Faceted Answer-Set Navigation*

Christian Alrabbaa[1], Sebastian Rudolph[2], and Lukas Schweizer[2]

[1] Institute of Theoretical Computer Science
[2] Institute of Artificial Intelligence
TU Dresden, Dresden, Germany
firstname.lastname@tu-dresden.de

Abstract. Even for small logic programs, the number of resulting answer-sets can be tremendous. In such cases, users might be incapable of comprehending the space of answer-sets as a whole nor being able to identify a specific answer-set according to their needs. To overcome this difficulty, we propose a general formal framework that takes an arbitrary logic program as input, and allows for navigating the space of answer-sets in a systematic interactive way analogous to faceted browsing. The navigation is carried out stepwise, where each step narrows down the remaining solutions, eventually arriving at a single one. We formulate two navigation modes, one stringent conflict avoiding, and a "free" mode, where conflicting selections of facets might occur. For the latter mode, we provide efficient algorithms for resolving the conflicts. We provide an implementation of our approach and demonstrate that our framework is able to handle logic programs for which it is currently infeasible to retrieve all answer sets.

1 Introduction

Answer-set programming (ASP) is a well-known declarative rule-based programming paradigm, developed for knowledge-representation and problem solving [9,13,3]. It originates from the field of logic programming (thus its syntactic relationship with Prolog) and the field of non-monotonic reasoning. ASP has become popular as generic language for computationally hard problems; that is, problems are encoded as logic programs (rules) and evaluated under the answer-set semantics, such that each answer-set (logical model) yields a solution of the problem. For solvers, such as `Clingo` [8,7] and `DLV` [4], the logic programming community is actively improving usability and interoperability. For example, IDEs have been developed that support users in the same way as it is known for other programming languages [5], or tools able to visualize answer-sets [12]. More recently, both tools were enriched with APIs for a more seamless integration in other programming languages.

However, even if the developed answer-set program behaves in the desired way, i.e. gives rise to all solutions (and only those), it might yield by design a large number of answer-sets, which a user might simply not be able to handle and overview. Even worse, it might even not be possible to compute all solutions in reasonable time. For example,

* We would like to thank the anonymous reviewers for their valuable feedback. This work is partially supported by the German Research Foundation (DFG) within the Research Training Group QuantLA (GRK 1763) and within the Collaborative Research Center SFB 912 – HAEC.

consider the simple encoding of the N-Queens problem in Example 1; it already has 92 solutions for $n = 8$, 724 for $n = 10$, and 14200 solutions for $n = 12$, all computed in just a few seconds; however, $n = 27$ is the largest N-Queens instance for which the number of solutions is known [16]. Answer-sets of a program might be of course parsed, inspected, and visualized individually in some specific way, but this is apparently limited to a small set of answer-sets only and always specific for some application domain.

Certainly, one can (to some extent) formulate a problem in such a way that it becomes an optimization problem, thereby only optimal models are returned. This might suffice in some cases to obtain a limited number of answer-sets, but even then, there might be more optimal models than any user is able to consume.

Our motivation in this paper therefore is to provide a formal framework for *exploring the space of answer-sets* of a logic program in a systematic way by means of a navigation method similar to *faceted browsing*. A facet can be seen as a partial solution that, if activated, is enforced to be present in any answer-set. Intuitively, a successive selection of facets narrows down the remaining answer-sets as well as available facets, thus leading to a manageable amount, or ultimately to one single remaining answer-set. In Example 1, any answer-set consists exactly of those n atoms `q(X,Y)`, where a queen is placed at row `X` and column `Y`. Now any of these atoms can act as facet; e.g. under an active facet `q(1,1)`, 4 answer-sets out of 92 are left, and selecting any other facet then yields a unique solution. In contrast to this goal-oriented navigation, we also define *free navigation* where facets can be in conflict with each other. Considering Example 1, the facet `q(1,4)` would not be available anymore, though enforcing it would require to retract the previous activation of facet `q(1,1)`. We call this retraction of facets *corrections* and provide an efficient approach to compute them, enabling an even more powerful answer-set navigation.

Example 1. Consider the following program Π_1, that is a concise and efficient encoding of the well-known N-Queens problem, taken from [9].

```
(1)  #const n = 8.
(2)  { q(I,1..n) } == 1 :- I = 1..n.
(3)  { q(1..n,J) } == 1 :- J = 1..n.
(4)  :- { q(D-J,J) } >= 2, D = 2..2*n.
(5)  :- { q(D+J,J) } >= 2, D = 1-n..n-1.
```

While the first line fixes the value of constant n, each remaining line corresponds to constraints ensuring exactly one queen per column (2), row (3) and diagonals (4 – 5).

1.1 Related Work

Our framework can be seen as additional layer on top of an ASP solver and is defined purely on the resulting answer-sets of a given program, thus the logic program itself may remain syntactically unknown. In contrast to our motivation, other approaches exist, designed to *debug* an answer-set program in case it does not behave as expected [15]; i.e. one would like to get an answer to the question why a given answer-set is actually not an answer-set of the presumably faulty program, or why some ground atom is not

present in any of the answer-sets. In the stepwise approach of Oetsch et al. [15], one starts with a partial interpretation and debugs the program step by step, where a step is an application of a rule acquired from the original program. This is different from our notion of a navigation step, which narrows down remaining answer-sets by adding a constraint rule. In other words, in the suggested navigation scenario, the user's task is to steer the exploration of the answer-set space towards the desired answer-set, which from the beginning on is an answer set, unlike in the debugging setting where the desired answer-set is most likely not an answer-set of the initial program.

Moreover, in databases several approaches have been developed to compute so-called *repairs* [11,1]. In short, a repair denotes the set of tuples that need to be added or retracted from a database instance in order to make it consistent w.r.t. to a set of constraints. Thus, again the initial situation is an inconsistent artifact (here the database) that needs to be repaired somehow in order to regain consistency. This is again in contrast to our approach, since from the beginning we constantly retain consistency; i.e. the initial program is never in an inconsistent state.

