Abstraction for Non-Ground Answer Set Programs [⋆]

Zeynep G. Saribatur, Peter Schüller, and Thomas Eiter

Institute of Logic and Computation, TU Wien

Abstract. We address the issue of abstraction, a widely used notion to simplify problems, in the context of Answer Set programming (ASP), which is a highly expressive formalism and a convenient tool for declarative problem solving. We introduce a method to automatically abstract non-ground ASP programs given an abstraction over the domain, which ensures that each original answer set is mapped to some abstract answer set. We discuss abstraction possibilities on several examples and show the use of abstraction to gain insight into problem instances, e.g., domain details irrelevant for problem solving; this makes abstraction attractive for getting to the essence of the problem. We also provide a tool implementing automatic abstraction from an input program.

1 Introduction

Abstraction is an approach that is widely used in Computer Science and AI to simplify problems, cf. [8, 22, 2, 16, 14]. By omitting details, complex scenarios are reduced to ones that are easier to deal with and to understand; in fact, abstraction is ubiquitous in building models of reality, which approximate the latter to meet specific application purposes.

Surprisingly, abstraction has not been considered much in the context of nonmonotonic knowledge representation and reasoning, and specifically not in Answer Set Programming (ASP) [7]. Simplification methods such as equivalence-based rewriting [12, 25], partial evaluation [6, 20], or forgetting (cf. [23] for a recent survey), have been extensively studied. However, they strive for preserving the semantics, while abstraction may change it and lead to an over-approximation of the models (answer sets) of a program, in a modified language.

Recently, Saribatur and Eiter [27] presented such an approach that omits atoms from an ASP program, similar in spirit to abstraction in planning problems [18]. The approach is propositional in nature and does not account for the fact that in ASP, non-ground rules talk about a domain of discourse; e.g., a rule

$$col(X,r) \leftarrow node(X), not\ col(X,g), not\ col(X,b).$$

may express that a node X in a graph must be colored red, if it is not colored green nor blue; or the rule

⋆ We thank the reviewers for their feedback. This work has been supported by Austrian Science Fund (FWF) project W1255-N23 and Austrian Federal Ministry of Transport Innovation and Technology (BMVIT) project 861263 (DynaCon).

$$\{moveToTable(B, A, T)\} \leftarrow on(B, B_1, T), free(B, T)$$

that the block B on top of a stack may at time T be moved to a table area A. For the (non)existence of an answer set, the precise set of elements (nodes resp. blocks and areas) may not matter, but rather how certain elements are related; for that, some elements may be abstracted into single elements. Then, a coloring of the abstracted graph, if exists, may be refined to the original graph; if not, the latter is not colorable. Similarly, a plan for a blocksworld problem with abstract areas may be turned into a concrete one by instantiating them.

It is unexplored how for a non-ground ASP program Π, given an abstraction over its domain, a suitable abstract program Π' can be automatically constructed and evaluated. We tackle this issue and make the following contributions.

• We introduce the notion of domain abstraction for ASP programs. For that, an abstraction of domain elements for a program Π is supplied with an abstract program Π' such that each answer set of Π maps to an abstract answer set of Π'.

• We provide a method to automatically construct such an abstract program Π'. It works modularly on the syntactic level, by constructing for each rule abstract rules with a similar structure, where uncertainty caused by the abstracted domain is carefully respected.

• We show how abstract answer sets can be computed and further processed. This includes a concreteness check, with possible output of an answer set of the original program, and a refinement strategy to deal with spurious answer sets using local search. The whole approach is implemented in a tool that provides automatic abstraction from an input program.

• We consider the domain abstraction approach for several examples, where we also discuss how to use it for subdomains (sorts) such as time, and how to compose sort abstractions. An experimental evaluation shows the potential of the approach for various applications.

2 Domain Abstraction for ASP

ASP. We adopt as a function-free first order language, in which a logic program Π is a finite set of rules r of the form $\alpha \leftarrow B(r)$ where α is an atom and the body $B(r) = l_1, \ldots, l_n$ is a set of positive and negative literals l_i of the form β or *not* β, respectively, where β is an atom and *not* is default negation; $B^+(r)$ and $B^-(r)$ are the sets of all positive resp. negative literals in $B(r)$. A rule r resp. program Π is ground, if it is variable-free, and r is a *fact* if moreover $n = 0$. Rules r with variables stand for the sets $grd(r)$ of their ground instances, and semantically Π induces a set $AS(\Pi)$ of stable models (or answer sets) [15] which are Herbrand models (i.e., sets I of ground atoms) of Π justified by the rules, in that I is a $\subseteq$-minimal model of $f\Pi^I = \{r \in grd(\Pi) \mid I \models B(r)\}$ [11], where $grd(\Pi) = \bigcup_{r \in \Pi} grd(r)$. A program Π is *unsatisfiable*, if $AS(\Pi) = \emptyset$. A common syntactic extension are *choice rules* of the form $\{\alpha\} \leftarrow B$, which stands for the rules $\alpha \leftarrow B, not\ \alpha'$ and $\alpha' \leftarrow B, not\, \alpha$, where α' is a fresh atom.

To illustrate various challenges of abstraction we use the following example.

Example 1 (Running example) *Consider the following example program Π with domain predicate int for an integer domain $D = \{0, \dots, 5\}$.*

$$c(X) \leftarrow not\ d(X), X < 5, int(X). \tag{1}$$
$$d(X) \leftarrow not\ c(X), int(X). \tag{2}$$
$$b(X,Y) \leftarrow a(X), d(Y), int(X), int(Y). \tag{3}$$
$$e(X) \leftarrow c(X), a(Y), X \leq Y, int(X), int(Y). \tag{4}$$
$$\leftarrow b(X,Y), e(X), int(X), int(Y). \tag{5}$$

We also have facts $a(1)$, $a(3)$, $int(0), \dots, int(5)$.

Abstraction. A generic notion of abstraction is as follows.

Definition 1 *Given ground programs Π and Π' on sets $\mathcal{A}$ and $\mathcal{A}'$ of atoms, respectively, where $|\mathcal{A}| \geq |\mathcal{A}'|$, Π' is an* abstraction *of Π, if a mapping $m : \mathcal{A} \to \mathcal{A}'$ exists s.t. for each $I \in AS(\Pi)$, $I' = \{m(a) \mid a \in I\}$ is an answer set of Π'.*

We refer to m as an *abstraction mapping.* This notion aims at the grounding (propositional) view of programs. In this paper, we take a first-order view in which $\mathcal{A}$ is the Herbrand base of Π, which results from the available predicate symbols and the constants symbols (the domain D of discourse, i.e., the Herbrand universe), which are by default those occurring in Π. *Domain abstraction* induces abstraction mappings in which constants are merged.

