in $\mathcal{P}^{\mathcal{S}}$ by beginning with the facts and proceeding with those rules the body literals of which are already inferred until no more head literals can be inferred. The resulting model is denoted with $\mathsf{Cl}(\mathcal{P}^{\mathcal{S}})$. Based on this, a state $\mathcal{S}$ is a *(stable) model* of $\mathcal{P}$ if $\mathcal{S} = \mathsf{Cl}(\mathcal{P}^{\mathcal{S}})$. Models in answer set programming are called *answer sets*. Unlike classic logic programs, ground ASP programs may have several models (answer sets) and also none. With $\mathcal{AS}(\mathcal{P})$ we denote the set of all answer sets of $\mathcal{P}$.

Example 2 (Ex. 1 cont'd)**.** *Let* $\mathcal{F} = \{s(b), f(a,b), f(b,a)\}$*. Then,* $\mathcal{AS}(\mathcal{P}^g_{\mathsf{smk}}) = \{\mathcal{S}_1\colon \{\overline{s(a)}\} \cup \mathcal{F},\ \mathcal{S}_2\colon \{s(a)\} \cup \mathcal{F}\}$ *and*

$$(\mathcal{P}^g_{\mathsf{smk}})^{\mathcal{S}_1} = \{r_1, r_2, r_3^{aa}, r_3^{ab}, r_3^{ba}, r_3^{bb}, \hat{r}_4^{ab}, \hat{r}_4^{bb}, \hat{r}_5^a\}$$

where $\hat{r}_4^{ab}\colon s(b) \leftarrow f(a,b), s(a).$, $\hat{r}_4^{bb}\colon s(b) \leftarrow f(b,b), s(b).$, *and* $\hat{r}_5^a\colon \overline{s(a)} \leftarrow .$ *The minimal model* $\mathcal{S}_1$ *of* $(\mathcal{P}^g_{\mathsf{smk}})^{\mathcal{S}_1}$ *is obtained by applying the rules* r_1*,* r_2*,* r_3^{ab}*, and* $\hat{r}_5^a$*, for example.*

[1] We abbreviate $\{1, \ldots, m\}$ with $[m]$. If $m = 0$, then $[m] = \emptyset$.

For non-ground programs $\mathcal{P}$ we set $\mathcal{AS}(\mathcal{P}) := \mathcal{AS}(\mathcal{P}^g)$. If $\mathcal{AS}(\mathcal{P}) \neq \emptyset$, then we call $\mathcal{P}$ consistent.

Conditionals and System Z. A *conditional* $(B|A)$ where A and B are sentences is a formal representation of the defeasible statement "if A, then usually B." Finite sets of conditionals serve as *belief bases*. The semantics of conditionals is based on *possible worlds*. Here, *possible worlds* are the interpretations from $\mathcal{L}$ denoted as complete conjunctions of the satisfied ground literals. That is, each ground atom from $\mathcal{L}$ occurs in a possible world once, either positive or negated. The set of all possible worlds is denoted with Ω. A *ranking function* $\kappa\colon \Omega \to \mathbb{N}_0^\infty$ (Spohn 2014) assigns to every possible world a degree of implausibility while satisfying the normalization condition $\kappa^{-1}(0) \neq \emptyset$. The higher the *rank* $\kappa(\omega)$, the less plausible the possible world ω is. The *rank* of a sentence $A \in \mathcal{L}$ is $\kappa(A) = \min_{\omega \models A} \kappa(\omega)$. A ranking function κ *accepts* a conditional $(B|A)$ if $\kappa(A \wedge B) < \kappa(A \wedge \neg B)$, and it is a *ranking model* of a belief base Δ if it accepts all conditionals in Δ. A belief base is *consistent* if it has a ranking model.

Consistent belief bases Δ have infinitely many models of different quality, e.g., in terms of inference properties. A sophisticated ranking model is provided by *System Z* (Pearl 1990) that is based on a *tolerance partition* of Δ. This so-called *Z-partition* $Z(\Delta) = (\Delta_0, \Delta_1, \ldots, \Delta_m)$ is given as follows. A conditional $\delta = (B|A)$ is *tolerated* by Δ if there is a possible world ω which *verifies* δ, i.e., $\omega \models A \wedge B$, and for all conditionals $\delta' = (B'|A')$ from Δ, either ω *verifies* δ' or δ' is *not applicable* in ω, in symbols $\omega \models B' \vee \neg A'$. Based on this, Δ_0 is the set of all conditionals from Δ which are tolerated by Δ and $(\Delta_1, \ldots, \Delta_m)$ is the Z-partition of $\Delta \setminus \Delta_0$. The *Z-rank* $Z_\Delta(\delta)$ of a conditional $\delta \in \Delta$ is the index i of the subbase $\Delta_i \in Z(\Delta)$ with $\delta \in \Delta_i$. The higher its Z-rank, the less *normal* a conditional is in the sense that it makes a statement about a more specific case. The *System Z ranking model* of Δ is given then, for $\omega \in \Omega$, by

$$\kappa^Z_\Delta(\omega) = \begin{cases} 0, & \mathsf{fal}_\Delta(\omega) = \emptyset \\ 1 + \max_{\delta \in \mathsf{fal}_\Delta(\omega)} Z_\Delta(\delta), & \text{otherwise} \end{cases}, \quad (2)$$

where $\mathsf{fal}_\Delta(\omega) = \{(B|A) \in \Delta \mid \omega \models A \wedge \neg B\}$. We illustrate System Z using the well-known penguin example.