We want to emphasize, that our approach is not dedicated to a specific application domain, and generally applicable to arbitrary answer-set programs of any application domain. For example, the framework can also be seen as an engine for product configuration; i.e. the underlying program resembles a specification of a product, where answer-sets then represent configurations of that product. Our contributions in this paper are:

1. A formalization of faceted navigation in the space of answer-sets, based on the notion of brave and cautious consequences.
2. An extended navigational concept, allowing for arbitrary navigation directions and resolving resulting conflicts.
3. An implementation that demonstrates feasibility of the framework even for programs where the total number of answer-sets remains unknown.

2 Answer-Set Programming

We review the basic notions of answer-set programming [14], for further details we refer to [2,6,10].

We fix a countable set $\mathcal{U}$ of *(domain) elements*, also called *constants*; and suppose a total order $<$ over the domain elements. An *atom* is an expression $p(t_1, \ldots, t_n)$, where p is a *predicate* of arity $n \geq 0$ and each t_i is either a variable or an element from $\mathcal{U}$. An atom is *ground* if it is free of variables. $B_\mathcal{U}$ denotes the set of all ground atoms over $\mathcal{U}$. A *(disjunctive) rule* ρ is of the form

$$a_1, \ldots, a_n \leftarrow b_1, \ldots, b_k, \ not\ b_{k+1}, \ldots, \ not\ b_m.$$

with $m \geq k \geq 0$, where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms, and "*not*" denotes *default negation*. The *head* of ρ is the set $H(\rho) = \{a_1, \ldots, a_n\}$ and the *body* of ρ is $B(\rho) = \{b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m\}$. Furthermore, $B^+(\rho) = \{b_1, \ldots, b_k\}$ and $B^-(\rho) = \{b_{k+1}, \ldots, b_m\}$. A rule ρ is *safe* if each variable in ρ occurs in $B^+(r)$. A rule ρ is *ground* if no variable occurs in ρ. A *fact* is a ground rule with empty body. An

(input) database is a set of facts. A (disjunctive) *program* is a finite set of disjunctive rules. For a program Π and an input database D, we often write $\Pi(D)$ instead of $D \cup \Pi$. For any program Π, let U_Π be the set of all constants appearing in Π. $Gr(\Pi)$ is the set of rules $\rho\sigma$ obtained by applying, to each rule $\rho \in \Pi$, all possible substitutions σ from the variables in ρ to elements of U_Π.

An *interpretation* $I \subseteq B_\mathcal{U}$ *satisfies* a ground rule ρ iff $H(\rho) \cap I \neq \emptyset$ whenever $B^+(\rho) \subseteq I$, $B^-(\rho) \cap I = \emptyset$. I satisfies a ground program Π, if each $\rho \in \Pi$ is satisfied by I. A non-ground rule ρ (resp., a program Π) is satisfied by an interpretation I iff I satisfies all groundings of ρ (resp., $Gr(\Pi)$). $I \subseteq B_\mathcal{U}$ is an *answer set* (also called *stable model*) of Π iff it is a subset-minimal set satisfying the *Gelfond-Lifschitz reduct* $\Pi^I = \{H(\rho) \leftarrow B^+(\rho) \mid I \cap B^-(\rho) = \emptyset, \rho \in Gr(\Pi)\}$. For a program Π, we denote the set of its answer sets by $\mathcal{AS}(\Pi)$.

We make use of further syntactic extensions, namely integrity constraints and count expressions, which both can be recast to ordinary normal rules as described in [6]. An *integrity constraint* is a rule ρ where $H(\rho) = \emptyset$, intuitively representing an undesirable situation; i.e. it has to be avoided that $B(\rho)$ evaluates positively. Count expressions are of the form $\#count\{l : l_1, \ldots, l_i\} \bowtie u$, where l is an atom and $l_j = p_j$ or $l_j = not\, p_j$, for p_j an atom, $1 \leq j \leq i$, u a non-negative integer, and $\bowtie \in \{\leq, <, =, >, \geq\}$. The expression $\{l : l_1, \ldots, l_n\}$ denotes the set of all ground instantiations of l, governed through $\{l_1, \ldots, l_n\}$. We restrict the occurrence of count expressions in a rule ρ to $B^+(\rho)$ only. Intuitively, an interpretation satisfies a count expression, if $N \bowtie u$ holds, where N is the cardinality of the set of ground instantiations of l, $N = |\{l \mid l_1, \ldots, l_n\}|$, for $\bowtie \in \{\leq, <, =, >, \geq\}$ and u a non-negative integer.

Consequences We rely on two notions of consequence, given a program Π and an atom α, we say that Π cautiously entails α, written $\Pi \models_\forall \alpha$, if for every answer-set $S \in \mathcal{AS}(\Pi)$, $\alpha \in S$. Likewise, we say that Π bravely entails α, written $\Pi \models_\exists \alpha$, if there exists and answer-set $S \in \mathcal{AS}(\Pi)$, such that $\alpha \in S$. The set of all cautious consequences of Π is denoted $\mathcal{CC}(\Pi)$ and the set of its brave consequences $\mathcal{BC}(\Pi)$.

3 Faceted Navigation

We distinguish two different modes of faceted navigation for a logic program. In the first one, the facets that can be applied are the ones that are compatible with those previously selected; this is what we call *restricted navigation*. Conversely, in the *free navigation* mode, we drop this restriction and describe a technique that resolves conflicts that can occur due to the unrestricted application of facets.

3.1 Facets and Navigation Step

We first start by defining *facets* of a program, before we introduce the notion *navigation step* as basic navigational building block. If not mentioned otherwise, we use the term *program* to refer to disjunctive programs as introduced in Section 2.

Definition 1 (Facet). *Let Π be a disjunctive logic program. Then we denote with $\mathcal{F}^+(\Pi) = \mathcal{BC}(\Pi) \setminus \mathcal{CC}(\Pi)$, the set of inclusive facets, and with $\mathcal{F}^-(\Pi) = \{\overline{p(\bar{t})} \mid$*

$p(\bar{t}) \in \mathcal{F}^+(\Pi)\}$, *the set of exclusive facets. With* $\mathcal{F}(\Pi) = \mathcal{F}^+(\Pi) \cup \mathcal{F}^-(\Pi)$, *we denote the set of all facets applicable to* Π. *We say an answer-set* S *of* Π *satisfies an inclusive facet* $p(\bar{t}) \in \mathcal{F}^+(\Pi)$ *if* $p(\bar{t}) \in S$. *It satisfies an exclusive facet* $\overline{p(\bar{t})} \in \mathcal{F}^-(\Pi)$ *if* $p(\bar{t}) \notin S$.