Definition 2 *Given a domain D of Π, a* domain (abstraction) mapping *is a function $m : D \to \widehat{D}$ for a set $\widehat{D}$ (the* abstracted domain*) with $|\widehat{D}| \leq |D|$.*

Thus, a domain mapping divides D into *clusters* of elements $\{d \in D \mid m(d) = \hat{d}\}$, where $\hat{d} \in \widehat{D}$, seen as equal; if unambiguous, we also write $\hat{d}$ for its cluster $m^{-1}(\hat{d})$.

Example 2 (ctd) *A possible abstraction mapping for Π with $\hat{D}_1 = \{k_1, k_2, k_3\}$ clusters $1, 2, 3$ to the element k_1 and 4 and 5 to singleton clusters, i.e., $m_1 = \{\{1, 2, 3\}/k_1, \{4\}/k_2, \{5\}/k_3\}$. A naive mapping is $m_2 = \{\{1, .., 5\}/k\}$ with $\hat{D}_2 = \{k\}$.*

Each domain mapping m naturally extends to ground atoms $a = p(v_1, \dots, v_n)$ by $m(a) = p(m(v_1), \dots, m(v_n))$. To obtain for a program Π and a Herbrand base $\mathcal{A}$ an induced abstraction mapping $m : \mathcal{A} \to \mathcal{A}'$ where $\mathcal{A}' = m(\mathcal{A}) = \{m(a) \mid a \in \mathcal{A}\}$, we need a program Π' as in Definition 1. However, simply applying m to Π does not work. Moreover, we want domain abstraction for non-ground Π that results in a non-ground program Π'. Building a suitable Π' turns out to be challenging and needs to solve several issues, which we gradually address in the next section.

3 Towards an Abstract Program

Handling built-ins and (in)equalities. Original rules may rely on certain *built-in relations* involving variables, such as $<, \leq$ in (1) and (4), or $=$ and $\neq$. The idea is to lift the rules by lifting these relations and dealing with the uncertainty caused by the domain clustering.

Example 3 (ctd) *We abstract from Π using m_2. The rule (3) has no built-in relation and thus it is lifted with no change:*

$$b(X,Y) \leftarrow a(X), d(Y), \widehat{int}(X), \widehat{int}(Y);$$

however, lifting rule (4) simply to

$$e(X) \leftarrow c(X), a(Y), X \leq Y, \widehat{int}(X), \widehat{int}(Y).$$

does not work, as $X \leq Y$ behaves differently over the cluster k. As $k \leq k$, whenever $c(k)$ and $a(k)$ holds the rule derives $e(k)$. This applies, e.g., to the abstraction of $I = \{a(1),\ a(3), c(4), d(0), \ldots, d(3)\}$, where (4) derives no e-atom as $4 \not\leq 3$ and $4 \not\leq 1$. However, I is an answer set of Π and must not be lost in the abstraction. Thus, when a cluster causes uncertainties over the built-ins, we modify the above rule to

$$\{e(X)\} \leftarrow c(X), a(Y), X \leq Y, \widehat{int}(X), \widehat{int}(Y).$$

which allows to skip $e(k)$ even if $c(k)$ and $a(k)$ holds.

Negation. A naive abstraction approach is to turn all rule heads into choices. However, negative literals or certain built-ins (e.g., $\neq, <$) may cause a loss of original answer sets in the abstraction.

Example 4 (ctd) *We change in (4) the symbol $\leq$ to $\neq$ and consider*

$$\{e(X)\} \leftarrow c(X), a(Y), X \neq Y, \widehat{int}(X), \widehat{int}(Y).$$

As $k = k$, the abstract body is never satisfied and $e(k)$ is never derived. However, Π has answer sets containing $c(2)$,$a(3)$ and thus also $e(2)$, as $2 \neq 3$; they are all lost. Adding an additional choice rule with a flipped relation, $X = Y$, is able to catch such cases.

Similarly, let us change in (4) $a(Y)$ to not $a(Y)$. When the rule is lifted to the abstract rule

$$\{e(X)\} \leftarrow c(X), \mathit{not}\ a(Y), X \leq Y, \widehat{int}(X), \widehat{int}(Y)$$

$e(k)$ is not derived as $a(k)$ holds and originally a holds only for 1 and 3. Thus original answer sets I may contain $e(2)$ or $e(4)$; such I exist, and they are lost. Such cases can be caught with additional rules

$$\{e(X)\} \leftarrow c(X), a(Y), X \leq Y, \widehat{int}(X), \widehat{int}(Y).$$

Constraints. Naively lifting the constraints to the abstract rules would result in losing answer sets for the non-singleton domain clusters.

Example 5 (ctd) *If the constraint (5) is lifted with no change, then $b(k,k)$ and $e(k)$ would never occur in the abstract answer sets, while in the original program answer sets can contain $b(x_1, y)$ and $c(x_2)$ as long as $x_1 \neq x_2$.*

In conclusion, only creating choices is not enough to preserve all original answer sets; we need a fine-grained, systematic approach to deal with uncertainties by abstraction.

Lifted built-in relations. As shown before, built-in relations need special treatment, and so do multiple usages of a variable in a rule. To unify both issues, our approach focuses on rules of form $r : l \leftarrow B(r), \mathit{\Gamma}_{rel}(r)$ where the variables in $B(r)$ are standardized apart and $\mathit{\Gamma}_{rel}$ consists of built-in atoms that impose restrictions on the variables in $B(r)$.

Example 6 (ctd) *The rule* (3) *has* $\mathit{\Gamma}_{rel}(r) = \top$ *while the rule* (5) *needs to be standardized apart by rewriting it into* $\leftarrow b(X, Y), e(X_1), \mathit{\Gamma}_{rel}$ *with* $\mathit{\Gamma}_{rel} = (X = X_1)$.

The uncertainty by abstraction may occur due to relation restrictions over non-singleton clusters (i.e., $|\widehat{d}| > 1$) or negative literals mapped to non-singleton abstract literals.

