Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-22)
Journal Track

Abstraction for Non-Ground Answer Set Programs (Extended Abstract)*

Zeynep G. Saribatur† , **Thomas Eiter** , **Peter Schüller**
Institute of Logic and Computation, TU Wien
{zeynep, eiter, ps}@kr.tuwien.at

Abstract

Abstraction is a powerful technique that has not been considered much for nonmonotonic reasoning formalisms including Answer Set Programming (ASP), apart from related simplification methods. We introduce a notion for abstracting from the domain of an ASP program that shrinks the domain size and over-approximates the set of answer sets, as well as an abstraction-&-refinement methodology that, starting from an initial abstraction, automatically yields an abstraction with an associated answer set matching an answer set of the original program if one exists. Experiments reveal the potential of the approach, by its ability to focus on the program parts that cause unsatisfiability and by achieving concrete abstract answer sets that merely reflect relevant details.

1 Introduction

For solving combinatorial problems and figuring out the key elements, humans arguably employ abstraction. In AI, such problems vary from planning problems like blocks world to solving constraint problems such as finding an admissible node coloring of a given graph. In the latter problem, for instance, isolated nodes can be viewed as a single node and colored the same without thinking about the specific details (Fig. 1). If the graph is non-colorable, we may try to find some subgraph (e.g., a clique) which causes the unsolvability, and we would not care about other nodes in the graph. Similarly with the blocks: if the labels are not important, we would disregard them. If the goal configuration cannot be achieved, we would aim to find out the particular blocks causing this.

Notably, such disregard of detail also occurs for problems with multi-dimensional structures such as grid-cells in the Sudoku problem, where a partially filled 9×9 board must be completed using numbers 1..9 under constraints. If a Sudoku is unsolvable, humans can meaningfully grasp the reason only by focusing on relevant sub-regions, as looking at the whole grid is too complex. To illustrate, Fig. 1 shows for an instance the sub-regions causing that no solution exists: as 6 and 7 occur in the middle column, they must appear in the sub-region below in the left column where only a single cell is empty.

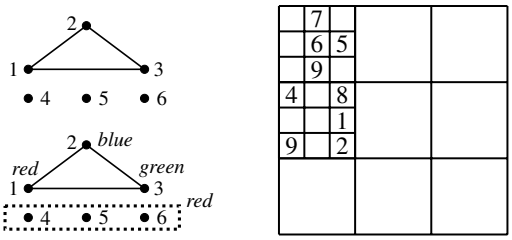

Figure 1: Abstraction for graph 3-coloring (left) and Sudoku (right)

In [Saribatur and Eiter, 2018], a method for abstraction of ground (propositional) answer set programming (ASP) has been presented. In this paper, the approach is lifted to non-ground ASP programs, with the following contributions.

(1) We formally introduce domain abstraction for ASP programs Π. To this end, we define abstraction mappings m from the original (concrete) domain D of Π to an abstract domain D', and construct an abstract program Π' over D' such that each answer set I of Π maps to an abstract answer set I' of Π'. We give a systematic approach to construct Π' that modularly transforms each rule in Π into a set of abstract rules of similar structure. Built-in relations and in particular $=$, whose treatment is the backbone of the method, are lifted to the abstract level, where uncertainty caused by D' is carefully respected.

(2) We present a method to compute for spurious abstract answer sets I' of a program Π w.r.t. mapping m a refinement m' of m that eliminate I'. To find promising such m', we apply an ASP meta-programming technique for debugging a program built for I', Π, and m whose unsatisfiability shows that I' is spurious. To this end, we lift the SPOCK approach [Brain *et al.*, 2007] to the non-ground level such that decisions on m' can be based on special debugging atoms. We embed this in a CEGAR-style [Clarke *et al.*, 2003] iterative abstraction and refinement methodology that automatically searches for an abstraction having some non-spurious answer set.

(3) We introduce multi-dimensional abstraction mappings over a domain. While our basic abstraction method can deal with sorts, it must be modified to form an abstraction over the relations akin to existential abstraction [Clarke *et al.*, 2003]. We extend our methodology with handling the structural aspects of grid-cells by using an abstraction of quad-trees, and

*Appears in *Artificial Intelligence* 300 (2021) 103563.
†Contact Author

we consider more sophisticated decision making for the refinement to observe its effects on the resulting abstractions.

(4) We analyze semantic and computational properties of the abstraction approach, which can be exploited for modeling and for guiding the design of suitable implementations.