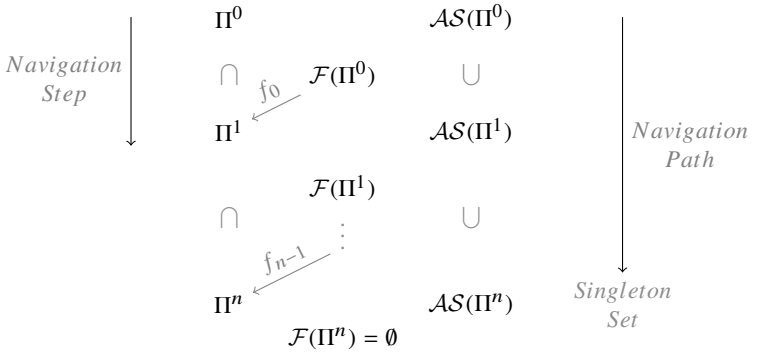

Fig. 1. The interplay of an initial program Π^0, its answer-sets $\mathcal{AS}(\Pi^0)$ and its corresponding facets $\mathcal{F}(\Pi^0)$, for n navigation steps.

With the notion of a facet at hand, we can now start to define the navigation as a sequence of single navigation steps, defined in the following.

Definition 2. *Given a set of facets* $\mathcal{F}$, *we define the function* ic *that rewrites every* $f \in \mathcal{F}$ *into a corresponding integrity constraint. Formally,* $ic(f) = \leftarrow neg(f)$, *with*

$$neg(f) = \begin{cases} not\ p(\bar{t}) & \text{if } f = p(\bar{t}). \\ p(\bar{t}) & \text{if } f = \overline{p(\bar{t})}. \end{cases}$$

We let $ic(\mathcal{F}) \coloneqq \{ic(f) \mid f \in \mathcal{F}\}$.

Definition 3 (Navigation Step). *A navigation step, written* $\Pi \overset{f}{\Longrightarrow} \Pi'$, *is a transition from one program* Π *to another program* Π', *where* Π' *is obtained by adding the integrity constraint* $ic(f)$ *to* Π, *where* $f \in \mathcal{F}(\Pi)$.

Faceted navigation of Π is possible as long as $\mathcal{BC}(\Pi) \setminus \mathcal{CC}(\Pi) \neq \emptyset$.

Example 2. Consider the following program Π_2:

$$b, a \leftarrow \quad d, e \leftarrow \quad c \leftarrow a$$

Which has the following answer-sets $\mathcal{AS}(\Pi_2) = \{\{b, e\}, \{b, d\}, \{a, e, c\}, \{a, d, c\}\}$, and consequently the facets $\mathcal{F}(\Pi_2) = \{a, b, e, d, c, \bar{a}, \bar{b}, \bar{e}, \bar{d}, \bar{c}\}$. Let $f = a \in \mathcal{F}^+(\Pi)$, then applying $\Pi_2 \overset{a}{\Longrightarrow} \Pi_2'$ yields the answer-sets $\mathcal{AS}(\Pi_2') = \{\{a, e, c\}, \{a, d, c\}\}$ and facets $\mathcal{F}(\Pi_2') = \{e, d, \bar{e}, \bar{d}\}$. Continuing, we apply $\Pi_2' \overset{\bar{e}}{\Longrightarrow} \Pi_2''$, resulting $\mathcal{AS}(\Pi_2'') = \{\{a, d, c\}\}$, and $\mathcal{F}(\Pi_2'') = \emptyset$, thus not allowing any further navigation step.

The following theorem establishes that performing one navigation step by applying a facet has exactly the desired consequence of including or excluding the respective atom from the space of considered answer-sets.

Theorem 1. *Let* Π' *be the program obtained from* Π *by applying one navigation step using some facet* $f \in \mathcal{F}(\Pi)$*, i.e.* $\Pi \overset{f}{\Longrightarrow} \Pi'$*. Then* $\mathcal{AS}(\Pi') = \{S \in \mathcal{AS}(\Pi) \mid S \text{ satisfies } f\}$.

Proof. For the $\supseteq$ direction, let $S \in \mathcal{AS}(\Pi)$ and S satisfy f. Then $\Pi^S = \Pi'^S$ (since $ic(f)$ is entirely removed from the Gelfond-Lifschitz reduct of Π' wrt. S). Therefore, by assumption, S is a subset-minimal set satisfying Π'^S and hence $S \in \mathcal{AS}(\Pi')$.

For the $\subseteq$ direction, let $S \in \mathcal{AS}(\Pi')$. If S would not satisfy f, the Gelfond-Lifschitz reduct Π'^S would contain the rule "$\leftarrow$" due to $ic(f)$, and therefore be unsatisfiable, resulting in $S \notin \mathcal{AS}(\Pi')$, contradicting our assumption. Hence S must satisfy f, resulting in $\Pi^S = \Pi'^S$ as above, therefore $S \in \mathcal{AS}(\Pi')$ implies $S \in \mathcal{AS}(\Pi)$. □

Note that this theorem also entails $\mathcal{AS}(\Pi') \subsetneq \mathcal{AS}(\Pi)$ due to the definition of $\mathcal{F}(\Pi)$.

Lemma 1. *Given a logic program* Π*, applying a navigation step* ($\Pi \overset{f}{\Longrightarrow} \Pi'$) *will never cause the generation of an unsatisfiable logic program* Π'.

Proof. By definition, $f \in \mathcal{F}(\Pi)$ means that there is some $p(\bar{t}) \in \mathcal{BC}(\Pi) \setminus \mathcal{CC}(\Pi)$. From $p(\bar{t}) \in \mathcal{BC}(\Pi)$ immediately follows that there is some $S \in \mathcal{AS}(\Pi)$ with $p(\bar{t}) \in S$. On the other hand, from $p(\bar{t}) \notin \mathcal{CC}(\Pi)$ follows that there is some $S' \in \mathcal{AS}(\Pi)$ with $p(\bar{t}) \notin S$. Therefore, applying Theorem 1, either $S \in \mathcal{AS}(\Pi')$ (if f is inclusive) or $S' \in \mathcal{AS}(\Pi')$ (if f is exclusive). Hence, Π' is satisfiable in any case. □

Figure 1 sketches a sequence of navigation steps, and depicts the correlation of the resulting programs, answer-sets and facets.