For simplicity, we first focus on binary built-ins, e.g., $=, <, \leq, \neq$, and a $\mathit{\Gamma}_{rel}(r)$ of the form $\mathit{rel}(X, c)$ or $\mathit{rel}(X, Y)$. When the relation rel is lifted to the abstract domain, the following cases (or types) τ_{I}–τ_{IV} for $\mathit{rel}(\hat{d}_1, \hat{d}_2)$ occur in a mapping:

$$\tau^{rel}_{\mathrm{I}}(\hat{d}_1, \hat{d}_2): \quad \mathit{rel}(\hat{d}_1, \hat{d}_2) \wedge \forall x_1 \in \hat{d}_1, \forall x_2 \in \hat{d}_2.\, \mathit{rel}(x_1, x_2)$$
$$\tau^{rel}_{\mathrm{II}}(\hat{d}_1, \hat{d}_2): \neg\mathit{rel}(\hat{d}_1, \hat{d}_2) \wedge \forall x_1 \in \hat{d}_1, \forall x_2 \in \hat{d}_2.\, \neg\mathit{rel}(x_1, x_2)$$
$$\tau^{rel}_{\mathrm{III}}(\hat{d}_1, \hat{d}_2): \quad \mathit{rel}(\hat{d}_1, \hat{d}_2) \wedge \exists x_1 \in \hat{d}_1, \exists x_2 \in \hat{d}_2.\, \neg\mathit{rel}(x_1, x_2)$$
$$\tau^{rel}_{\mathrm{IV}}(\hat{d}_1, \hat{d}_2): \neg\mathit{rel}(\hat{d}_1, \hat{d}_2) \wedge \exists x_1 \in \hat{d}_1, \exists x_2 \in \hat{d}_2.\, \mathit{rel}(x_1, x_2)$$

If $\mathit{rel}(\hat{d}_1, \hat{d}_2)$ holds for some $\hat{d}_1, \hat{d}_2 \in \widehat{D}$, type III is more common in domain abstractions with clusters, while type I occurs for singleton mappings (i.e., $|\hat{d}_1| = |\hat{d}_2| = 1$) or for relations such as $\neq, <$.

Example 7 *Consider a mapping* $m = \{\{1\}/k_1,\ \{2, 3\}/k_2,\ \{4, 5\}/k_3\}$. *For the relation "$=$", $k_1 = k_1$ holds and for any $x_1, x_2 \in k_1 = \{1\}$, $x_1 = x_2$ holds and type I applies. In contrast, $k_2 = k_2$ holds while $2, 3 \in k_2$ and $2 \neq 3$; so type III applies. Further, $k_2 < k_3$ holds and for any $x \in k_2 = \{2, 3\}$ and $y \in k_3 = \{4, 5\}$, we have $x < y$ and so type I applies.*

If $\mathit{rel}(\hat{d}_1, \hat{d}_2)$ does not hold for some $\hat{d}_1, \hat{d}_2 \in \widehat{D}$, type II is common, e.g., $=, \leq$, whereas type IV may occur for $\neq, <$.

Example 8 (ctd) *Reconsider m. Then $k_2 \neq k_2$ does not hold while $k_2 = \{2, 3\}$ has different elements $2 \neq 3$ (type IV). Moreover, $k_1 = k_2$ does not hold in $\widehat{D}$ nor does $x = y$ for every $x \in k_1 = \{1\}$ and $y \in k_2 = \{2, 3\}$ (type II).*

For an abstraction m, we let $\mathcal{T}_m$ be the set of all atoms $\tau^{rel}_{\iota}(\hat{d}_1, \hat{d}_2)$ where $\iota \in \{\mathrm{I}, \ldots, \mathrm{IV}\}$ is the type of the built-in instance $\mathit{rel}(\hat{d}_1, \hat{d}_2)$ for m; note that $\mathcal{T}_m$ is easily computed.

4 Abstract Program Construction

By our analysis, the basic idea to construct an abstract program for a program $\mathit{\Pi}$ with a domain mapping m is as follows: we either just abstract each atom in a rule, or in case of uncertainty due to domain abstraction, we guess rule heads to catch possible cases, or we treat negated literals by shifting their polarity depending on the abstract domain clusters.

For simplicity and ease of presentation, we first consider programs Π with rules having (i) at most one negative body literal which shares an argument with the relation, (ii) a single, binary built-in literal and (iii) no cyclic dependencies between non-ground atoms.

For any rule r and $* \in \{+,-\}$, let the set $S^*_{rel}(r) = \{l_j \in B^*(r) \mid arg(l_j) \cap \{t_1, t_2\} \neq \emptyset\}$ be the positive and negative body literals, respectively, that share an argument with $rel(t_1, t_2)$. By assumption (i) we have $B^-(r) \subseteq S^*_{rel}(r)$.

Definition 3 *Given a program Π and a domain mapping m, we construct an abstract program Π^m as follows. For each rule $r \in \Pi$ of form $l \leftarrow B(r), rel(t_1, t_2)$ we add:*

(1) If $S^+_{rel}(r) = \emptyset$: $m(l) \leftarrow m(B(r))$.
(2) If $S^+_{rel}(r) \neq \emptyset$:
  (a) $m(l) \leftarrow m(B(r)), rel(\hat{t}_1, \hat{t}_2), \tau^{rel}_{\mathrm{I}}(\hat{t}_1, \hat{t}_2)$.
  (b) $\{m(l)\} \leftarrow m(B(r)), rel(\hat{t}_1, \hat{t}_2), \tau^{rel}_{\mathrm{III}}(\hat{t}_1, \hat{t}_2)$.
  (c) $\{m(l)\} \leftarrow m(B(r)), \overline{rel}(\hat{t}_1, \hat{t}_2), \tau^{rel}_{\mathrm{IV}}(\hat{t}_1, \hat{t}_2)$.
(3) If $l_i \in S^-_{rel}(r)$:
  (a′) $\{m(l)\} \leftarrow m(B^{sh}_{l_i}(r)), rel(\hat{t}_1, \hat{t}_2), \tau^{rel}_{\mathrm{III}}(\hat{t}_1, \hat{t}_2)$.
  (b′) if $S^+_{rel}(r) = \emptyset$: same as (c),
    $\{m(l)\} \leftarrow m(B^{sh}_{l_i}(r)), \overline{rel}(\hat{t}_1, \hat{t}_2), \tau^{rel}_{\mathrm{IV}}(\hat{t}_1, \hat{t}_2)$.
    $\{m(l)\} \leftarrow m(B^{sh}_{l_i}(r)), rel(\hat{t}_1, \hat{t}_2)$.

where $B^{sh}_{l_i}(r) = B^+(r) \cup \{l_i\}, not\ B^-(r) \backslash \{l_i\}$, $\overline{rel}$ denotes the complement of rel, and for $j \in \{1, 2\}$, if t_j is a constant then $\hat{t}_j = m(t_j)$, else $\hat{t}_j = t_j$, i.e., variables are not mapped.

