Abstraction for Non-Ground Answer Set Programs

Zeynep G. Saribatur, Peter Schüller, and Thomas Eiter

Institute of Logic and Computation,
TU Wien, Vienna, Austria

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 [8, 23, 2, 16, 14]. By omitting details, 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, 26], partial evaluation [6, 21], or forgetting [24], 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, such an approach was presented in [29] 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

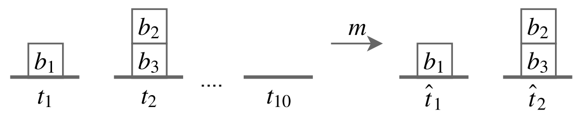
$$col(X, r) \leftarrow node(X), not\ col(X, g), not\ col(X, b).$$

may express that node X must be red if it is neither green nor blue; or the rule

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

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.

Fig. 1. Initial state of a blocksworld with multiple tables (concrete $\overset{m}{\to}$ abstract).



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 one 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.

Example 1 *Figure 1 depicts a generalized blocks world with multiple tables. The (natural) encoding (cf. Appendix A*[1]*) contains the actions* $moveToT(B, Ta, T)$ *and* $moveToB(B, B', T)$ *that denote moving block* B *onto table* Ta *and onto block* B'*, resp., at time* T*. Initially, blocks can be located anywhere; the goal is to pile them up at a picked table, say* t_1*. An abstraction that distinguishes table* t_1 *and clusters all other tables, leads to a concrete abstract answer set containing* $moveToT(b_2, \hat{t}_2, 0), moveToT(b_3, \hat{t}_1, 1), moveToB(b_2, b_3, 2), moveToB(b_1, b_2, 3)$*. The abstraction shows that, for solving the problem, it is essential to distinguish the picked table from all others and that the number of tables is irrelevant.*

Although lots of advanced solving techniques are available for ASP, the support for program analysis, especially in singling out relevant objects, is scarce. 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' so 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 in finding non-trivial abstractions for various applications.

[1] `http://www.kr.tuwien.ac.at/staff/zeynep/pub/jelia/SSE19appendix.pdf`

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 $B(r) = l_1, \dots, l_n$ 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 is a *constraint* if α is falsity ($\perp$, then omitted). 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 2 (Running example) *Consider the following example program Π with domain predicate int/1 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 furthermore 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 $m : \mathcal{A} \rightarrow \mathcal{A}'$ 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 $m : D \rightarrow \widehat{D}$ 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 3 (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 $a = p(v_1, \ldots, v_n)$ by $m(a) = p(m(v_1), \ldots, m(v_n))$. 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 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 4 (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 lifted 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 built-ins, we permit* $e(k)$ *to be false even if* $c(k)$ *and* $a(k)$ *holds by creating instead the following rule:*

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

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 5 (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 a choice rule with a flipped relation,* $X = Y$, *catches such cases.*

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

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

$e(k)$ 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 $e(4)$ but they are lost in the abstraction. Such cases are caught by additional rules with reversed negation for $a(Y)$:

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

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

3.1 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, we focus on rules of form

$$r : l \leftarrow B(r), \Gamma_{rel}(r)$$

where the variables in $B(r)$ are standardized apart and Γ_{rel} consists of built-in atoms that constrain the variables in $B(r)$. E.g., the rule (3) has $\Gamma_{rel}(r) = \top$ while the rule (5) must be standardized apart into $\leftarrow b(X,Y), e(X_1), \Gamma_{rel}$ with $\Gamma_{rel} = (X = X_1)$.

Uncertainty is caused by relation restrictions over non-singleton clusters (i.e., $|\widehat{d}| > 1$) or by negative literals mapped to non-singleton abstract literals. For simplicity, we first focus on binary built-ins, e.g., $=, <, \leq, \neq$, and a $\Gamma_{rel}(r)$ of the form $rel(X,c)$ or $rel(X,Y)$. When the relation rel is lifted to the abstract domain, the following cases τ_{I}–τ_{IV} for $rel(\hat{d}_1, \hat{d}_2)$ occur in a mapping:

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

If $rel(\hat{d}_1, \hat{d}_2)$ 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 6 *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 $rel(\hat{d}_1, \hat{d}_2)$ 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 7 (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 $\tau_\iota^{rel}(\hat{d}_1, \hat{d}_2)$ where $\iota \in \{\mathrm{I}, \ldots, \mathrm{IV}\}$ is the type of the built-in instance $rel(\hat{d}_1, \hat{d}_2)$ 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 Π 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 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 rule* $r: \ l \leftarrow B(r), rel(t_1, t_2)$ *as above and a domain mapping* m, *the set* r^m *contains the following rules:*

(a) $m(l) \leftarrow m(B(r)), rel(\hat{t}_1, \hat{t}_2), \tau_{\mathrm{I}}^{rel}(\hat{t}_1, \hat{t}_2)$.