3.2 Free Navigation Mode

In the previous setting, the assumption was that only facets of the current program are available to be applied in the next step. This ensures a conflict free navigation eventually resulting in a single answer-set. This might be convenient in some situations, however, we intend to relax this stringent process and extend the faceted navigation approach and allow the application of arbitrary facets from the initial program in any step. This inherently leads to conflicts; i.e. would cause an unsatisfiable program if two or more facets in combination enforce solutions that do not exist.

Therefore, when applying a facet that would lead to an unsatisfiable program, we aim to pick facets applied in previous steps that necessarily need to be retracted, in order to apply the desired facet. We start by identifying all facets that have already been applied.

Definition 4 (Active Facets). *For a program* Π^n*, obtained after an application of* n *navigation steps, i.e.* $\Pi^0 \overset{f_0}{\Longrightarrow} \ldots \overset{f_{n-1}}{\Longrightarrow} \Pi^n$*, we denote with* $\mathcal{F}_a(\Pi^n)$ *the set of facets active in* Π^n*, i.e.* $\mathcal{F}_a(\Pi^n) = \{f_0, \ldots, f_{n-1}\}$.

Amongst these active facets, some might be incompatible with some facet f from the initial program. We now aim to identify those facets that need to be retracted in order to be able to apply $f-$ and call such a set *correction set*.

Definition 5 (Correction Set). *Let $f \in \mathcal{F}(\Pi^0)$ be the facet chosen to be applied next to a program Π^n, but $f \notin \mathcal{F}(\Pi^n)$. A set $K \subseteq \mathcal{F}_a(\Pi^n)$, is a correction set of Π^n w.r.t. f, if $\Pi^n \setminus ic(K) \cup \{ic(f)\}$ is satisfiable. We denote by $\mathcal{K}(\Pi^n)$ the set of all correction sets of Π^n w.r.t. f.*

Example 3. Continuing with $\Pi_2 \overset{a}{\Longrightarrow} \Pi_2' \overset{\bar{e}}{\Longrightarrow} \Pi_2''$ from Example 2, where $\mathcal{F}(\Pi_2) = \{a, b, e, d, c, \bar{a}, \bar{b}, \bar{e}, \bar{d}, \bar{c}\}$ and $\mathcal{F}_a(\Pi_2'') = \{a, \bar{e}\}$. For $b \in \mathcal{F}(\Pi_2)$ we therefore have the correction sets $\{a\}$ and $\{a, \bar{e}\}$ for Π_2''.

It is now of interest to highlight how correction sets can be computed. As it turns out, this can be done by a program obtained from the original one by adding some extra rules.

Definition 6. *Given a navigation path of length n, $\Pi^0 \overset{f_0}{\Longrightarrow} \ldots \overset{f_{n-1}}{\Longrightarrow} \Pi^n$, a facet $f \in \mathcal{F}(\Pi^0) \setminus \mathcal{F}(\Pi^n)$. Then the program $\Pi^{\mathcal{K}}$ is defined as:*

$$\Pi^{\mathcal{K}} := \Pi^0 \cup \{ic(f)\} \cup \{remove(i) \leftarrow neg(f_i) \mid f_i \in \mathcal{F}_a(\Pi^n) \text{ for all } 0 \leq i < n\}$$

where "remove" is a new predicate name.

Lemma 2. *For each $S \in \mathcal{AS}(\Pi^{\mathcal{K}})$ there exists a correction set K such that*

$$K = \{f_i \in \mathcal{F}_a(\Pi^n) \mid remove(i) \in S, 0 \leq i < n\}.$$

Proof. Let $S \in \mathcal{AS}(\Pi^{\mathcal{K}})$. First observe, that S certainly contains at least one (ground) atom $remove(i)$. Since $\mathcal{F}_a(\Pi^n) \neq \emptyset$, we construct a non-empty set E, s.t. for all $0 \leq i < n$

$$\begin{aligned} E := \{f_i \in \mathcal{F}_a(\Pi^n) \mid f_i \notin S, remove(i) \in S\} \cup \\ \{\bar{f}_i \in \mathcal{F}_a(\Pi^n) \mid f_i \in S, remove(i) \in S\}, \end{aligned}$$

and find $E \subseteq \mathcal{F}_a(\Pi^n)$. It remains to show that, $\Pi^\star := (\Pi^n \setminus ic(E) \cup \{ic(f)\})$ is satisfiable. Because of the definition of E and $\Pi^{\mathcal{K}}$, every $f_i \in E$ corresponds to some $remove(i)$ in some $S \in \mathcal{AS}(\Pi^{\mathcal{K}})$ where for every S we have $f \in S$. Which means that for every $f_i \in E$, the body of the corresponding rule in $\Pi^{\mathcal{K}}$, namely $remove(i) \leftarrow neg(f_i)$, must evaluate to true, hence f_i (whether inclusive or exclusive) to false. This is equivalent to the removal of $ic(f_i)$ from Π^n for all $f_i \in E$. Knowing that $\Pi^{\mathcal{K}}$ is satisfiable, moreover $\Pi^0 \cup \{ic(f)\}$ is satisfiable, then $\Pi^\star$ must be satisfiable as well. □

Lemma 3. *Given a navigation path $\Pi^0 \overset{f_0}{\Longrightarrow} \ldots \overset{f_{n-1}}{\Longrightarrow} \Pi^n$, a facet $f \in \mathcal{F}(\Pi^0)$, but $f \notin \mathcal{F}(\Pi^n)$, and the program $\Pi^{\mathcal{K}}$. Then for every correction set $K \in \mathcal{K}(\Pi^n)$ w.r.t. f, there exists an answer-set $S \in \mathcal{AS}(\Pi^{\mathcal{K}})$ induced by K.*