In case (1), we have positive literals that do not share arguments with rel; this includes $rel{=}\top$ (e.g., (3)). Here the abstraction of r is added. If $B^-(r) \neq \emptyset$, then the assumption on $B^-(r)$ prohibits the case $B^-(r) \backslash S^-_{rel}(r) \neq \emptyset$ (i.e., there is no default negated literal without a respective relation).

In case (2), $rel(t_1, t_2)$ shares arguments with a positive body literal. We add rules to grasp possible effects of the relation type. In case of uncertainty, the head becomes a choice, and for case IV, we flip the relation, $\overline{rel}$, to catch the case of the relation holding true.

The constraints in the program (e.g., (5)) gets omitted in the cases with uncertainty (i.e., case (2-b,c) and case (3)).

Example 9 (ctd) *Consider Ex. 1 with domain mapping $m = \{\{1\}/k_1, \{2, 3\}/k_2, \{4, 5\}/k_3\}$. In rule (4), the relation $X \leq Y$ has $S^+_{\leq}(r) = \{c(X), a(Y)\}$. We have $\tau^{\leq}_{\mathrm{I}}(x, y)$ true for $(x, y) \in \{(k_1, k_1), (k_1, k_2), (k_1, k_3), (k_2, k_3)\}$, and $\tau^{\leq}_{\mathrm{III}}(x, y)$ true for $(x, y) \in \{(k_2, k_2), (k_3, k_3)\}$, and only type* II *for all other tuples (x, y). The abstract rules for (4) are:*

$$e(X) \leftarrow c(X), a(Y), X \leq Y, \tau^{\leq}_{\mathrm{I}}(X, Y), \widehat{int}(X), \widehat{int}(Y).$$
$$\{e(X)\} \leftarrow c(X), a(Y), X \leq Y, \tau^{\leq}_{\mathrm{III}}(X, Y), \widehat{int}(X), \widehat{int}(Y).$$

In case (3) of Definition 3, $rel(t_1, t_2)$ shares arguments with a negative body literal. We grasp the uncertainty arising from negation by adding rules where the related literal is shifted to the positive body via $B^{sh}_{l_i}(r)$. (3-b$'$) deals with the special case of a type IV relation and a negative literal.

Example 10 (ctd) *Rule* (1) *has a negative literal,* $not\, d(X)$*, and the relation* $X < 5$ *with shared argument* X*. When it is lifted to* $X < k_3$*, it has* $\tau^{<}_{\mathrm{II}}(a, b)$ *true for* $(a, b) \in \{(k_3, k_1), (k_3, k_2)\}$*,* $\tau^{<}_{\mathrm{IV}}(k_3, k_3)$*, and type* I *for all other tuples* (a, b)*.*

By case (1), it is abstracted without change for τ_{I} *abstract values, while for* τ_{IV} *specially treated rules are added:*

$$\begin{aligned}
c(X) &\leftarrow not\ d(X), X < k_3, \widehat{int}(X).\\
\{c(X)\} &\leftarrow not\ d(X), X \geq k_3, \tau^{<}_{\mathrm{IV}}(X, k_3), \widehat{int}(X).\\
\{c(X)\} &\leftarrow d(X), X \geq k_3, \tau^{<}_{\mathrm{IV}}(X, k_3), \widehat{int}(X).\\
\{c(X)\} &\leftarrow d(X), X < k_3, \widehat{int}(X).
\end{aligned}$$

The abstract program is now as follows.

Definition 4 *Given a program* Π *and a domain abstraction* m*, the program* Π^m *consists of all non-facts of* Π *rewritten as shown above, facts* $\{x. \mid x \in \mathcal{T}_m\}$ *and facts* $p(\boldsymbol{t})$ *of the original program lifted to abstract facts* $p(m(\boldsymbol{t}))$*.*

Notably, the construction of Π^m is modular, rule by rule.

Theorem 1 *Let* m *be a domain mapping of a program* Π *under the above assumptions (i)–(iii). Then for every* $I \in AS(\Pi)$*,* $m(I) \cup \mathcal{T}_m \in AS(\Pi^m)$*.*

Proof (sketch). The rules added in cases (1),(2a-2b) and (3a$'$) are to ensure that $m(I)$ is a model of Π^m, as either the original rule is kept or it is changed to a choice rule. The rules added in cases (2c) and (3b$'$) serve to catch the cases that may violate the minimality of the model due to a negative literal or a relation over non-singleton clusters.

Abstract Program (General Case). We now describe how to remove the restrictions (i)–(iii) on programs from above.

(i) Multiple negative literals. If rule r has $|B^-(r)|>1$, we shift each negative literal that either (a) shares an argument with the abstracted relation rel, or (b) shares arguments mapped to a non-singleton cluster. Thus, instead of having $B^{sh}_{l}(r)$ for one literal, we consider the shifting of multiple literals at a time $B^{sh}_{L}(r)=B^+(r) \cup L, not\ B^-(r)\backslash L$, and all combinations of (non-)shifting of the literals in $L \in B^-(r)$.

(ii) Multiple relation literals. A simple approach to handle a built-in part $\Gamma_{rel} = rel(t_{1,1}, t_{2,1}), \ldots, rel(t_{1,k}, t_{2,k})$, $k > 1$, is to view it as literal of an $2k$-ary built-in $rel'(X_{1,1}, X_{2,1}, \ldots, X_{1,k}, X_{2,k})$. The abstract version of such rel' and the cases I-IV are readily lifted from x_1, x_2 to $x_1, \ldots, x_n$. E.g., for $\Gamma_{rel} =$ $(X_1 = X_2, X_3 = X_4)$, we use a new relation $rel'(X_1, X_2, X_3, X_4)$. For abstract values $\hat{d}_1, \ldots, \hat{d}_4$ such that $\hat{d}_1 = \hat{d}_2 \wedge \hat{d}_3 = \hat{d}_4$ holds, we have type τ_{I} if all $\hat{d}_i$ are singleton clusters and τ_{III} if some $\hat{d}_i$ is non-singleton; otherwise (i.e., $\overline{rel}'(\hat{d}_1,\hat{d}_2,\hat{d}_3,\hat{d}_4)$ holds) type τ_{II} applies.