(5) An evaluation of our prototypical tools DASPAR and mDASPAR for plain resp. multi-dimensional abstraction, on finding non-trivial abstractions for problems with well-known ASP encodings (graph coloring, scheduling) and detecting unsolvable problems over grid-cells, shows the potential of the approach. Notably a small user-study for a natural grid-cell problem indicates its capability of putting a human-like focus.

This is an extended abstract of [Saribatur *et al.*, 2021], which has elaborated technical definitions and more results.

2 Abstraction in ASP

We consider finite logic programs Π with rules r of the form $H(r) \leftarrow B(r)$ where $H(r)$ is either a function-free atom α or falsity ($\bot$), and $B(r) = l_1, \ldots, l_n$ consists of literals l_i of either the form β_i or $not\ \beta_i$, where β_i is a function-free atom and not is default negation; r is a *fact* if $n=0$ and it is ground (variable-free). The *Herbrand universe* (resp., *Herbrand base*) of Π is the set of all constants (resp. ground atoms with predicates and constants) occurring in it.

The set of stable models (or answer sets) $AS(\Pi)$ of Π is defined from its grounding $grd(\Pi) = \bigcup_{r\in\Pi} grd(r)$, where $grd(r)$ contains all ground instances of rule r [Gelfond and Lifschitz, 1988]. They amount to the Herbrand interpretations (i.e., subsets of the Herbrand base) I of $grd(\Pi)$ that are $\subseteq$-minimal models of $\Pi^I = \{H(r) \leftarrow B^+(r) \mid r \in grd(\Pi), I \models B(r)\}$, where $B^+(r)$ are the not -free literals in $B(r)$. We call Π *unsatisfiable* if $AS(\Pi) = \emptyset$.

We utilize *choice rules* r with $H(r) = \{\alpha\}$, which stand for $\alpha \leftarrow B(r), not\ \alpha'$ and $\alpha' \leftarrow B(r), not\ \alpha$ with a fresh atom α'.

Definition 1 (cf. [Saribatur and Eiter, 2018])**.** *A ground program Π' is an* abstraction *of a ground program Π w.r.t. a mapping $m : \mathcal{A} \to \mathcal{A}' \cup \{\top\}$ from atoms $\mathcal{A}$ of Π to atoms $\mathcal{A}'$ of Π', where $|\mathcal{A}| \geq |\mathcal{A}'|$, and a truth constant $\top$, if for every $I \in AS(\Pi)$, it holds that $I' = \{m(\alpha) \mid \alpha \in I\} \in AS(\Pi')$. Further Π' is an* abstraction *of Π, if such a mapping m exists.*

We note the following useful property.

Proposition 1. *Let Π' be an abstraction of Π. If $AS(\Pi') = \emptyset$, then $AS(\Pi) = \emptyset$, i.e., abstraction preserves satisfiability.*

In general, over-approximation can cause abstract answer sets that have no corresponding original answer set.

Definition 2. *Given an abstraction Π' of Π for the mapping m, an answer set $I' \in AS(\Pi')$ is* concrete *if some answer set $I \in AS(\Pi)$ exists s.t. $m(I) = I'$; otherwise, I' is* spurious*.*

Abstraction Refinement Methodology. We consider a CEGAR-style abstraction refinement approach (Fig. 2) that refines an initial abstraction repeatedly until a concrete solution is found or unsatisfiability is detected.

3 Domain Abstraction

To illustrate the abstraction and its various challenges, we use the following example.

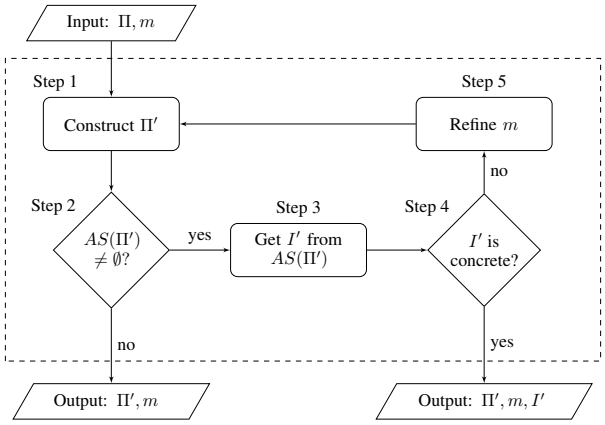

Figure 2: Abstraction & Refinement Methodology

Example 1 (running example)**.** *Consider the program Π:*

$$a(1).\ a(3). \tag{1}$$
$$c(X) \leftarrow not\ d(X), dom(X). \tag{2}$$
$$d(X) \leftarrow not\ c(X), dom(X). \tag{3}$$
$$b(X,Y) \leftarrow a(X), d(Y). \tag{4}$$
$$e(X) \leftarrow c(X), a(Y), X \neq Y. \tag{5}$$
$$\bot \leftarrow b(X,Y), e(X). \tag{6}$$

Here dom is a domain predicate, assumed to hold true for all constants over which Π is grounded, say $D = \{1, \ldots, 5\}$.