Proof. For arbitrary $K \in \mathcal{K}(\Pi^n)$, let $\bar{K} \coloneqq \mathcal{F}_a(\Pi^n) \setminus K$ be of those facets that can safely remain in Π^n; i.e. $\Pi' = \Pi^0 \cup \{ic(f)\} \cup ic(\bar{K})$ is satisfiable. Consequently, the program

$$\Pi'' \coloneqq \Pi' \cup \{remove(i) \leftarrow neg(f_i) \mid f_i \in K\}$$

is also satisfiable, thus let $S \in \mathcal{AS}(\Pi'')$. In particular, $\{remove(i) \mid f_i \in K\} \subset S$. Notice that Π'' is similar to $\Pi^{\mathcal{K}}$, except that $\Pi^{\mathcal{K}}$ does not contain the integrity constraints $ic(\bar{K})$, but instead the corresponding rules with a "$remove(i)$" atom in the head. Due to the construction of Π'' and the restrictions imposed by the integrity constraints $ic(\bar{K})$, we find that $\mathcal{AS}(\Pi'') \subseteq \mathcal{AS}(\Pi^{\mathcal{K}})$ and thus $S \in \mathcal{AS}(\Pi^{\mathcal{K}})$. □

3.3 Preferred Correction Sets

Inherently, there are potentially many correction sets, though it is natural to prefer correction sets for which most of the active facets can be retained; in other words, we would like to obtain correction sets that contain only facets that necessarily need to be retracted. We therefore introduce two preference notions.

Definition 7 (Minimal Correction Set). *A correction set K for some program Π w.r.t. to some facet f is minimal, if there exist no other correction set K', s.t. $K' \subset K$. Then $\mathcal{K}_{\subseteq}(\Pi) = \{K \mid K \text{ is minimal}\}$, is the set of all minimal correction sets for some program Π w.r.t. to some facet f.*

Amongst all minimal correction sets, which might vary in size, we now further restrict the preference to those that are also cardinality minimal.

Definition 8 (Small Correction Set). *A correction set K for some program Π w.r.t. to some facet f is small, if there exist no other correction set K', s.t. $|K'| < |K|$. Then $\mathcal{K}_{|<|}(\Pi) = \{K \mid K \text{ is small}\}$, is the set of the smallest correction sets for some program Π w.r.t. to some facet f.*

Amongst the correction sets from Example 3, only $\{a\}$ is minimal as well as the smallest. In order to find all correction sets for some program Π^n w.r.t. some facet $f \in \mathcal{F}(\Pi^0) \setminus \mathcal{F}(\Pi^n)$, the program $\Pi^{\mathcal{K}}$ needs to be constructed, from which then the correction sets are extracted from the answer-sets. In general, $|\mathcal{AS}(\Pi^{\mathcal{K}})|$ is quite large which hinders the computation of all correction sets; and is another practical reason for preferences over correction sets. However, since finding minimal correction sets is also computationally expensive, we propose an incremental computation of $\mathcal{K}_{\subseteq}(\Pi^n)$, starting from $\mathcal{K}_{|<|}(\Pi^n)$. The advantage that small correction sets have over the minimal ones is that once we find the size of a small correction set we do not need to investigate further. This does not apply for the minimal correction sets which they might differ in size. In order to have a feasible way of computing $\mathcal{K}_{\subseteq}(\Pi^n)$, we can not simply use $\Pi^{\mathcal{K}}$ as defined earlier (without modification), to generate all correction sets and then filter out the minimal ones, because we would run into problems when it comes to $\Pi^{\mathcal{K}}$s with a large number of answer-sets. Therefore, it is shown next how to generate $\mathcal{K}_{|<|}(\Pi^n)$, from which $\mathcal{K}_{\subseteq}(\Pi^n)$ can be computed.

Small Correction Sets. Given Π^0 and Π^n where $\Pi^0 \overset{f_0}{\Longrightarrow} \ldots \overset{f_{n-1}}{\Longrightarrow} \Pi^n$ and a facet $f \in \mathcal{F}(\Pi^0) \setminus \mathcal{F}(\Pi^n)$; $\mathcal{K}_{|<|}(\Pi^n)$ w.r.t. f can be computed via the program $\Pi^{\mathcal{K}}$ as mentioned earlier, then extending it with the following rule, $\varrho_i = \leftarrow not\ \#count\ \{Y : remove(Y)\}\ i$, where $0 < i \leq n$. Basically, ϱ_i enforce to have at most i *remove* atoms present in an answer-set. Since $f \in (\mathcal{F}(\Pi^0) \setminus \mathcal{F}(\Pi^n))$, it is guaranteed that there exists no answer-set $S \in \mathcal{AS}(\Pi^{\mathcal{K}})$ where S does not contain any atom with "*remove*" as predicate name, and thus it can not be the case that $i = 0$. Therefore, the solver would be called at most (worst case) n times and the answer-sets of the first satisfiable $\Pi^{\mathcal{K}}$ will contain the answer-sets that provide all small correction sets of Π^n w.r.t. f. The refinement of this rule plus the creation of $\Pi^{\mathcal{K}}$ is shown in the first algorithm in the next section.

Minimal Correction Sets. Based on $\mathcal{K}_{|<|}(\Pi^n)$ w.r.t. some f, $\mathcal{K}_{\subseteq}(\Pi^n)$ w.r.t. f can be computed. But first, we need to introduce a variation of the function $ic(\mathcal{F})$, namely $cic(\mathfrak{F})$, where $\mathfrak{F}$ is a set of sets of facets, defined as:

$$cic(\mathfrak{F}) = \bigcup_{\mathcal{F} \in \mathfrak{F}} \{ \leftarrow neg(f_0), \ldots, neg(f_n) \mid \mathcal{F} = \{f_0, \ldots, f_n\}\}.$$

Initially, we describe the computation of $\mathcal{K}_{\subseteq}(\Pi^n)$ informally. Let $|K| = d$ where $K \in \mathcal{K}_{|<|}(\Pi^n)$ w.r.t. some $f \in \mathcal{F}(\Pi^0)$ but $f \notin \mathcal{F}(\Pi^n)$. For every small K, we add an integrity constraint of the form "$cic(K)$" to $\Pi^{\mathcal{K}}$ (thus obtaining $\Pi^{\mathcal{K}'}$), then for all $S \in \mathcal{AS}(\Pi^{\mathcal{K}})$ that lead to the generation of some small correction set $K \in \mathcal{K}_{|<|}(\Pi^n)$, we have $S \notin \mathcal{AS}(\Pi^{\mathcal{K}'})$. If $\mathcal{AS}(\Pi^{\mathcal{K}'}) \neq \emptyset$, then there must exist some answer-set $S' \in \mathcal{AS}(\Pi^{\mathcal{K}'})$ s.t. $K' = \{f_i \in \mathcal{F}_a(\Pi^n) \mid remove(i) \in S', 0 \leq i < n\}$. For the small correction sets K' we know that $K' \not\subseteq K$ for any $K \in \mathcal{K}_{|<|}(\Pi^n)$ and since $\mathcal{AS}(\Pi^{\mathcal{K}'}) \subset \mathcal{AS}(\Pi^{\mathcal{K}})$ – follows from Theorem 1 – we conclude that $K' \in \mathcal{K}_{\subseteq}(\Pi^n)$. In the following lemma, we adjust the notation of a correction set of a program Π^n w.r.t. some f by adding a superscript, namely K^{d+j}, where $d + j$ is the size of K, d is the size of the small correction sets and $0 \leq j < n$.