(b) $\{m(l)\} \leftarrow m(B(r)), rel(\hat{t}_1, \hat{t}_2), \tau_{\mathrm{III}}^{rel}(\hat{t}_1, \hat{t}_2)$.

(c) $\{m(l)\} \leftarrow m(B(r)), \overline{rel}(\hat{t}_1, \hat{t}_2), \tau_{\mathrm{IV}}^{rel}(\hat{t}_1, \hat{t}_2)$.

(d) *For* $l_i \in S^-_{rel}(r)$:

  (i) $\{m(l)\} \leftarrow m(B^{sh}_{l_i}(r)), rel(\hat{t}_1, \hat{t}_2), \tau_{\mathrm{III}}^{rel}(\hat{t}_1, \hat{t}_2)$.

  (ii) $\{m(l)\} \leftarrow m(B^{sh}_{l_i}(r)), \overline{rel}(\hat{t}_1, \hat{t}_2), \tau_{\mathrm{IV}}^{rel}(\hat{t}_1, \hat{t}_2)$.

  (iii) $\{m(l)\} \leftarrow m(B^{sh}_{l_i}(r)), rel(\hat{t}_1, \hat{t}_2), isCluster(l_i)$.

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. The auxiliary atom* $isCluster(l_i)$ *holds true if a variable from* $arg(l_i)$ *is mapped to a non-singleton cluster.*

In step (a), the case of having no uncertainty due to abstraction is applied. Steps (b) and (c) are for the cases 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 (which is causing the uncertainty). Constraints (e.g., (5)) are omitted in the cases with uncertainty (i.e., all steps except (a)).

Example 8 (ctd) *Consider Ex 2 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*

$\tau_{\mathrm{I}}^{\leq}(x,y)$ *true for* $(x,y) \in \{(k_1,k_1), (k_1,k_2), (k_1,k_3), (k_2,k_3)\}$*, and* $\tau_{\mathrm{III}}^{\leq}(x,y)$ *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:*

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

In step (d) 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 $B^{sh}_{l_i}(r)$. (d-iii) shifts the negative literal only if it shares arguments mapped to a non-singleton cluster.

Example 9 (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:*

$$c(X) \leftarrow not\ d(X), X < k_3, \tau_{\mathrm{I}}^{<}(X,k_3), \widehat{int}(X).$$
$$\{c(X)\} \leftarrow not\ d(X), X \geq k_3, \tau_{\mathrm{IV}}^{<}(X,k_3), \widehat{int}(X).$$
$$\{c(X)\} \leftarrow d(X), X \geq k_3, \tau_{\mathrm{IV}}^{<}(X,k_3), \widehat{int}(X).$$
$$\{c(X)\} \leftarrow d(X), X < k_3, isCluster(d(X)), \widehat{int}(X).$$

The abstract program is now as follows.

Definition 4 *Given a program* Π *as above and a domain abstraction* m*, the abstract program for* m *consists of the rules*

$$\Pi^m = \bigcup_{r:\ l \leftarrow B(r), rel(t_1,t_2) \in \Pi} r^m \cup \{x. \mid x \in \mathcal{T}_m\} \cup \{m(p(\boldsymbol{c})). \mid p(\boldsymbol{c}). \in \Pi\}.$$

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 steps (a)-(b) 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. Steps (c)-(d) 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 $\Gamma_{rel} = rel(t_{1,1}, t_{2,1}), .., rel(t_{1,k}, t_{2,k})$, $k > 1$, is to view it as literal of an $2k$-ary built-in $rel'(X_{1,1}, X_{2,1}, .., X_{1,k}, X_{2,k})$. The abstract version of such rel' and the cases I-IV are lifted from x_1, x_2 to $x_1, .., x_n$. 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 $\hat{d}_1, .., \hat{d}_4$ s.t. $\hat{d}_1 = \hat{d}_2 \wedge \hat{d}_3 = \hat{d}_4$ 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 10 *Consider the rules* (1)-(2) *(Ex. 2) and the mapping* $\{\{1,\ldots,5\}/k\}$. *The abstract rules for them are*

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

in addition to the abstracted rules due to step (a). 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. For example, we eliminate $d(X)$ and $c(X)$ from the bodies of abstract rules (6)–(8).

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 has no 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) $AS(\Pi^{m_{id}}) = \{I \cup \mathcal{T}_{m_{id}} \mid I \in AS(\Pi)\}$ *for identity* $m_{id} = \{\{x\}/x \mid x \in D\}$.