Example 3. *Let b (bird), f (flies), and p (penguin) be ground atoms and* $\Delta_{\mathsf{bfp}} = \{\delta_1, \delta_2, \delta_3\}$ *with* $\delta_1 = (b|p)$*,* $\delta_2 = (\bar{f}|p)$*, and* $\delta_3 = (f|b)$*. There is no possible world* ω *that verifies* δ_1 *in which both* δ_2 *and* δ_3 *are verified or not applicable because* $\omega \models bp$ *implies that* δ_2 *and* δ_3 *are applicable so that* ω *would have to verify both. This, however, is not possible. Hence,* δ_1 *is not tolerated by* Δ*. Analogously,* δ_2 *is not tolerated by* Δ *but* δ_3 *is (consider* $\omega = bf\bar{p}$*). Further, with* $\omega = b\bar{f}p$ *there is a possible world which proves that* δ_1 *and* δ_2 *are tolerated by* $\{\delta_1, \delta_2\}$*. Consequently, the Z-partition of* Δ *is* $Z(\Delta) = \{\{\delta_3\}, \{\delta_1, \delta_2\}\}$*. The most plausible worlds are those in which no conditional is falsified (*$\kappa^Z_\Delta(\omega) = 0$*), followed by those which only falsify* δ_3 *(*$\kappa^Z_\Delta(\omega) = 1$*):*

$$\kappa^Z_\Delta(\omega) = \begin{cases} 0, & \text{if } \omega \in \{\bar{b}f\bar{p},\ \bar{b}\bar{f}\bar{p},\ bf\bar{p}\} \\ 1, & \text{if } \omega \in \{b\bar{f}p,\ b\bar{f}\bar{p}\} \\ 2, & \text{otherwise} \end{cases}.$$

Related Work

Answer set programming is especially suited to solve complex combinatorial problems which can be described in form of logical statements. For example, in (Thevapalan et al. 2023) ASP is used to calculate all feasible layouts of a warehouse. Applying ASP to real world problems, however, often leads to a vast amount of answer sets and it is a natural desire to prioritize these answer sets, for instance, in order to select the most preferred warehouse layout(s) among all feasible ones. Hence, the subdiscipline of *prioritized* resp. *preferred ASP* has emerged with many different approaches that have been published in the last decades (Brewka and Eiter 1999; Sakama and Inoue 2000; Delgrande, Schaub, and Tompits 2000; Wang, Zhou, and Lin 2000; Nieuwenborgh and Vermeir 2006; Zhang 2018; Wilhelm, Thevapalan, and Kern-Isberner 2023).

All of these approaches have in common that they require an additional input which guides the prioritization of answer sets. Most of them use an ordering on the rules (Brewka and Eiter 1999; Delgrande, Schaub, and Tompits 2000; Wang, Zhou, and Lin 2000; Nieuwenborgh and Vermeir 2006) or an ordering on the literals (Sakama and Inoue 2000) which occur in the program in order to infer an ordering on the answer sets. Alternative methods use external expert knowledge (Wilhelm, Thevapalan, and Kern-Isberner 2023) or special literals (Zhang 2018) to convey information about preferences. Another widely used methodology to prioritize answer sets is optimization (Gebser, Kaminski, and Schaub 2011). For example, `smodels` (Simons, Niemelä, and Soininen 2002) provides optimization statements for expressing cost functions on sets of weighted literals.

The probably best known approach to prioritized ASP is (Brewka and Eiter 1999) which is implemented in the software tool `asprin` (Brewka et al. 2015). It uses a partial ordering on the rules in the program. We explain the basic functionality of this approach by means of an ASP-based adaption of Example 3.

Example 4. *Let* $t \in \mathsf{Const}$*, and* $\mathcal{P}_{\mathsf{bfp}} = \{r_1, \ldots, r_4\}$*,*

$$r_1\colon\ p(t) \leftarrow .,\qquad r_3\colon\ \overline{f(X)} \leftarrow p(X), \mathsf{not}\ \underline{f(X)}.,$$
$$r_2\colon\ b(X) \leftarrow p(X).,\qquad r_4\colon\ f(X) \leftarrow b(X), \mathsf{not}\ \overline{f(X)}.,$$

state that Tweety (t*) is a penguin (*r_1*), penguins are birds (*r_2*), penguins usually do not fly (*r_3*), and birds usually fly (*r_4*). Then, the program* $\mathcal{P}_{\mathsf{bfp}}$ *has two answer sets,* $\mathcal{S}_1 = \{p(t), b(t), \overline{f(t)}\}$ *and* $\mathcal{S}_2 = \{p(t), b(t), f(t)\}$*. Intuitively,* $\mathcal{S}_1$ *is preferable to* $\mathcal{S}_2$ *which can be realized in the approach (Brewka and Eiter 1999) by prioritizing the rules in* $\mathcal{P}_{\mathsf{bfp}}$ *according to their indices. In particular,* r_3 *is prioritized over* r_4 *then. Prioritized rules have to be applied first which means that from* r_1 *we derive* $p(t)$ *and from* r_2 *also* $b(t)$*. Then, rule* r_3 *leads to* $\overline{f(t)}$ *and we obtain the preferred answer set* $\mathcal{S}_1$*. The answer set* $\mathcal{S}_2$ *can not be inferred because* r_4 *is always blocked by the prioritized rule* r_3*.*

With our approach it will be possible to prioritize answer sets without any additional input but solely based on the ASP program itself. For instance, we will obtain the same prioritization on $\mathcal{AS}(\mathcal{P}_{\mathsf{bfp}})$ as in Example 4 but without the need to order the rules in $\mathcal{P}_{\mathsf{bfp}}$. The crucial information in Example 4 is that penguins constitute a subclass of birds (r_2) and should inherit properties from the superclass only if nothing contrary is stated for the subclass. Note that this is implemented in Example 4 by prioritizing r_3 over r_4 (only) implicitly. Our approach will exploit the subclass-superclass-relation directly based on the notion of tolerance.

Tolerance in Answer Set Programming

We adapt concepts from conditional reasoning, particularly the notion of *tolerance*, as a formal basis for the intrinsic prioritization of answer sets according to their specificity. This seems natural because ranking models of conditional belief bases are *preferential models* (Kraus, Lehmann, and Magidor 1990) and, thus, bring along the necessary ingredients for prioritization originally. However, when adapting concepts from conditional reasoning to answer set programming one has to be aware of the different semantics of conditionals and ASP rules. Before we propose our definition of *tolerance* for ASP, we note some semantical divergences between both frameworks which help us understand that the notion of tolerance can not be translated to ASP one-to-one.