Lemma 4. *Let d be the size of a small correction set of some Π^n w.r.t. some f; let K^{d+j} be a correction set of Π^n w.r.t. the same f and obtained from some $S \in \mathcal{AS}(\Pi^{\mathcal{K}})$ such that $K^{d+j} \in \mathcal{K}_{\subseteq}(\Pi^n)$. We define $\Pi^{\mathcal{K}'} = \bigcup_{0 \leq l < d+j} cic(\mathcal{K}^l_{\subseteq}(\Pi^n)) \cup \Pi^{\mathcal{K}} \cup \{\varrho_{d+j}\}$; if $\mathcal{AS}(\Pi^{\mathcal{K}'}) \neq \emptyset$, then the following holds: For every $K^{d+j} \in \mathcal{K}_{\subseteq}(\Pi^n)$, there must exist some $S' \in \mathcal{AS}(\Pi^{\mathcal{K}'})$ that induces some K' s.t. $K' = K^{d+j}$.*

Proof. By the definition of $\Pi^{\mathcal{K}'}$, we can conclude that for every $S' \in \mathcal{AS}(\Pi^{\mathcal{K}'})$ and every set of "$remove(\_)$" atoms that correspond to some K^l we have that

$$\{remove(i) \mid f_i \in K^l\} \not\subseteq S'$$

We also know that $\mathcal{AS}(\Pi^{\mathcal{K}'}) \subset \mathcal{AS}(\Pi^{\mathcal{K}})$ – which follows from Theorem 1 but we do not prove it – then for all $S' \in \mathcal{AS}(\Pi^{\mathcal{K}'})$ we have $S' \in AS(\Pi^{\mathcal{K}})$ and for every K' obtained from some S' we have $S' = S \in \mathcal{AS}(\Pi^{\mathcal{K}})$ and $|K'| = d + j$ (Because of the rule ϱ_{d+j}). Which means that $K' = K^{d+j}$. □

Therefore, in order to compute the set $\mathcal{K}_{\subseteq}(\Pi^n)$, we first compute $\mathcal{K}_{|<|}(\Pi^n)$, create the program $\Pi^{\mathcal{K}'} = cic(\mathcal{K}_{|<|}(\Pi^n)) \cup \Pi^{\mathcal{K}}$. If $\mathcal{AS}(\Pi^{\mathcal{K}'}) \neq \emptyset$, then we update $\Pi^{\mathcal{K}'}$ by adding the rule ϱ_i that enforces a restriction on the size of the correction sets obtained from $S' \in \mathcal{AS}(\Pi^{\mathcal{K}'})$, i.e. $\Pi^{\mathcal{K}'} = \Pi^{\mathcal{K}'} \cup \{\varrho_{d+j}\}$ where $j = 1$. If the updated $\Pi^{\mathcal{K}'}$ is satisfiable, then we can extract all minimal correction sets of Π^n with the size $d + 1$. If it is not the case, we increase the value of j by one, and we keep on increasing the value of j until we reach the first satisfiable version of the updated $\Pi^{\mathcal{K}'}$. Since the initial condition is $\mathcal{AS}(\Pi^{\mathcal{K}'}) \neq \emptyset$ ($\Pi^{\mathcal{K}'}$ before adding ϱ_i), then we know that there must exist a value j where $\Pi^{\mathcal{K}'} \cup \{\varrho_{d+j}\}$ is actually satisfiable. We keep on repeating the whole process until we reach a program $\Pi^{\mathcal{K}^*}$ s.t. $\mathcal{AS}(\Pi^{\mathcal{K}^*}) = \emptyset$, hence computing $\mathcal{K}_{\subseteq}(\Pi^n)$. In the worst case scenario, the solver would be called less than $2 \times n$ times. The following example depict a case where the *free navigation* mode is applied.

Example 4. Let Π be a logic program defined as follows.

$$\{a; b; c; d\} \qquad\qquad e \leftarrow b$$
$$\leftarrow e, d, a \qquad\qquad e \leftarrow c$$

Since $\mathcal{F}(\Pi) = \{a, b, c, d, \bar{a}, \bar{b}, \bar{c}, \bar{d}\}$, we apply $\Pi \overset{a}{\Longrightarrow} \Pi^1$. Thus $\Pi^1 = \Pi \cup \{ic(a)\}$, and we obtain $\mathcal{F}(\Pi^1) = \{b, c, d, \bar{b}, \bar{c}, \bar{d}\}$. Applying $\Pi^1 \overset{b}{\Longrightarrow} \Pi^2$ yields $\Pi^2 = \Pi^1 \cup \{ic(b)\}$. With $\mathcal{F}(\Pi^2) = \{c, d, \bar{c}, \bar{d}\}$, we apply $\Pi^2 \overset{c}{\Longrightarrow} \Pi^3$, and obtain $\Pi^3 = \Pi^2 \cup \{ic(c)\}$. At this stage of the faceted navigation, $\mathcal{F}(\Pi^3) = \emptyset$; for $d \in \mathcal{F}(\Pi)$, which is an inactive facet, we have $d \notin \mathcal{F}(\Pi^3)$ and therefore $\Pi^3 \overset{d}{\Longrightarrow} \Pi^4$ can not be applied. We compute $\mathcal{K}_{\subseteq}(\Pi^3)$ w.r.t. d as follows.