(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 *Given a program Π, a domain mapping m and an abstract answer set $\widehat{I} \in AS(\Pi^m)$, deciding whether $\widehat{I}$ is not 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). As for testing faithfulness, we note the following result:

Theorem 5 *Given a program Π and a domain mapping m, deciding whether Π^m is faithful, i.e., has no spurious answer set, is co-$\mathbf{NEXP}^{\mathbf{NP}}$-complete in general and Π_3^p-complete for bounded predicate arities (i.e., by a constant).*

Membership is shown by a guess & check algorithm resorting to answer set existence, and 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 6 *$\widehat{I}$ 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 $m' : D \to D''$ 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. As an example, mapping $m' = \{\{1\}/k_1, \{2\}/k_{2,1}, \{3\}/k_{2,2}, \{4,5\}/k_3\}$ is a refinement of mapping $m = \{\{1\}/k_1, \{2,3\}/k_2, \{4,5\}/k_3\}$.

5.1 Implementation

We have implemented the workflow described above in a tool[2] that uses Python and Clingo 5 [13]. We next discuss practical implementation issues.

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

$$\bot \leftarrow in(\hat{\alpha}), \{\alpha : map(X_1, \hat{X}_1), \dots, map(X_k, \hat{X}_k)\} \leq 0.$$
$$\bot \leftarrow \alpha, not\ in(\hat{\alpha}), map(X_1, \hat{X}_1), \dots, map(X_k, \hat{X}_k)$$

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, with the set of facts $\{in(\hat{\alpha}). \mid \hat{\alpha} \in \widehat{I}\}$.

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] that introduces *tagging* atoms to the rules. We use a simplified encoding by disregarding loop formulas (cf. Appendix B); thus, we deal with tight programs only. E.g., in Example 2 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 in order 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. 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; we call this number the *cost* of the refinement m'. Then, we perform a local distance-based search, where the distance between an abstraction and its refinement is the difference in the number of abstract clusters. We pick the refinement with the least cost as the new abstraction until cost 0 is achieved.

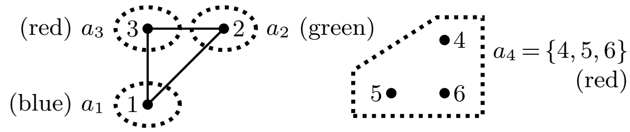
Further features. In our implementation, *strong negated literals* $\neg\alpha$ are encoded, at a preprocessing step, as neg_α and constraints of form $\leftarrow \alpha, neg\_\alpha$ are added to the encoding. *Choice rules* are treated specially by ensuring that the abstraction is done on the body, and the choice over the head is kept. We *precompute the deterministic part* of a program (i.e., not involved in unstratified negation resp. guesses) 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

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

Fig. 2. Graph 3-coloring instance and abstract solution



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.

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 11 *Consider the following 3-coloring encoding:*

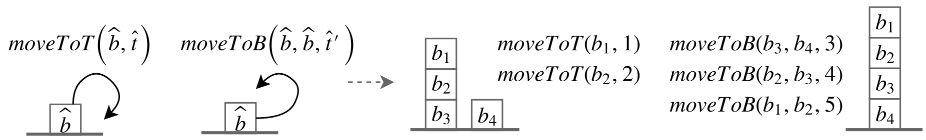
$col(X_1, r) \leftarrow not\ col(X_1, g), not\ col(X_2, b), X_1 = X_2.$
$col(X_1, g) \leftarrow not\ col(X_1, r), not\ col(X_2, b), X_1 = X_2.$
$col(X_1, b) \leftarrow not\ col(X_1, g), not\ col(X_2, r), X_1 = X_2.$
$hasEdgeTo(X, C) \leftarrow edge(X, Y_1), color(Y_2, C), Y_1 = Y_2.$
$\leftarrow hasEdgeTo(X_1, C), col(X_2, C), X_1 = X_2.$
$\leftarrow node(X), not\ colored(X).$
$colored(X) \leftarrow col(X, C), node(X).$

and the graph with 6 nodes in Figure 2. 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.*

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

Example 12 *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*

Fig. 3. Abstract and concrete plan of Example 13



only two 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 13 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 7 *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 1. However, if block number i can not be put on table number j if $i = j$, then the above property can not hold.