The abstraction mapping is defined over the Herbrand universe of Π as D, called *domain*, by merging the constants.

Definition 3. *Given a domain D of Π, a* domain abstraction (da) mapping *is a surjective function $m : D \to \widehat{D}$ for a set $\widehat{D}$ (the* abstracted domain*).*

Thus, a da-mapping divides D into *clusters* $\{d' \in D \mid m(d') = \hat{d}\}$ of elements seen as equal, identified by $\hat{d} \in \widehat{D}$.

Example 2 (ctd)**.** *Π has the Herbrand universe $\{1,2,3,4,5\}$. A da-mapping for $\widehat{D}_1 = \{k_1, k_2, k_3\}$ is $m_1 = \{\{2,3\} \mapsto k_2, \{4,5\} \mapsto k_3, \{1\} \mapsto k_1\}$, clustering $2,3$ to k_2; $4,5$ to k_3; and 1 to k_1.[1] A naive da-mapping is $m_2 = \{D \mapsto k\}$ for $\widehat{D}_2 = \{k\}$.*

Abstracting the elements in the Herbrand universe induces an abstraction of the Herbrand base. Each domain abstraction mapping m naturally extends to ground atoms as follows:

$$\text{for } \alpha = p(v_1, \ldots, v_n) \text{ let } m(\alpha) = p(m(v_1), \ldots, m(v_n)).$$

Then α is *mapped to a singleton cluster* if $|m^{-1}(m(\alpha))| = 1$, and is *mapped to a non-singleton cluster* otherwise.

Example 3 (ctd)**.** *The ground atoms $a(1)$, $b(1,5)$ and $e(2)$, for instance, are changed by m into $a(k_1)$, $b(k_1,k_3)$, $e(k_2)$, respectively; $a(1)$ is mapped to a singleton cluster while $b(1,5), e(2)$ are mapped to non-singleton clusters.*

Given a (non-ground) program Π and an induced mapping $m : \mathcal{A} \to \hat{\mathcal{A}}$ from the Herbrand base $\mathcal{A}$ of Π to $\hat{\mathcal{A}} = \{m(\alpha) \mid \alpha \in \mathcal{A}\}$, we want a (non-ground) abstract program Π' that achieves over-approximation as in Defn 1. However, even for a ground Π, simply applying m does not work in general.

[1] We write $m^{-1}(\hat{d}) \mapsto \hat{d}$ to denote $e \mapsto \hat{d}$ for each $e \in m^{-1}(\hat{d})$.

3.1 Towards an Over-Approximation

A naive way of simply keeping the rules in Π and considering the abstract domain for evaluation turns out to be not enough to achieve an over-approximation.

Shared arguments. Arguments in a rule body that are shared (occur multiply) outside dom may cause an issue. This is because truth of an atom $p(k)$ in the abstract program for a cluster k represents that for some, but not necessarily all, original domain elements x in k the atom $p(x)$ holds true. Instead of treating the shared arguments for each predicate individually, we use a uniform approach relegating the treatment to auxiliary atoms, by *standardizing apart* the shared arguments.

(In)equality relation over the abstract domain. The domain clustering might cause that (in)equality relations in the abstract domain fail in the original domain, resulting in an *uncertainty* that the abstraction process must treat. We handle this by assigning *types* to the relations w.r.t the mappings.

Overall, for a relation $\circ$, we have four relation types. Type I, $\tau^{\circ}_{\mathrm{I}}$, and type II, $\tau^{\circ}_{\mathrm{II}}$, are the cases of no uncertainty, while Type III, $\tau^{\circ}_{\mathrm{III}}$, and type IV, $\tau^{\circ}_{\mathrm{IV}}$ are the cause for uncertainty.