1. Generate $\Pi^{\mathcal{K}} = \Pi \cup \overbrace{\{r(1) \leftarrow not\, a,\ r(2) \leftarrow not\, b,\ r(3) \leftarrow not\, c\}}^{\mathcal{R}} \cup \{ic(d)\}$.
2. Extend $\Pi^{\mathcal{K}}$ with ϱ_1, i.e. $\Pi^{\mathcal{K}} = \Pi^{\mathcal{K}} \cup \{ \leftarrow not\, \#count\, \{Y : r(Y)\}\, 1\}$.
3. $\mathcal{AS}(\Pi^{\mathcal{K}}) = \{\{b, c, d, e, r(1)\}\}$; moreover $\mathcal{K}_{|<|}(\Pi^3) = \{\{a\}\}$.
4. Compute $\Pi^{\mathcal{K}'} = \Pi \cup \mathcal{R} \cup \{ic(d)\} \cup cic(\{\{a\}\})$.
5. Since $\mathcal{AS}(\Pi^{\mathcal{K}'}) \neq \emptyset$, there must exist some $S' \in \mathcal{AS}(\Pi^{\mathcal{K}'})$ that leads to some correction sets of Π^3 w.r.t. d.
6. Extend $\Pi^{\mathcal{K}'}$ with ϱ_2, i.e. $\Pi^{\mathcal{K}'} = \Pi^{\mathcal{K}'} \cup \{ \leftarrow not\, \#count\, \{Y : r(Y)\}\, 2\}$.
7. $\mathcal{AS}(\Pi^{\mathcal{K}'}) = \{\{a, d, r(2), r(3)\}\}$; moreover $\mathcal{K}_{\subseteq}(\Pi^3) = \mathcal{K}_{|<|}(\Pi^3) \cup \{\{b, c\}\}$.
8. At this stage, we can terminate the search for more minimal correction sets of Π^n w.r.t. d because $\Pi^{\mathcal{K}''} = \Pi \cup \mathcal{R} \cup \{ic(d)\} \cup cic(\{\{a\}\}) \cup cic(\{\{b, c\}\})$ is unsatisfiable; which means all minimal correction sets have been computed.

4 Implementation and Evaluation

The following algorithm describes the computation of the set of all minimal correction sets of some program Π^n where $\Pi^0 \overset{f_0}{\Longrightarrow} \ldots \overset{f_{n-1}}{\Longrightarrow} \Pi^n$.

Algorithm 1: *ASP-Based Minimal Correction Sets Generator*

Data: $\mathcal{F}_a(\Pi^n)$, f, Π^0
$Seq \coloneqq list(\mathcal{F}_a(\Pi^n)); \mathcal{K}_{\subseteq}(\Pi^n) \coloneqq \emptyset; j \coloneqq 1;$
$\Pi^{\mathcal{K}} \coloneqq \Pi^0 + ic(f);$
$counter \coloneqq$ ":- *not* #*count* $\{Y : remove(Y)\}$" $+ str(j) +$ ".";

for $(i \; in \; 0 \ldots n)$ **do**
  $\Pi^{\mathcal{K}}$+= "$remove($" $+ str(i) +$ "$)$" $+ ic(Seq[i]);$
$\Pi^{\mathcal{K}}$+= $counter;$

while $(j \leq n)$ **do**
  $Corrections = solve(\Pi^{\mathcal{K}});$
  $\mathcal{K}_{\subseteq}(\Pi^n).addAll(Corrections);$
  j++ ;
  if $(Corrections \neq \emptyset)$ **then**
    $\Pi^{\mathcal{K}}$+= $cic(Corrections);$
    $updatePi(counter, n);$ `// replace in counter: j -> n`
    if $(!sat(\Pi^{\mathcal{K}}))$ **then**
      $break;$
  $updatePi(counter, j);$
return $\mathcal{K}_{\subseteq}(\Pi^n)$

As mentioned in the previous section, the size of the correction sets varies, and usually it is not clear what would the effect of applying a correction set be (In particular, what implicit consequential facets would be lost when a certain correction set is applied); therefore, our framework provides an additional functionality that computes the consequences of removing a certain set of facets $F \subseteq \mathcal{F}_a(\Pi^n)$ from Π^n. This computation is described in the following algorithm.

Algorithm 2: *Facets Removal Consequences*

Data: Π^n, F
$Constraints = \emptyset;$
for $f \; in$ F **do**
  $Constraints.add(ic(f));$
$\Pi^{tmp} = \Pi^n \setminus Constraints;$
return $\mathcal{CC}(\Pi^n) \setminus (\mathcal{CC}(\Pi^{tmp}) \cup F)$

Based on what has been introduced in this paper, we implemented a tool called INCA (Interactive General Configuration using Facets) that takes a logic program as an input

and interactively applies the faceted navigation technique in both its *restricted* and *free* modes. INCA is programmed in Python 2.7 and uses Clingo 5.2, as the backbone reasoner and it is publicly available on Github[3]. It must be pointed out that the brave and cautious consequences of the input program are acquired via one of the built-in features of Clingo, which computes these consequences in an efficient reliable manner.

As already indicated in the introduction, the number of all solutions for the n-queens problem with $n > 27$ is still unknown [16]; therefore in the following table we display some performance results of INCA where different instances of the n-queens problem are plugged-in as an input.

Table 1. INCA performance for 8/30 – Queens.

	8-Queens	30-Queens
$\mathcal{F}(\Pi^0)$	0.011 sec	1.054 sec
$\Pi^0 \overset{f_0}{\Longrightarrow} \Pi^1$	0.016 sec	6.935 sec
$\mathcal{K}_{\subseteq}(\Pi^n)$	(n = 4) 1 set in 0.028 sec	(n = 17) 1 set in 0.091 sec
	(n = 4) 3 sets in 0.017 sec	(n = 17) 16 sets in 0.211 sec
	(n = 4) 4 sets in 0.021 sec	(n = 17) 18 sets in 0.185 sec

Table 2. INCA performance for 40/50 – Queens.