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

Example 13 *Consider the blocksworld problem with a single table in Fig.3. The encoding of Example 1 is modified for a single table (table argument omitted from moveToT/onT). The encoding gets 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. (heads of form $1\{\ldots\}$ *choose at least one element and can here be treated like explained before):*

$existsOnBlock(T) \leftarrow onB(B, B_1, T).$

$allOnTable(T) \leftarrow not\ existsOnBlock(T), time(T).$

$atPhase2(T_1) \leftarrow allOnTable(T), T < T_1.$

$1\{moveToT(B,T) : onB(B, B_1, T)\} \leftarrow T < t_{max}, not\ atPhase2(T), not\ allOnTable(T).$

$1\{moveToB(B, B_1, T) : onT(B,T), block(B_1)\} \leftarrow T < t_{max}, allOnTable(T).$

$1\{moveToB(B, B_1, T) : onT(B,T), onB(B_1, B_2, T)\} \leftarrow T < t_{max}, atPhase2(T).$

Table 1. Experimental results for 3-coloring (above) and scheduling (below).

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

	$t = 10$: v1	v2	$t = 20$: v1	v2	$t = 30$: 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

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

7 Experiments

To see whether our approach automatically finds non-trivial domain abstractions that yield concrete answer sets, we conducted several experiments.

3-Coloring. We randomly generated 20 graphs on 10 nodes with edge probability $0.1, 0.2, \dots, 0.5$ each; out of the 100 graphs, 74 were 3-colorable. We evaluated the abstraction m reached from the initial single-cluster abstraction, by checking whether the corresponding abstract program has spurious 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. The left side shows the average number of steps needed until a concrete answer was found, and the average of the resulting abstract domain sizes. The right side shows the percentage of the observed properties of the resulting abstractions. Trivial abstraction (*id*) corresponds to the case where the abstraction is refined back to the original domain. Observe that faithful and non-trivial abstractions were achieved, which shows the potential of the approach in singling out relevant objects. 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. Furthermore, 80% of the non-colorable graphs were revealed by non-trivial 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 $\{1, \dots, t\}$. Table 1 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 [28, 22] 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. Recently, the same authors used abstraction for problems that contain quantifiable objects [20], e.g., some number of packages to deliver to points A and B, to find generalized plans by abstracting away from the quantification that works for multiple instances of the problem. For this, they build a quantified planning problem by identifying sets of indistinguishable objects using reformulation techniques [27] to reduce symmetry, and then use an algorithm to compute a general policy. With our method, abstracting over the packages and time is possible as done in Example 13. It constructs an abstract program which contains a generalized plan (among possible spurious ones) for all instances of the problem. Furthermore, if the package delivery problem is extended with having a choice of points to pass through when moving from A to B, then abstracting over the points passed to reach B from A is possible with our method. Such a constraint is not representable by [20] due to the quantifiability conditions. Nevertheless, our method has the orthogonal potential drawback of producing spurious answers.

Abstraction was also studied for agent verification in situation calculus action theory [2] and multi-agent systems against specifications in epistemic logic [25] and temporal logic [3]. Lomuscio and Michaliszyn [25] present an automated predicate abstraction method in 3-valued semantics, and interpolant-based refinement [4]. All these works are quite different from ours; 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 a refinement, where different heuristics may be employed. It can also be made more sophisticated by using domain-specific knowledge. Furthermore, the current quality assessment of refinements can be advanced by considering more than one abstract answer set or making the largest cluster size a parameter in determining the refinement quality. Predicate abstraction would be an interesting extension of this work. Our aim was not to increase reasoning efficiency, but this is an interesting future direction that needs significant follow-up work.

Acknowledgements

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).

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. Illanes, L., McIlraith, S.A.: Generalized planning via abstraction: Arbitrary numbers of objects. In: AAAI (2019)
21. 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)
22. Knoblock, C.A.: Automatically generating abstractions for planning. Artificial intelligence **68**(2), 243–302 (1994)
23. Kouvaros, P., Lomuscio, A.: A counter abstraction technique for the verification of robot swarms. In: Proc. of AAAI (2015)
24. Leite, J.: A bird's-eye view of forgetting in answer-set programming. In: Proc. LPNMR. pp. 10–22 (2017)
25. Lomuscio, A., Michaliszyn, J.: Verification of multi-agent systems via predicate abstraction against ATLK specifications. In: Proc. of AAMAS. pp. 662–670 (2016)
26. Pearce, D.: Simplifying logic programs under answer set semantics. In: Demoen, B., Lifschitz, V. (eds.) Logic Programming. pp. 210–224 (2004)
27. Riddle, P., Douglas, J., Barley, M., Franco, S.: Improving performance by reformulating pddl into a bagged representation. In: HSDIP@ICAPS. pp. 28–36 (2016)
28. Sacerdoti, E.D.: Planning in a hierarchy of abstraction spaces. Artificial intelligence **5**(2), 115–135 (1974)
29. Saribatur, Z.G., Eiter, T.: Omission-based abstraction for answer set programs. In: Proc. KR. pp. 42–51 (2018)