(iii) Cyclic dependencies. Rules which are involved in a cyclic dependency containing at least one negation between two literals need special consideration.

Example 11 *Consider the rules* (1)-(2) *(Ex.1) and the mapping* $\{\{1,\ldots,5\}/k\}$. *The abstract rules for them are*

$$\{c(X)\} \leftarrow not\ d(X), X \geq k, \tau^{<}_{\mathrm{IV}}(X,k), \widehat{int}(X).$$
$$\{c(X)\} \leftarrow d(X), X \geq k, \tau^{<}_{\mathrm{IV}}(X,k), \widehat{int}(X). \tag{6}$$
$$\{c(X)\} \leftarrow d(X), X < k, \widehat{int}(X). \tag{7}$$
$$\{d(X)\} \leftarrow c(X), \widehat{int}(X). \tag{8}$$

in addition to the abstracted rules due to case (1). While $\{c(k),\ d(k)\}$ *is a model of the rules, it is not minimal and hence not an answer set. However, the original rules have "choice" answer sets with c- and d-atoms, e.g.,* $I = \{c(0), d(1), c(2), d(3), c(4), d(5)\}$*; they are lost by the abstraction.*

To resolve this, we preprocess the program Π and mark atoms involved in a negative cyclic dependency. Then, in step (3) of Definition 3, we modify $B^{sh}_{l_i}(r)$ to eliminate marked literals l_i instead of shifting their polarity.

Example 12 (ctd) *Instead of* (6)–(8)*, the abstract rules are*

$$\{c(X)\} \leftarrow X \geq k, \tau^{<}_{\mathrm{IV}}(X,k), \widehat{int}(X). \quad \{c(X)\} \leftarrow X < k, \widehat{int}(X). \quad \{d(X)\} \leftarrow \widehat{int}(X).$$

Let Π^m denote the program obtained from a general program Π with the generalized abstraction procedure. Then:

Theorem 2 *Let* m *be a domain mapping of a program* Π. *Then for every* $I \in AS(\Pi)$, $\widehat{I} = m(I) \cup \mathcal{T}_m$ *is an answer set of* Π^m.

Proof (sketch). For (i) and (iii), shifting the polarity of each negative literal related with a non-singleton cluster and omitting the ones that are involved in a negative cycle with the head of the rule ensures that the minimality is preserved. The approach in (ii) is a simple combination of the relations.

Over-approximation. The abstraction yields in general an over-approximation of the answer sets of a program. This motivates the following notion.

Definition 5 *An abstract answer set* $\widehat{I} \in AS(\Pi^m)$ *is* concrete, *if* $\widehat{I} = m(I) \cup \mathcal{T}_m$ *for an* $I \in AS(\Pi)$, *else it is* spurious.

A spurious abstract answer set does not have any corresponding concrete answer set. (Non-)existing spurious answer sets allow us to infer properties of the original program.

Proposition 3 *For any program* Π, *(i) For the identity mapping* $id = \{\{x\}/x \mid x \in D\}$ *we have that*

$$AS(\Pi^{id}) = \{I \cup \mathcal{T}_{id} \mid I \in AS(\Pi)\}.$$

(ii) $AS(\Pi^m) = \emptyset$ *implies that* $AS(\Pi) = \emptyset$.

(iii) $AS(\Pi) = \emptyset$ *iff some* Π^m *has only spurious answer sets.*

Checking spuriousness has the following complexity.

Theorem 4 *Deciding whether an abstract answer set* $\widehat{I} \in AS(\Pi^m)$ *of a program* Π *is spurious is* **NEXP**-*complete in general and* Σ_2^p-*complete for bounded predicate arities.*

That is, the worst case complexity is the one of answer set existence for non-ground programs; the two problems can be reduced to each other in polynomial time. However, it drops to Σ_2^p if the domain size $|D|$ is polynomial in the abstracted domain size $|\widehat{D}|$; e.g., if each abstract cluster is small (and multiple clusters exist). Notably, deciding whether Π^m has some spurious abstract answer set of Π can be shown to be $\mathbf{NEXP}^{\mathbf{NP}}$-complete in general and Σ_3^p-complete for bounded predicate arities (i.e., by a constant). The membership is shown by a guess & check algorithm involving ordinary answer set existence, and the hardness by encoding the evaluation of suitable second-order formulas.

5 Abstract Answer Set Computation

After constructing the abstract program Π^m, we can run an ASP solver to obtain abstract answer sets $\widehat{I}$ for the program Π with the mapping m. We then need to check its concreteness, which can be done as follows.

Concreteness check. Let $Q^m_{\widehat{I}}$ be the following constraints:

$$\bot \leftarrow \{\alpha \mid m(\alpha) = \hat{\alpha}\} \leq 0. \quad \hat{\alpha} \in \widehat{I} \setminus \mathcal{T}_m \tag{9}$$

$$\bot \leftarrow \alpha. \quad \hat{\alpha} \notin \widehat{I} \setminus \mathcal{T}_m, m(\alpha) = \hat{\alpha} \tag{10}$$

Here (9) ensures that a witnessing answer set I of Π contains for every non-τ_ι, abstract atom in $\widehat{I}$ some atom that is mapped to it. The constraint (10) ensures that I has no atom that is mapped to an abstract atom not in $\widehat{I}$. We then obtain:

Proposition 5 $\widehat{I}$ *is spurious iff* $\Pi \cup Q^m_{\widehat{I}}$ *is unsatisfiable.*

Refining Abstractions. After checking an abstract answer set, one can either continue finding other abstract answer sets and check their correctness, or *refine* the abstraction to reach an abstraction where less spurious answer sets occur.

Definition 6 *Given a domain mapping* $m : D \to D'$, *a mapping* $m' : D \to D''$, *where* $D'' \neq D'$, *is a* refinement *of* m *if for all* $x \in D$, $m'^{-1}(m'(x)) \subseteq m^{-1}(m(x))$.