- Ranking models of belief bases are preferential models and, thus, hierarchically organized while the set of answer sets $\mathcal{AS}(\mathcal{P})$ of an ASP program $\mathcal{P}$ is "flat".
- ASP programs need facts or at least rules without positive body $A_1, \ldots, A_n$ (cf. (1)) in order to have a non-trivial model $\mathcal{S} \neq \emptyset$ while ranking models can be inferred from belief bases without facts.
- Not all ASP rules in the program have to be applicable in order to obtain answer sets while ranking models have to accept all conditionals in the belief base.
- Conditionals $(B|A)$ formalize default statements ("in case of A, B is more plausible than $\neg B$"). In contrast, ASP rules *can* be used to formalize default statements but their semantics is more lax in the sense that ASP rules can be used to model alternatives without stating any preferences as well (consider $\mathcal{P} = \{a \leftarrow \mathsf{not}\ b.,\ b \leftarrow \mathsf{not}\ a.\}$).
- Default statements whether formalized as conditionals or ASP rules leave room for exceptions. In conditional reasoning, these exceptions are handled implicitly while the exceptions of ASP rules are named explicitly in form of the default negated literals $B_1, \ldots, B_m$ in the rule body.

Different from conditionals, the explicit mentioning of exceptions in ASP rules suggests to distinguish two reasons why an ASP rule (1) is not applicable in a possible world ω: The positive body $A_1, \ldots, A_n$ of the rule might not hold in ω which is a strong reason to reject the rule, or an exceptional case out of $B_1, \ldots, B_m$ might apply which also blocks the rule but for another reason. For instance, the rule $r\colon \mathit{flies} \leftarrow \mathit{bird}, \mathsf{not}\ \mathit{penguin}.$ states that a bird flies unless it is a penguin. In the world $\neg\mathit{bird} \wedge \neg\mathit{flies} \wedge \neg\mathit{penguin}$ the rule r does not apply because the considered individual is not a bird while in the world $\neg\mathit{flies} \wedge \mathit{bird} \wedge \mathit{penguin}$ the rule r does not apply because the individual is a penguin. Intuitively, in the second case, rule r is closer to the actual

state of the possible world. Therefore, we differentiate two notions of *non-applicability* of ASP rules.

Definition 1 (Verification and Non-Applicability of ASP Rules)**.** *Let r be a ground ASP rule of the form (1), and let $\omega \in \Omega$ be a possible world. Then, ω* verifies *r if*

$$\omega \models H \wedge \bigwedge\nolimits_{i\in[n]} A_i \wedge \bigwedge\nolimits_{j\in[m]} \overline{B_j}. \tag{3}$$

The rule r is strongly not applicable *in ω if there is $i \in [n]$ with $\omega \models \overline{A_i}$, and r is* not applicable *in ω if r is strongly not applicable or there is $j \in [m]$ with $\omega \models B_j$.*

The two notions of non-applicability result in two notions of tolerance.

Definition 2 (Tolerance of ASP Rules)**.** *Let $\mathcal{P}$ be a ground ASP program, and let r be a ground ASP rule. The program $\mathcal{P}$ (strongly) tolerates r if there is a possible world $\omega \in \Omega$ such that ω verifies r and, for every rule $r' \in \mathcal{P}$, either ω verifies r' or r' is (strongly) not applicable in ω.*

By definition, if r is strongly tolerated by $\mathcal{P}$, then r is also tolerated by $\mathcal{P}$ (wrt. the weaker notion of tolerance). In contrast, there are rules which are tolerated but not strongly tolerated.

Example 5. *Let* $\mathsf{Const} = \{a, b\}$*, and let* $\mathcal{P}_{\mathsf{alt}} = \{r_1, r_2\}$ *with* $r_1\colon a \leftarrow \mathsf{not}\ b$. *and* $r_2\colon b \leftarrow \mathsf{not}\ a$. *state that a and b are alternative options. Then, r_1 is tolerated by $\mathcal{P}_{\mathsf{alt}}$ but not strongly tolerated. In order that r_1 is strongly tolerated by $\mathcal{P}_{\mathsf{alt}}$ there must be a possible world ω such that ω verifies r_1 ($\omega \models a\bar{b}$) and either ω verifies r_2 ($\omega \models \bar{a}b$) which leads to a contradiction, or r_2 is strongly not applicable in ω which is impossible because r_2 does not have a positive body. Hence, r_1 is not strongly tolerated. However, r_1 is tolerated because r_2 is not applicable in $\omega = a\bar{b}$ as* $\mathsf{not}\ a$ *blocks r_2. The same applies to r_2.*

A strong motivation for introducing strong tolerance is the fact that *normal* ASP rules (cf. (Reiter 1980) for the concept of *normal defaults*) are always verified or not applicable. That is, they are "imperceptible" wrt. (weak) tolerance.

Definition 3 (Normal ASP Rule)**.** *An ASP rule r is* normal*, if it is of the form*

$$r\colon \quad H \leftarrow A_1, \ldots, A_n, \mathsf{not}\ \overline{H}., \tag{4}$$

where $H, A_1, \ldots, A_n$ are literals from $\mathcal{L}$.

Proposition 1. *Let r be a normal ground ASP rule and let $\omega \in \Omega$ be a possible world. Then, either ω verifies r or r is not applicable in ω.*

Proof. Let r be a normal ground ASP rule of the form (4). If r is not verified in ω, then there exists $i \in [n]$ with $\omega \models \overline{A_i}$, or $\omega \models \overline{H}$. In both cases, r is not applicable. If $\omega \models \overline{H}$, then the default negated part of r blocks the application of r. □

As a consequence, if r is a ground ASP rule and $\mathcal{P}$ a ground ASP program, then normal rules $r' \in \mathcal{P}$ do not impose a condition to the tolerance of r wrt. $\mathcal{P}$. In particular, if $\mathcal{P}$ consists of normal rules only, then r is tolerated by $\mathcal{P}$ if r can be verified already. The combination of strong and normal tolerance leads to the following useful definition of *tolerance partitions* of ASP programs, though. In addition, it respects the special roles of facts and non-applicable rules.

```
Input:  ASP program P
Output: Tolerance partition T(P)
P_0 = {r ∈ P^g | r is fact} and m = 1
P^g ← P^g \ P_0
while there is r ∈ P^g that is tolerated by P^g:
    if there is r ∈ P^g that is strongly tolerated by P^g:
        P_m = {r ∈ P | r strongly tolerated by P^g}
    else: P_m = {r ∈ P | r tolerated by P^g}
    P^g ← P^g \ P_m and m ← m + 1
P_∞ = P^g
return (P_0, P_1, ... P_m, P_∞)
```

Algorithm 1: Computation of the tolerance partition $\mathcal{T}(\mathcal{P})$.