	40-Queens	50-Queens
$\mathcal{F}(\Pi^0)$	2.651 sec	5.229 sec
$\Pi^0 \overset{f_0}{\Longrightarrow} \Pi^1$	35.462 sec	2:13.356 min
$\mathcal{K}_{\subseteq}(\Pi^n)$	(n = 26) 145 sets in 1.188 sec	(n = 33) 60 sets in 1.760 sec
	(n = 26) 214 sets in 1.510 sec	(n = 33) 92 sets in 2.121 sec
	(n = 26) 284 sets in 2.040 sec	(n = 33) 114 sets in 3.695 sec

The first row shows the time spent on calculating all facets of the program, the second indicates the time needed to perform the first navigation step, whereas every entry in the third row displays the total number of correction sets of some program Π^n w.r.t different facets f (mostly the exclusive version of some consequential facets) where $f \in \mathcal{F}(\Pi^0)$ and $f \notin \mathcal{F}(\Pi^n)$. Another problem, where the calculation of all possible answer-sets is computationally expensive, is Sudoku. INCA takes approximately 0.186 seconds to compute all facets of an empty 9×9 Sudoku. The time needed to apply the first navigation step is approximately 2.930 seconds. The following shows the time spent to calculate different instances of the set $\mathcal{K}_{\subseteq}(\Pi^{40})$ with distinctive sizes depending on different facets f where $f \in \mathcal{F}(\Pi^0)$ and $f \notin \mathcal{F}(\Pi^{40})$.

[3] https://github.com/lukeswissman/inca

Table 3. Computation of all minimal correction sets of 9×9 Sudoku for different facets.

$\lvert\mathcal{K}_{\subseteq}(\Pi^{40})\rvert = 497$	$\lvert\mathcal{K}_{\subseteq}(\Pi^{40})\rvert = 713$	$\lvert\mathcal{K}_{\subseteq}(\Pi^{40})\rvert = 2499$	$\lvert\mathcal{K}_{\subseteq}(\Pi^{40})\rvert = 4013$
3.559 sec	4.650 sec	47.122 sec	2:10.874 min

5 Conclusion

We provide a general formal way to overcome the situation of answer-set programs with an unmanageable amount of answer-sets, where users might either be incapable of comprehending the space of answer-sets as a whole, or being able to identify a specific answer-set according to their needs. Our approach is therefore based on faceted navigation, wherein a facet can be seen as a partial solution that, if activated, is enforced to be present in any answer-set. This can be realized as stringent sequential process succeeding in a single answer-set, but also as navigation mode allowing unrestricted application of facets, potentially causing conflicts; i.e. *conflicting facets*. We provide a pure answer-set programming encoding to resolve conflicts, which besides giving a certain degree of freedom in navigation also contributes to the overall experience of answer-set exploration.

Regarding future work, we would like to extend the approach to support aggregate facets supporting to restrict the remaining answer-sets in terms of some numerical value. The same idea can be applied to preferences, which could be enriched with a cost value, depending on the underlying application scenario.

It also remains imperative to explore the possibilities of enriching the user interface, e.g. by an aggregated presentation of atoms; unary ground atoms can be represented as (drop-down) lists. We see huge potential to explore answer-sets visually, where methods for visualizing large data-sets, established by other disciplines, could be reused. On the other hand our implementation can be used as backend to realize domain specific user interfaces. For example, product configurators, where users interactively can configure parts and properties of some product, being assisted in case of conflicting selections.

References

1. Arenas, M., Bertossi, L.E., Chomicki, J.: Specifying and querying database repairs using logic programs with exceptions. In: Larsen, H.L., Andreasen, T., Christiansen, H., Kacprzyk, J., Zadrozny, S. (eds.) Flexible Query Answering Systems - Recent Advances Proceedings of the Fourth International Conference on Flexible Query Answering Systems, FQAS 2000, October 25-28, 2000. Advances in Soft Computing, vol. 7, pp. 27–41. Physica-Verlag Heidelberg New York, A Springer-Verlag Company (2000)
2. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM 54(12), 92–103 (2011)
3. Eiter, T., Ianni, G., Krennwallner, T.: Answer set programming: A primer. In: Reasoning Web. Semantic Technologies for Information Systems. vol. 5689, pp. 40–110 (2009)
4. Eiter, T., Leone, N., Mateis, C., Pfeifer, G., Scarcello, F.: The KR system dlv: Progress report, comparisons and benchmarks. In: Cohn, A.G., Schubert, L.K., Shapiro, S.C. (eds.)

Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998. pp. 406–417. Morgan Kaufmann (1998)
5. Febbraro, O., Reale, K., Ricca, F.: ASPIDE: integrated development environment for answer set programming. In: Delgrande, J.P., Faber, W. (eds.) Proceedings of Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, May 16-19, 2011. Lecture Notes in Computer Science, vol. 6645, pp. 317–330. Springer (2011)
6. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning 6, 1–238 (2012)
7. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Clingo = ASP + control: Preliminary report. CoRR abs/1405.3694 (2014)
8. Gebser, M., Kaminski, R., König, A., Schaub, T.: Advances in *gringo* series 3. In: Delgrande, J.P., Faber, W. (eds.) Proceedings of Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, May 16-19, 2011. Lecture Notes in Computer Science, vol. 6645, pp. 345–351. Springer (2011)
9. Gebser, M., Roland Kaminski, B.K., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan and Claypool Publishers (2012)
10. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. 9(3/4), 365–386 (1991)
11. Gertz, M.: Diagnosis and repair of constraint violations in database systems. Datenbank Rundbrief 19, 96 (1997)
12. Kloimüllner, C., Oetsch, J., Pührer, J., Tompits, H.: Kara: A system for visualising and visual editing of interpretations for answer-set programs. In: Tompits, H., Abreu, S., Oetsch, J., Pührer, J., Seipel, D., Umeda, M., Wolf, A. (eds.) Applications of Declarative Programming and Knowledge Management - 19th International Conference, INAP 2011, and 25th Workshop on Logic Programming, WLP 2011, September 28-30, 2011, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7773, pp. 325–344. Springer (2011)
13. Lifschitz, V.: What is answer set programming? In: Fox, D., Gomes, C.P. (eds.) Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, July 13-17, 2008. pp. 1594–1597. AAAI Press (2008)
14. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Ann. Math. Artif. Intell. 25(3-4), 241–273 (1999)
15. Oetsch, J., Pührer, J., Tompits, H.: Stepwise debugging of answer-set programs. TPLP 18(1), 30–80 (2018)
16. Preußer, T.B., Engelhardt, M.R.: Putting Queens in Carry Chains, No27. Signal Processing Systems 88(2), 185–201 (2017)