Refinement is on dividing the abstract clusters to a finer grained domain.

Example 13 *Consider* $m = \{\{1\}/k_1, \{2,3\}/k_2, \{4,5\}/k_3\}$. *The mapping* $m' = \{\{1\}/k_1, \{2\}/k_{2,1}, \{3\}/k_{2,2}, \{4,5\}/k_3\}$ *is a refinement of* m *since for* $x \in \{2,3\}$, *we have that* $m'^{-1}(m'(x)) = \{x\} \subseteq \{2,3\} = m^{-1}(m(x))$, *while for* $x \in \{1,4,5\}$, *we have that* $m'^{-1}(m'(x)) = m^{-1}(m(x))$.

5.1 Implementation

We have implemented the workflow above in a tool[1], that uses Python and Clingo 5 [13]. We next discuss practical implementation issues.

Concreteness check. We use a non-ground version of $Q^{m}_{\widehat{I}}$:

$$\begin{aligned}&\bot \leftarrow in(\hat{\alpha}), \{\alpha : map(X_1, \hat{X}_1), \ldots, map(X_k, \hat{X}_k)\} \leq 0.\\ &\bot \leftarrow \alpha, not\ in(\hat{\alpha}), map(X_1, \hat{X}_1), \ldots, map(X_k, \hat{X}_k)\end{aligned}$$

where $\alpha = p(X_1, ..., X_k)$ and $\hat{\alpha} = p(\hat{X}_1, ..., \hat{X}_k)$, and $map(X_i, \hat{X}_i)$ expresses the abstract mapping.

If an abstract answer set $\widehat{I}$ is spurious, $\Pi \cup Q^{m}_{\widehat{I}}$ is unsatisfiable; this gives us no information on the reason of spuriousness. To overcome this, we add abnormality atoms, *ab*, in the rules of Π that contain arguments from the domain. This approach is inspired from [5] and the introduced $\overline{ok}$ atoms to the rules. We use a simplified encoding by disregarding loop formulas, thus, deal with tight programs only. E.g., in Example 1 rule (3) is converted to

$$b(X, Y) \leftarrow a(X), d(Y), int(X), int(Y), not\ ab(r3, X, Y).$$

and new rules for a guess over *ab* at a cost for its existence in the answer set are added. This extended program, Π_{ab}, gives us the possibility to catch the rules that need to be deactivated to keep satisfiability while checking the concreteness of an abstract answer set $\widehat{I}$, in case it is spurious.

Refinement search. We run a basic search among all possible refinements of a given initial abstraction (by default, the mapping $m = \{D/k_1\}$) until an abstraction that gives a concrete answer set is reached. In that, for a refinement m' of m, we check the first abstract answer set, $\widehat{I}$, of $\Pi^{m'}$, using Π_{ab}, i.e., $\Pi_{ab} \cup Q^{m'}_{\widehat{I}}$, to see if $\widehat{I}$ is concrete. We then choose the answer set with the smallest number of *ab* atoms in it; this number is the *cost* of the refinement m'. Then a local, distance-based search is done, where the distance between an abstraction and its refinement is the difference in the number of abstract clusters. The refinement with the least cost is picked as the new abstraction until cost 0 is achieved.

Further features. In our implementation, *strong negated literals* $\neg\alpha$ are encoded, at a preprocessing step, as neg_α and constraints of form $\leftarrow \alpha, neg\_\alpha$ are added to the encoding. *Choice rules* are treated specially by ensuring that the abstraction is done on the body, and the choice over the head is kept. We *precompute the deterministic part* of a problem and encode it as facts which are then lifted without introducing (unnecessary) nondeterminism.

6 Applications

Applications usually contain *sorts* that form subdomains of the Herbrand universe. For example, blocksworld contains sorts for blocks and time while in scheduling there are sorts of tasks and time or in coloring there are sorts for nodes and colors. We define an abstraction over a sort as follows.

[1] `http://www.kr.tuwien.ac.at/research/systems/abstraction/`

Fig. 1. Graph 3-coloring instance and abstract solution

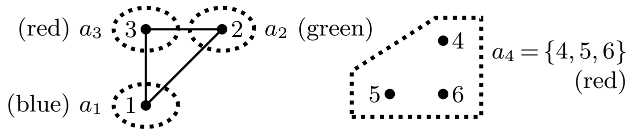
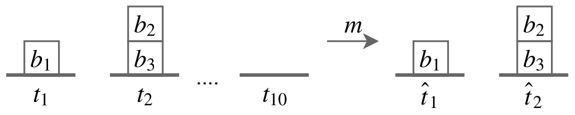


Fig. 2. A blocksworld instance with multiple tables



Definition 7 *An abstraction is limited to a sort $D_i \subseteq D$ if all elements $x \in D \setminus D_i$ form singleton clusters $\{x\}/x$.*

For practical purposes, sorts can use overlapping elements of the domain, provided that all occurrences of the sort are guarded by domain predicates.

We next show our abstraction method on examples.

Example 14 *Consider the 3-coloring problem (encoding omitted for space reasons) and the graph with 6 nodes in Figure 1. The abstraction $\{\{1\}/a_1, \{2\}/a_2, \{3\}/a_3, \{4,5,6\}/a_4\}$, which distinguishes the nodes in the clique 1-2-3 and clusters all others, has* only concrete abstract answer sets*, one of them is $\widehat{I} = \{col(a_1, b), col(a_2, g), col(a_3, r), col(a_4, r)\}$ where the nodes 4,5,6 clustered to a_4 are red.*

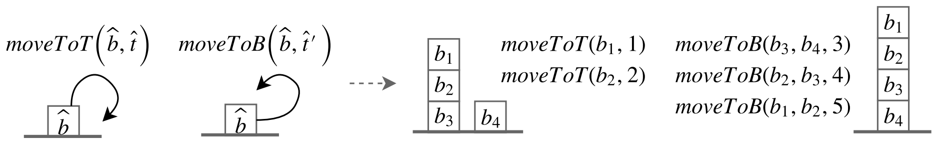
The next problem shows where abstraction allows to grasp the essence of the problem.