Definition 4 (Tolerance Partition of ASP Programs)**.** *Let $\mathcal{P}$ be a ground ASP program. Then, $(\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_m, \mathcal{P}_\infty)$ is a* tolerance partition *of $\mathcal{P}$ if*

- *$(\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_m, \mathcal{P}_\infty)$ is a partition of $\mathcal{P}$ with possibly empty sets $\mathcal{P}_0$ and $\mathcal{P}_\infty$,*
- *$\mathcal{P}_0$ is the set of facts from $\mathcal{P}$,*
- *for $i = 1, \ldots, m$, every rule $r \in \mathcal{P}_i$ is tolerated by the program $\mathcal{P}_i^+ := \bigcup_{j=i}^{m} \mathcal{P}_j \cup \mathcal{P}_\infty$,*
- *no rule in $\mathcal{P}_\infty$ is tolerated by $\mathcal{P}_\infty$,*
- *for $i = 1, \ldots, m$, if there are rules $r, r' \in \mathcal{P}_i^+$ where r is strongly tolerated by $\mathcal{P}_i^+$ but r' is only tolerated by $\mathcal{P}_i^+$, then r is in a partitioning set with a lower index than r'.* (priority condition)

With $\mathcal{T}(\mathcal{P})$ we denote the unique tolerance partition of $\mathcal{P}$ whose sets $\mathcal{P}_1, \ldots, \mathcal{P}_m$ are chosen maximally, beginning from $\mathcal{P}_1$. The tolerance partitions of a non-ground ASP program are the tolerance partitions of its grounding.

The integration of $\mathcal{P}_\infty$ guarantees that every ASP program has a tolerance partition and the *priority condition* will guarantee that strongly tolerated rules are prioritized over rules which are tolerated only. Algorithm 1 provides a construction method for $\mathcal{T}(\mathcal{P})$. The complexity of Algorithm 1 is $\mathcal{O}(|\mathcal{P}^g|^2 \cdot \mathsf{SAT}(\mathcal{G}))$ where $\mathcal{G}$ is the set of ground atoms in $\mathcal{P}^g$ and $\mathsf{SAT}(\mathcal{G})$ is the complexity of testing satisfiability in the propositional language induced by $\mathcal{G}$ (cf. (Pearl 1990)).

Example 6 (Ex. 4 cont'd)**.** *We recall the grounding $\mathcal{P}^g_{\mathsf{bfp}}$ of $\mathcal{P}_{\mathsf{bfp}}$. The set of facts in $\mathcal{P}^g_{\mathsf{bfp}}$ is $\mathcal{P}_0 = \{r_1\}$. For $i \in \{2, 3, 4\}$, let r_i^c be the rule r_i from $\mathcal{P}_{\mathsf{bfp}}$ after substituting variable X with any constant $c \in \mathsf{Const}$. r_2^c is not strongly tolerated by $\mathcal{P}^g_{\mathsf{bfp}} \setminus \mathcal{P}_0$ as for every possible world ω which verifies r_2^c ($\omega \models b(c)p(c)$), either r_3^c or r_4^c is neither verified nor strongly not applicable (r_3^c in case of $\omega \models f(c)$ and r_4^c in case of $\omega \models \overline{f(c)}$). Also r_3^c is not strongly tolerated because with r_2^c it follows that every possible world ω which verifies r_3^c ($\omega \models \overline{f(c)}p(c)$) must satisfy $b(c)$ which leads to a violation of r_4^c. Consequently, only r_4^c is strongly tolerated (consider ω with $\omega \models b(c)f(c)\overline{p(c)}$). Thus, $\mathcal{P}_1 = \{r_4^c \mid c \in \mathsf{Const}\}$. Note that both r_2^c and r_3^c are tolerated by $\mathcal{P}^g_{\mathsf{bfp}} \setminus \mathcal{P}_0$ because every possible world ω with $\omega \models b(c)\overline{f(c)}p(c)$ verifies r_2^c and r_3^c and blocks r_4^c such that r_4^c is not applicable*

(but not strongly not applicable). Hence, without the notion of strong non-applicability, r_2^c and r_3^c would be in the same set of the tolerance partition as r_4^c. However, here, the priority condition of Definition 4 tells us that strongly tolerated ASP rules have precedence. Further, with any possible world ω satisfying $\omega \models b(c)p(c)\overline{f(c)}$, it follows that r_2^c and r_3^c are strongly tolerated by $\mathcal{P}^g_{\mathsf{bfp}} \setminus (\mathcal{P}_0 \cup \mathcal{P}_1)$ so that $\mathcal{P}_2 = \{r_2^c \mid c \in \mathsf{Const}\} \cup \{r_3^c \mid c \in \mathsf{Const}\}$. Altogether, we have $\mathcal{T}(\mathcal{P}_{\mathsf{bfp}}) = \mathcal{T}(\mathcal{P}^g_{\mathsf{bfp}}) = (\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \emptyset)$ which is similar to the tolerance partition of Δ_{bfp} in Example 3 except for the fact in $\mathcal{P}_0$. Note that we were able to calculate $\mathcal{T}(\mathcal{P}_{\mathsf{bfp}})$ in a lifted manner because the named constant t occurs in $\mathcal{P}_0$ only which is not affected by the tolerance conditions.