Example 15 *Consider a generalized blocks world with multiple tables (Fig. 2). Initially, blocks can be located anywhere; the goal is to pile them up at a picked table, say #1. The (natural) encoding (which is omitted for space reasons) contains the actions $moveToT(B, Ta, T)$ and $moveToB(B, B', T)$ that denotes the moving block B onto table Ta and onto block B', respectively, at time T. An abstraction that distinguishes table #1 and clusters all other tables, i.e., $\{\{t_1\}/\hat{t}_1, \{t_2, \ldots, t_{10}\}/\hat{t}_2\}$, leads to a concrete abstract answer set containing $moveToT(b_2, \hat{t}_2, 0)$, $moveToT(b_3, \hat{t}_1, 1)$, $moveToB(b_2, b_3, 2)$, $moveToB(b_1, b_2, 3)$.*

Starting with an abstraction that puts all tables in one cluster, a one-step refinement that moves table #1 into its own cluster immediately reaches a concrete abstract answer set. This shows that for solving the problem, the number of tables is irrelevant as long as the picked table is distinguished.

Abstraction over Time. In ASP, it is customary to represent time by an additional argument in atoms. Abstraction over time is handled equivalently as to other domains, which can be useful, e.g., in scheduling, time intervals where 'nothing changes' in a schedule can be abstracted as single time points. Moreover, time is an ordered domain which must be respected by the refinements, e.g., by splitting intervals.

Fig. 3. Abstract and concrete plan of Example 17



Example 16 *Consider the disjunctive scheduling problem of [1]: given tasks I with fixed duration D ($task(I, D)$), earliest start time S ($est(I, S)$), latest end time E ($let(I, E)$), and disjunctive constraints ($disj(I, I')$) for tasks that cannot overlap, assign to each task a start time such that all constraints are satisfied. We use the provided encoding (with variables standardized apart) and precomputed deterministic part of the program.*

For an instance $\{task(a, 7), est(a, 1), let(a, 8), task(b, 5), est(b, 3), let(b, 10), task(c, 2), est(c, 8), let(c, 10), disj(a, c), disj(b, c)\}$, we reach from $\{\{1, \ldots, 10\}/k\}$ the abstraction $\{\{4, \ldots, 7\}/k_1, \{9, 10\}/k_2\}$ where only 2 abstract answer sets exist, and a concrete one is easily identified; it yields a solution $time(a, 1)$, $time(b, 3)$, $time(c, 8)$.

Abstraction over Multiple Sorts. While time is important in scheduling and planning, abstracting only over time may not suffice for planning as spurious abstract answer sets with an incorrect order of action execution may occur. This can be countered by additional abstraction over other sorts in the agent domain, which allows for more abstract instances of actions that abstract from the concrete order of application as shown in Example 17 below. It is particularly desirable that the individual abstractions fulfill the following property.

Definition 8 *For a program Π and domain D, subdomains $D_1, \ldots, D_n \subseteq D$ are* independent*, if no rel-atom in Π shares arguments from D_i and D_j, $1 \leq i < j \leq n$.*

For independent sorts, abstractions can be composed.

Proposition 6 *For domain mappings m_1 and m_2 over independent domains D_1 and D_2, $(\Pi^{m_2})^{m_1} = (\Pi^{m_1})^{m_2}$.*

This property readily extends to multiple sorts. Note that sorts in the problems above mentioned are often independent; e.g., blocks, tables and time in Example 15. However, if block number i can not be put on table number j if $i = j$, then above property can not hold.

Abstraction over time and the agent domain allows to obtain abstract plans representing sequences of concrete actions.

Example 17 *Consider a blocksworld problem with a single table shown in Fig.3. The encoding used in Ex. 15 is standardized apart according to the block sort and the time sort. Suppose further rules realize a policy that first puts all blocks on the table and piles them up in a second phase. Given the initial state*

Table 1. Experimental results for 3-coloring (averages)

	full	projected
number of steps	7.65	5.25
abs domain size	8.65	6.19
faithful abs domain size	7.42	6.32
concrete answer in 1 step	0%	3%
trivial abstractions (*id*)	47%	6%
faithful & non-trivial abs.	27%	43%
non-faithful abstraction	26%	51%

Table 2. Experimental results for task scheduling (averages)

	$t = 10$		$t = 20$		$t = 30$	
	v1	v2	v1	v2	v1	v2
number of steps	7.25	3.7	14.6	5.2	22.6	7.4
abs domain size	8.25	8.6	15.6	13.9	23.6	20

$\{onT(b_4,1), onT(b_3,1), onB(b_2,b_3,1), onB(b_1,b_2,1)\}$ *and time domain* $\{1,\ldots,6\}$, *we abstract using the block mapping* $\{\{b_1,\ldots,b_4\}/\hat{b}\}$ *and the time mapping* $\{\{1,2\}/\hat{t}, \{3,\ldots,6\}/\hat{t}'\}$. *The abstract program has 8 answer sets, including* $\{moveToT(\hat{b},\hat{t}), onT(\hat{b},\hat{t}), onB(\hat{b},\hat{b},\hat{t}), onT(\hat{b},\hat{t}'), onB(\hat{b},\hat{b},\hat{t}'), moveToB(\hat{b},\hat{b},\hat{t}')\}$, *which contains two abstract actions:* $moveToT(\hat{b},\hat{t})$ *and* $moveToB(\hat{b},\hat{t}')$ *(Fig. 3).*

7 Experiments

To assess the approach and the tool, we conducted preliminary experiments for three of the problems presented above.

3-Coloring. We randomly generated 20 graphs on 10 nodes with edge probability 0.1, 0.2, ..., 0.5 each; out of the 100 graphs, 74 were 3-colorable. We evaluated the abstraction m reached from the single-cluster abstraction, by checking whether the corresponding abstract program has spurious abstract answer sets (if not, m is *faithful*). In addition, we considered a *projected* notion of concreteness that limits the checking to a set of relevant atoms. E.g., only the colors of nodes 1-3 may be relevant, and an abstraction that assigns colors to them may be sufficient. Table 1 shows the collected results. In case of projection, the trivial abstraction is reached (in 9 steps) much less than in the full case; moreover, more non-trivial faithful abstractions are reached, which is beneficial. Moreover, 80% of the non-colorable graphs were revealed by non-trivial, faithful full abstractions, and 77% under projection; hence, abstraction may be useful to catch and explain unsolvability.