Example 7. *The ASP program $\mathcal{P}_{\mathsf{ex}} = \{r_1, \ldots, r_6\}$ with*

$$r_1 \colon p \leftarrow \mathsf{not}\ \bar{p}, \qquad r_3 \colon b \leftarrow p, \qquad r_5 \colon \overline{f} \leftarrow p,$$
$$r_2 \colon \bar{p} \leftarrow \mathsf{not}\ p, \qquad r_4 \colon f \leftarrow b, \mathsf{not}\ \overline{w}, \qquad r_6 \colon w \leftarrow p,$$

is an example where the partitioning sets in $\mathcal{T}(\mathcal{P}_{\mathsf{ex}})$ are built wrt. both notions of tolerance. With $\mathcal{P}_1 = \{r_2, r_4\}$ and $\mathcal{P}_2 = \mathcal{P} \setminus \mathcal{P}_1$, we have $\mathcal{T}(\mathcal{P}_{\mathsf{ex}}) = (\emptyset, \mathcal{P}_1, \mathcal{P}_2, \emptyset)$. As no rule in $\mathcal{P}$ is strongly tolerated by $\mathcal{P}$, $\mathcal{P}_1$ is the set of (weakly) tolerated rules from $\mathcal{P}$. The set $\mathcal{P}_2$ consists of the remaining rules which are strongly tolerated by $\mathcal{P}_2$.

The tolerance partition $\mathcal{T}(\mathcal{P})$ of a ground ASP program $\mathcal{P}$ which consists of normal rules only (in this case, we call $\mathcal{P}$ a *normal* ground ASP program) relates to the Z-partition of an appropriate conditional belief base. The idea is that a normal ground ASP rule r of the form (4) has the natural translation $\delta_r = (H | \bigwedge_{i \in [n]} A_i)$ into a conditional. Based on this, for normal ground ASP programs $\mathcal{P}$, we can define its *conditional counterpart* as $\Delta_{\mathcal{P}} = \{\delta_r \mid r \in \mathcal{P}\}$, and the following proposition holds.

Proposition 2. *Let $\mathcal{P}$ be a non-empty normal ground ASP program. If $\Delta_{\mathcal{P}}$ is consistent, then there are $m \in \mathbb{N}$ and $\mathcal{P}_1, \ldots, \mathcal{P}_m \subseteq \mathcal{P}$ such that $Z(\Delta_{\mathcal{P}}) = (\Delta_{\mathcal{P}_1}, \ldots, \Delta_{\mathcal{P}_m})$ and $\mathcal{T}(\mathcal{P}) = (\emptyset, \mathcal{P}_1, \ldots, \mathcal{P}_m, \emptyset)$.*

Proof. The existence of $\mathcal{P}_1, \ldots, \mathcal{P}_m \subseteq \mathcal{P}$ with $Z(\Delta_{\mathcal{P}}) = (\Delta_{\mathcal{P}_1}, \ldots, \Delta_{\mathcal{P}_m})$ follows from the consistency of $\Delta_{\mathcal{P}}$ and, hence, the existence of $Z(\Delta_{\mathcal{P}})$. Now, let $\mathcal{P}' \subseteq \mathcal{P}$. We show that (1) $r \in \mathcal{P}'$ is strongly tolerated by $\mathcal{P}'$ iff δ_r is tolerated by $\Delta_{\mathcal{P}'}$, and (2) for $i \in [m]$ there is a strongly tolerated rule in $\mathcal{P}_i^+ = \bigcup_{j=i}^m \mathcal{P}_j$ which finishes the proof. Each $r \in \mathcal{P}'$ is of the form (4), thus strongly tolerated by $\mathcal{P}'$ iff there is $\omega \in \Omega$ with $\omega \models H \wedge \bigwedge_{i \in [n]} A_i$ and, for all $r' \in \mathcal{P}'$, say $r' \colon H' \leftarrow A'_1, \ldots, A'_{n'}, \mathsf{not}\ \overline{H}'$., either $\omega \models H' \wedge \bigwedge_{i \in [n']} A'_i$ holds or there is $i \in [n']$ with $\omega \models \overline{A'_i}$. This, however, is exactly the definition of δ_r being tolerated by $\Delta_{\mathcal{P}'}$. The existence of a strongly tolerated rule in $\mathcal{P}_i^+$ follows from the non-emptiness of $\Delta_{\mathcal{P}_i}$ and the equivalence in (1). Hence, all partitioning sets in $\mathcal{T}(\mathcal{P})$ mention strongly tolerated rules only and $\mathcal{T}(\mathcal{P})$ must be of the given form. □

In the next section we discuss how the tolerance partition of an ASP program can be used to prioritize its answer sets.

Intrinsic Prioritization of Answer Sets

We prioritize the answer sets of a ground answer set program $\mathcal{P}$ based on a ranking on the rules in $\mathcal{P}$ which is induced by the tolerance partition $\mathcal{T}(\mathcal{P})$, similar to the Z-ranks of conditionals in System Z induced by the Z-partition.

Definition 5 (Rank of an ASP Rule resp. ASP Program)**.** *Let $\mathcal{P}$ be a ground ASP program, and let $r \in \mathcal{P}$. We call $\rho_{\mathcal{P}}(r) = i$ where i is the index of $\mathcal{P}_i \in \mathcal{T}(\mathcal{P})$ with $r \in \mathcal{P}_i$ the* rank *of r. The rank of a subprogram $\mathcal{P}' \subseteq \mathcal{P}$ is $\rho_{\mathcal{P}}(\mathcal{P}') = \mathsf{mean}_{r \in \mathcal{P}'} \rho(r)$ if $\mathcal{P}' \neq \emptyset$ and $\rho_{\mathcal{P}}(\mathcal{P}') = 0$ otherwise. Hereby,* mean *is the arithmetic mean.*

The rank $\rho(\cdot)$ is not a degree of plausibility for now but specifies the specificity of a rule resp. the average specificity of a subprogram. For instance, subprograms which solely consist of facts have rank zero as facts apply generally. The idea of using the arithmetic mean in the definition of the rank of a subprogram $\mathcal{P}'$ instead of the maximum, for instance, is to take all rules from $\mathcal{P}'$ into account which leads to a more global view on the specificity of $\mathcal{P}'$.

In order to define a rank for an answer set $\mathcal{S}$ of $\mathcal{P}$, too, we identify specific subprograms of $\mathcal{P}$ which *generate* $\mathcal{S}$. Our notion of these *generating sets* is based on the *generating rules* from (Brewka and Eiter 1999).

Definition 6 (Generating Rules (cf. (Brewka and Eiter 1999)))**.** *Let $\mathcal{P}$ be a ground ASP program and let $\mathcal{S}$ be an answer set of $\mathcal{P}$. A rule $r \in P$ (of the form (1)) is a* generating rule *for $\mathcal{S}$ if $A_1, \ldots, A_n \in \mathcal{S}$ and $B_1, \ldots, B_m \notin S$. We denote the set of all generating rules for $\mathcal{S}$ with $\mathcal{G}_{\mathcal{P}}(\mathcal{S})$.*

It is easy to see that $\mathcal{S}$ is the unique answer set of $\mathcal{G}_{\mathcal{P}}(\mathcal{S})$. In particular, $\mathcal{S}$ is the set of all head literals of the rules in $\mathcal{G}_{\mathcal{P}}(\mathcal{S})$. However, there might be subsets of $\mathcal{G}_{\mathcal{P}}(\mathcal{S})$ which have $\mathcal{S}$ as an answer set as well, namely when there are multiple rules in $\mathcal{G}_{\mathcal{P}}(\mathcal{S})$ with the same head literal.

Definition 7 (Generating Sets and Ranks of Answer Sets)**.** *Let $\mathcal{P}$ be a ground ASP program, and let $\mathcal{S}$ be an answer set of $\mathcal{P}$. We call $\mathcal{G} \subseteq \mathcal{G}_{\mathcal{P}}(\mathcal{S})$ a* generating set *of $\mathcal{S}$ if $\mathcal{S}$ is an answer set of $\mathcal{G}$. Further, $\rho_{\mathcal{P}}(\mathcal{S}) = \min_{\mathcal{G} \in \mathcal{G}^{\min}_{\mathcal{P}}(\mathcal{S})} \rho(\mathcal{G})$ is the* rank *of $\mathcal{S}$ where $\mathcal{G}^{\min}_{\mathcal{P}}(\mathcal{S})$ denotes the set of all inclusion minimal generating sets of $\mathcal{S}$.*

The reason why we consider inclusion minimal generating sets instead of arbitrary generating sets for calculating the rank of an answer set $\mathcal{S}$ is that redundant rules should not influence $\rho_{\mathcal{P}}(\mathcal{S})$. Minimal subprograms of $\mathcal{P}$ from which $\mathcal{S}$ follows and which have minimal rank can be understood as most general explanations for $\mathcal{S}$. We show that our approach leads to the same prioritization of the answer sets in the Tweety-example as the approach in (Brewka and Eiter 1999). Hereby, we use the following notion of preference.

Definition 8 (Prioritization of Answer Sets)**.** *Let $\mathcal{P}$ be a ground ASP program. We define the preference order $\phi_{\mathcal{P}}$ on $\mathcal{AS}(\mathcal{P})$ by $\phi_{\mathcal{P}}(\mathcal{S}_1) \leq \phi_{\mathcal{P}}(\mathcal{S}_2)$ iff $\rho_{\mathcal{P}}(\mathcal{S}_1) \geq \rho_{\mathcal{P}}(\mathcal{S}_2)$ for $\mathcal{S}_1, \mathcal{S}_2 \in \mathcal{AS}(\mathcal{P})$. If $\phi_{\mathcal{P}}(\mathcal{S}_1) \leq \phi_{\mathcal{P}}(\mathcal{S}_2)$ and $\phi_{\mathcal{P}}(\mathcal{S}_2) \not\leq \phi_{\mathcal{P}}(\mathcal{S}_1)$, then we write $\phi_{\mathcal{P}}(\mathcal{S}_1) < \phi_{\mathcal{P}}(\mathcal{S}_2)$ and say that $\mathcal{S}_1$ is* preferred *over $\mathcal{S}_2$.*

The preference order $\phi_{\mathcal{P}}$ is a total preorder. If $\mathcal{P}$ is consistent, then there is at least one most preferred answer set

of $\mathcal{P}$. In general, there may be several most preferred answer sets (e.g., the answer sets $\{a\}$ and $\{b\}$ of $\mathcal{P}_{\mathsf{alt}}$ are likely preferred (cf. Example 5)). The preference order $\phi_{\mathcal{P}}$ is extended to non-ground ASP programs by considering the respective preference order wrt. their groundings. Note that we do not build our notion of preferences among answer sets on the falsification of rules which would be a straightforward analogy to the System Z ranking model (cf. (2)). This is because, by definition, answer sets do not falsify rules (where falsification is defined similar to the verification of rules (cf. (3)) but with negation of the head literal H) and, thus, it is not possible to compare answer sets wrt. their falsification behavior. Therefore, we focus on the rules which are verified, thus, on the rules which *confirm* the answer sets.

Example 8 (Ex. 4 and 6 cont'd)**.** *The only generating set of $\mathcal{S}_1$ is $\{r_1, r_2^t, r_3^t\}$ which leads to*

$$\begin{aligned}\rho_{\mathcal{P}_{\mathsf{bfp}}}(\mathcal{S}_1) &= \mathsf{mean}\{\rho_{\mathcal{P}_{\mathsf{bfp}}}(r_1), \rho_{\mathcal{P}_{\mathsf{bfp}}}(r_2^t), \rho_{\mathcal{P}_{\mathsf{bfp}}}(r_3^t)\}\\ &= \mathsf{mean}\{0, 2, 2\} = {}^4\!/_3 > 1.\end{aligned}$$

Analogously, $\{r_1, r_2^t, r_4^t\}$ is the only generating set of $\mathcal{S}_2$ and

$$\begin{aligned}\rho_{\mathcal{P}_{\mathsf{bfp}}}(\mathcal{S}_2) &= \mathsf{mean}\{\rho_{\mathcal{P}_{\mathsf{bfp}}}(r_1), \rho_{\mathcal{P}_{\mathsf{bfp}}}(r_2^t), \rho_{\mathcal{P}_{\mathsf{bfp}}}(r_4^t)\}\\ &= \mathsf{mean}\{0, 2, 1\} = 1.\end{aligned}$$

Because $\rho_{\mathcal{P}_{\mathsf{bfp}}}(S_1) > \rho_{\mathcal{P}_{\mathsf{bfp}}}(S_2)$ we have that S_1 uses the more specific information on average than S_2 and, hence, is preferred over S_2, i.e., $\phi_{\mathcal{P}_{\mathsf{bfp}}}(\mathcal{S}_1) < \phi_{\mathcal{P}_{\mathsf{bfp}}}(\mathcal{S}_2)$. This is in compliance with the result in Example 4.