Disjunctive scheduling. For each $t \in \{10, 20, 30\}$, we generated 20 instances with 5 tasks over time $\{1,\ldots,t\}$. Table 2 shows the collected results. For the refinement search, we considered besides the one from above (v1) another one that looks at the domain elements in the *ab* atoms and guides the refinement either to not map these elements to the same cluster or to map them into singleton clusters (v2). Observe that in v2 the number of steps to obtain a solution is greatly reduced which moreover has fewer clusters (except for $t = 10$ as creating singleton clusters quickly ends up with the trivial abstraction). The results show that with larger domains, the effect of the abstraction can be seen much better, e.g., the average abstract domain size reached for $t = 30$ is 66.6% (=20/30) of the original domain, while for $t = 10$, it shrinks to 86%. Note that with more sophisticated refinement methods, better abstractions can be reached.

Multi-table blocksworld. We considered varying numbers of blocks and tables, starting with 5 each. Faithful abstractions readily resulted by 1-step refinements which separated the chosen table from the rest. However, as the abstraction is syntactic, other encodings may need more steps (e.g., bad auxiliary rules causing choices/spuriousness).

8 Conclusion

Related Work. Apart from simplification approaches to ASP we mentioned earlier, abstraction has been studied in logic programming [9]. However, the focus was on the use of abstract interpretations and termination analysis, and stable semantics was not addressed.

In planning, plan refinement [26, 21] uses abstract plans computed in an abstract space to find a concrete plan, while abstraction-based heuristics [10, 17] use the costs of abstract solutions to guide the plan search. Pattern databases [10] project the state space to a set of variables (a 'pattern'), while merge & shrink abstraction [17] starts with a suite of single projections, and then computes an abstraction by merging them and shrinking. In [19], abstraction for numeric planning problems by reduction to classical planning is studied.

Abstraction is further studied for agent verification in situation calculus action theory [2] and multi-agent systems against specifications in epistemic logic [24] and temporal logic [3]. Lomuscio and Michaliszyn [24] present an automated predicate abstraction method in 3-valued semantics, and interpolant-based refinement [4].

Apparently, all these works are quite different from ours, as they address specific applications and are based on different (monotonic) logic formalisms.

Outlook. This seminal work has room for improvement, especially in the search for an abstraction and the refinement. Different heuristics may be employed in the search on the refinements of an abstraction. The latter can be made more sophisticated by using domain-specific knowledge. Furthermore, the current quality assessment of refinements can be made more sophisticated by considering more than one abstract answer set or making the largest cluster size a parameter in determining the refinement quality. Predicate abstraction, where different predicates are represented with a single abstract one, would be an interesting extension of this work.

References

1. ASPCOMP-11: Third (open) answer set programmming competition: Disjunctive scheduling (2011), `www.mat.unical.it/aspcomp2011`
2. Banihashemi, B., De Giacomo, G., Lespérance, Y.: Abstraction in situation calculus action theories. In: Proc. of AAAI. pp. 1048–1055 (2017)

3. Belardinelli, F., Lomuscio, A.: Abstraction-based verification of infinite-state reactive modules. In: Proc. of ECAI. pp. 725–733 (2016)
4. Belardinelli, F., Lomuscio, A., Michaliszyn, J.: Agent-based refinement for predicate abstraction of multi-agent systems. In: ECAI. pp. 286–294 (2016)
5. Brain, M., Gebser, M., Pührer, J., Schaub, T., Tompits, H., Woltran, S.: Debugging asp programs by means of asp. In: Proc. LPNMR. pp. 31–43. Springer (2007)
6. Brass, S., Dix, J.: Characterizations of the disjunctive stable semantics by partial evaluation. J. Log. Program. **32**(3), 207–228 (1997)
7. Brewka, G., Eiter, T., Truszczyski, M.: Answer set programming at a glance. Communications of the ACM **54**(12), 92–103 (2011)
8. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM TOPLAS pp. 1512–1542 (1994)
9. Cousot, P., Cousot, R.: Abstract interpretation and application to logic programs. The Journal of Logic Programming **13**(2), 103 – 179 (1992)
10. Edelkamp, S.: Planning with pattern databases. In: Sixth European Conf. on Planning (2001)
11. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: Proc. JELIA. pp. 200–212. Springer (2004)
12. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: Engineering an incremental ASP solver. In: Proc. ICLP. pp. 190–205 (2008)
13. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The Potsdam Answer Set Solving Collection. AI Comm. **24**(2), 107–124 (2011)
14. Geißer, F., Keller, T., Mattmüller, R.: Abstractions for planning with state-dependent action costs. In: ICAPS. pp. 140–148 (2016)
15. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP. pp. 1070–1080 (1988)
16. Giunchiglia, F., Walsh, T.: A theory of abstraction. AIJ **57**(2-3), 323–389 (1992)
17. Helmert, M., Haslum, P., Hoffmann, J., Nissim, R.: Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. JACM **61**(3), 16 (2014)
18. Hoffmann, J., Sabharwal, A., Domshlak, C.: Friends or Foes? an AI planning perspective on abstraction and search. In: ICAPS. pp. 294–303 (2006)
19. Illanes, L., McIlraith, S.A.: Numeric planning via search space abstraction. In: Proc. KnowProS@IJCAI (2016)
20. Janhunen, T., Niemelä, I., Seipel, D., Simons, P., You, J.H.: Unfolding partiality and disjunctions in stable model semantics. ACM TOCL **7**(1), 1–37 (Jan 2006)
21. Knoblock, C.A.: Automatically generating abstractions for planning. Artificial intelligence **68**(2), 243–302 (1994)
22. Kouvaros, P., Lomuscio, A.: A counter abstraction technique for the verification of robot swarms. In: Proc. of AAAI (2015)
23. Leite, J.: A bird's-eye view of forgetting in answer-set programming. In: Proc. LPNMR. pp. 10–22 (2017)
24. Lomuscio, A., Michaliszyn, J.: Verification of multi-agent systems via predicate abstraction against ATLK specifications. In: Proc. of AAMAS. pp. 662–670 (2016)
25. Pearce, D.: Simplifying logic programs under answer set semantics. In: Demoen, B., Lifschitz, V. (eds.) Logic Programming. pp. 210–224 (2004)
26. Sacerdoti, E.D.: Planning in a hierarchy of abstraction spaces. Artificial intelligence **5**(2), 115–135 (1974)
27. Saribatur, Z.G., Eiter, T.: Omission-based abstraction for answer set programs. In: Proc. KR. pp. 42–51 (2018)