Example 9 (Ex. 1 and 2 cont'd)**.** *The tolerance partition $\mathcal{T}(\mathcal{P}^g_{\mathsf{smk}}) = (\mathcal{P}^g_0, \mathcal{P}^g_1, \mathcal{P}^g_2, \mathcal{P}^g_\infty)$ is given by ($i, j \in \{a, b\}$)*

$$\mathcal{P}^g_0 = \{r_1, r_2\},\ \ \mathcal{P}^g_1 = \{r_3^{ij}, r_5^i\},\ \ \mathcal{P}^g_2 = \{r_4^{ij}\},\ \ \mathcal{P}^g_\infty = \emptyset.$$

We have $\mathcal{G}_{\mathcal{P}^g_{\mathsf{smk}}}(\mathcal{S}_1) = \{r_1, r_2, r_3^{ab}, r_3^{ba}, r_5^a\}$ with the unique inclusion minimal generating set $\{r_1, r_2, r_3^{ab}, r_5^a\}$ of $\mathcal{S}_1$. Removing r_2 instead of r_3^{ba} from $\mathcal{G}_{\mathcal{P}^g_{\mathsf{smk}}}(\mathcal{S}_1)$ does not lead to a generating set because $\mathcal{S}_1$ is not an answer set then. Thus, $\rho_{\mathcal{P}^g_{\mathsf{smk}}}(\mathcal{S}_1) = \mathsf{mean}\{0, 0, 1, 1\} = {}^1\!/_2$. For S_2, we have $\mathcal{G}_{\mathcal{P}^g_{\mathsf{smk}}}(\mathcal{S}_2) = \{r_1, r_2, r_3^{ab}, r_3^{ba}, r_4^{ab}, r_4^{ba}\}$ with the unique inclusion minimal generating set $\{r_1, r_2, r_3^{ab}, r_4^{ba}\}$ of $\mathcal{S}_2$, and $\rho_{\mathcal{P}^g_{\mathsf{smk}}}(\mathcal{S}_2) = \mathsf{mean}\{0, 0, 1, 2\} = {}^3\!/_4$. We obtain that $\mathcal{S}_2$ is preferred above $\mathcal{S}_1$ which can be interpreted as follows: $\mathcal{S}_1$ and $\mathcal{S}_2$ differ in whether Alice is a smoker or not. While we assume by default that persons usually do not smoke (r_5), we have the more specific information about Alice in $\mathcal{P}_{\mathsf{smk}}$ that she is a friend of a smoker (Bob; r_2) and friends of smokers usually smoke, too (r_4). Hence, we ignore the default assumption and prefer the inference that Alice is a smoker, i.e., we prefer $\mathcal{S}_2$ above $\mathcal{S}_1$. However, we do not reject $\mathcal{S}_1$ totally but consider it as a less preferred answer set because Alice being a non-smoker is still a conceivable perception.

The tolerance partition $\mathcal{T}(\mathcal{P})$ of an ASP program $\mathcal{P}$ can be used to initialize the approach from (Brewka and Eiter 1999) as well. In general, this leads to results different from our approach, though. For instance, the answer set $\mathcal{S}_1$ of $\mathcal{P}_{\mathsf{smk}}$ would be preferred over $\mathcal{S}_2$ in contrast to Example 9.

In (Brewka and Eiter 1999) two principles are proposed as minimal requirements to well-behaved prioritization in rule-based systems such as answer set programming. Because the second principle is subject of controversial discussion in literature (Rintanen 1998), we focus on the first one here and prove that our approach satisfies this principle.

Proposition 3 (Principle I (cf. (Brewka and Eiter 1999)))**.** *Let $\mathcal{P}$ be a ground ASP program, and let $\mathcal{S}_1$ and $\mathcal{S}_2$ be different answer sets of $\mathcal{P}$ with $\mathcal{G}_{\mathcal{P}}(\mathcal{S}_i) = \mathcal{P}' \dot\cup \{r_i\}$ for $i = 1, 2$. If $\rho_{\mathcal{P}}(r_1) > \rho_{\mathcal{P}}(r_2)$, then S_2 is not a most preferred answer set of $\mathcal{P}$ (actually, S_1 is preferred over S_2).*

Proof. Let $\mathcal{S}$ be the set of head literals of $\mathcal{P}'$, let $i \in \{1, 2\}$, and let H_i be the head literal of r_i. Then, $\mathcal{S}_i = \mathcal{S} \cup \{H_i\}$ holds. Because answer sets of the same program can not include each other and $\mathcal{S}_1 \neq \mathcal{S}_2$ holds, $H_i \notin \mathcal{S}_i$ follows. As a consequence, r_i is the only rule in $\mathcal{G}_{\mathcal{P}}(\mathcal{S}_i)$ with head literal H_i. Thus, r_i is in every generating set of S_i and, for every (inclusion minimal) generating set $\mathcal{P}'' \cup \{r_1\}$ of S_1, $\mathcal{P}'' \cup \{r_2\}$ is a minimal generating set of S_2. With $\rho_{\mathcal{P}}(r_1) > \rho_{\mathcal{P}}(r_2)$ it follows that $\rho_{\mathcal{P}}(\mathcal{P}'' \cup \{r_1\}) > \rho_{\mathcal{P}}(\mathcal{P}'' \cup \{r_2\})$ (mind the monotony of mean) such that S_1 is preferred over S_2 and S_2 can not be a most preferred answer set of $\mathcal{P}$. □

Therewith, our approach proves to satisfy all of our expectations to prioritized ASP. The approach in (Brewka and Eiter 1999) satisfies Principle I as well but on the downside there are consistent ASP programs where no answer set is preferred which seems quite unintuitive. Thus, Brewka and Eiter proposed an alternative approach to preferring answer sets which always leads to so-called *weakly preferred answer sets* but this approach does not satisfy Principle I.

Discussion and Conclusion

We adapted the notion of tolerance from conditional reasoning to answer set programming (ASP) in order to intrinsically prioritize answer sets. For this, we assigned ranks to the rules in an ASP program $\mathcal{P}$ based on their occurrence in a specific tolerance partition of $\mathcal{P}$. Basically, the higher its rank the more specific a rule is, and an answer set is preferred over the remaining answer sets if it is built upon more specific rules (on average). A benefit of our approach is that it does not alter the semantics of ASP programs. The answer sets of a program can be calculated with any ASP solver, like `clingo` (Gebser et al. 2019), and the prioritization of these answer sets is a subsequent task. Also, our approach does not calculate most preferred answer sets only, like the approach in (Brewka and Eiter 1999), but provides a preference ordering on all answer sets. Our approach is a first attempt on intrinsic prioritization in ASP and can be extended in multiple ways. Instead of combining ASP with System Z, it is conceivable that other ranking formalisms from conditional reasoning like *c-representations* (Kern-Isberner 2004) and *lexicographic entailment* (Lehmann 1995) can be used to prioritize answer sets. In future work, we want to develop inference operators based on prioritized answer sets and examine their inference quality which allows conclusions about the quality of the prioritization as well.

Acknowledgments. This work was supported by Grant KE 1413/14-1 of the German Research Foundation (DFG) awarded to Gabriele Kern-Isberner.

References

Baral, C. 2010. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Brewka, G., and Eiter, T. 1999. Preferred answer sets for extended logic programs. *Artif. Intell.* 109(1-2):297–356.

Brewka, G.; Delgrande, J. P.; Romero, J.; and Schaub, T. 2015. asprin: Customizing answer set preferences without a headache. In Bonet, B., and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, 1467–1474. AAAI Press.

Clark, K. 1977. Negation as failure. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977*, Advances in Data Base Theory, 293–322. New York: Plemum Press.

Delgrande, J.; Schaub, T.; and Tompits, H. 2000. Logic programs with compiled preferences. In Horn, W., ed., *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20-25, 2000*, 464–468. IOS Press.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory Pract. Log. Program.* 19(1):27–82.

Gebser, M.; Kaminski, R.; and Schaub, T. 2011. Complex optimization in answer set programming. *Theory and Practice of Logic Programming* 11(4-5):821–839.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R. A., and Bowen, K. A., eds., *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, 1070–1080. MIT Press.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Gener. Comput.* 9(3/4):365–386.

Gelfond, M. 2008. Answer sets. In van Harmelen, F.; Lifschitz, V.; and Porter, B., eds., *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*. Elsevier. 285–316.

Kern-Isberner, G. 2004. A thorough axiomatization of a principle of conditional preservation in belief revision. *Ann. Math. Artif. Intell.* 40(1-2):127–164.

Kraus, S.; Lehmann, D.; and Magidor, M. 1990. Nonmonotonic reasoning, preferential models and cumulative logics. *Artif. Intell.* 44(1-2):167–207.

Lehmann, D. 1995. Another perspective on default reasoning. *Ann. Math. Artif. Intell.* 15(1):61–82.

Nieuwenborgh, D. V., and Vermeir, D. 2006. Preferred answer sets for ordered logic programs. *Theory and Practice of Logic Programming* 6(1-2):107–167.

Pearl, J. 1990. System Z: A natural ordering of defaults with tractable applications to nonmonotonic reasoning. *Probabilistic and Causal Inference*.

Reiter, R. 1980. A logic for default reasoning. *Artif. Intell.* 13(1-2):81–132.

Rintanen, J. 1998. Lexicographic priorities in default logic. *Artif. Intell.* 106(2):221–265.

Sakama, C., and Inoue, K. 2000. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence* 123(1-2):185–222.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138(1-2):181–234.

Spohn, W. 2014. *The Laws of Belief - Ranking Theory and Its Philosophical Applications*. Oxford University Press.

Thevapalan, A.; Wilhelm, M.; Kern-Isberner, G.; Kaiser, P.; and Roidl, M. 2023. An interactive modelling environment for designing warehouse layouts based on ASP. In Franklin, M., and Chun, S. A., eds., *Proceedings of the Thirty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2023, Clearwater Beach, FL, USA, May 14-17, 2023*. AAAI Press.

Wang, K.; Zhou, L.; and Lin, F. 2000. Alternating fixpoint theory for logic programs with priority. In Lloyd, J.; Dahl, V.; Furbach, U.; Kerber, M.; Lau, K.-K.; Palamidessi, C.; Pereira, L. M.; Sagiv, Y.; and Stuckey, P., eds., *Computational Logic - CL 2000, First International Conference, London, UK, 24-28 July, 2000, Proceedings*, volume 1861 of *LNCS*, 164–178. Springer.

Wilhelm, M.; Thevapalan, A.; and Kern-Isberner, G. 2023. Prioritizing answer sets based on conditional expert knowledge. In Franklin, M., and Chun, S. A., eds., *Proceedings of the Thirty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2023, Clearwater Beach, FL, USA, May 14-17, 2023*. AAAI Press.

Zhang, Z. 2018. Introspecting preferences in answer set programming. In Palù, A. D.; Tarau, P.; Saeedloei, N.; and Fodor, P., eds., *Technical Communications of the 34th International Conference on Logic Programming, ICLP 2018, July 14-17, 2018, Oxford, United Kingdom*, volume 64 of *OASIcs*, 3:1–3:13. Dagstuhl.