Example 4 (ctd)**.** *Rule* (6) *has a shared use of the variable* X*. By standardizing apart multiple occurrences of* X*, we get*

$$\bot \leftarrow b(X,Y), e(X_1), X = X_1. \tag{7}$$

Applying $=$ *on* $\widehat{D}_1$ *for* m_1 *causes uncertainty: if we instantiate* X *and* X_1 *to* k_2*, then while* $k_2 = k_2$ *holds not all elements mapped by* m_1 *to* k_2 *satisfy equality* $=$*, e.g.,* $2 \neq 3$*. We refer to this case as* type III*, denoted for* k_2 *by* $\tau^{=}_{\mathrm{III}}(k_2, k_2)$.

The set $\mathcal{T}_m$ contains all such abstract relation type atoms. For type III and type IV, we turn the head of a rule into a choice in order to capture the possible firing and the non-firing of the original rule in alternative abstract answer sets.

Default negation. For non-singleton clusters, negative literals may cause a loss of original answer sets. To cope with this, we add a rule for such clusters, where in the original rule the polarity of $not\ \alpha$ is shifted and its head becomes a choice.

3.2 Abstract Program Construction

The basic idea to construct an abstract program Π^m 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.

Example 5 (ctd)**.** *For* Π*, the abstract non-ground rules are*

$$c(X) \leftarrow not\ d(X), dom(X). \tag{8}$$
$$\{c(X)\} \leftarrow not\ isSingleton(X), dom(X). \tag{9}$$
$$d(X) \leftarrow not\ c(X), dom(X). \tag{10}$$
$$\{d(X)\} \leftarrow not\ isSingleton(X), dom(X). \tag{11}$$
$$b(X,Y) \leftarrow a(X), d(Y). \tag{12}$$
$$e(X) \leftarrow c(X), a(Y), X \neq Y, \tau^{\neq}_{\mathrm{I}}(X,Y). \tag{13}$$
$$\{e(X)\} \leftarrow c(X), a(Y), X = Y, \tau^{\neq}_{\mathrm{IV}}(X,Y). \tag{14}$$
$$\bot \leftarrow b(X,Y), e(X_1), X = X_1, \tau^{=}_{\mathrm{I}}(X,Y). \tag{15}$$

For $m = \{\{1,\dots,5\} \mapsto k\}$*, the facts* $\{a(1), a(3)\}$ *in* Π *are lifted to* $\{a(k)\}$ *and the type facts are* $\mathcal{T}_m = \{\tau^{=}_{\mathrm{III}}(k,k), \tau^{\neq}_{\mathrm{IV}}(k,k)\}$*; the abstract program* Π^m *consists of the abstract rules and all these facts. As* $isSingleton(k)$ *is false, in view of* $\mathcal{T}_m$ *only the rules* (8)-(12) *and* (14) *matter for answer sets.*

Our method can also handle multiple relation atoms and multiple negative literals. The literals of Π involved in a negative cycle must be treated specially, where they get eliminated from the newly defined rule if they are involved in a negative cycle with the head of the rule.

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

Abstract answer sets $\hat{I}$ may not be of the form in Thm. 2:

Definition 4. *An abstract answer set* $\hat{I} \in AS(\Pi^m)$ *is* concrete *if some* $I \in AS(\Pi)$ *exists s.t.* $\hat{I} = m(I) \cup \mathcal{T}_m$*, else* spurious*.*

As for computational complexity, we show the following.

Theorem 3. *Deciding whether a given* $\hat{I} \in AS(\Pi^m)$ *of a program* Π *w.r.t. da-mapping* m *is concrete is* NEXP*-complete in general (resp.* Σ^p_2*-complete for bounded predicate arities).*

Intuitively, an abstract atom may map back to exponentially (resp. polynomially) many atoms in an answer set I of Π witnessing that $\hat{I}$ is concrete; such I can be guessed and verified in exponential time (resp. with an NP oracle).

4 Refinement by Debugging Spuriousness

Over-approximation of an answer set program unavoidably introduces spurious answer sets. Once a spurious abstract answer set is encountered, one can either continue searching for a concrete abstract answer set, or *refine* the abstraction to reach one where less spurious answer sets occur.

Definition 5. *Given da-mappings* $m_i \colon D \to \widehat{D}_i$*,* $i = 1, 2$*,* m_2 refines m_1 *if for every* $x \in D$*,* $m_2^{-1}(m_2(x)) \subseteq m_1^{-1}(m_1(x))$.

In the CEGAR methodology, the decision in a refinement step depends on the correctness checking of the spurious abstract solution, through which the problematic part of the abstraction is detected. Inspired by this, we develop an alternative for checking the correctness of abstract answer sets that can be used to determine how the refinement should be made. That an abstract answer set $\hat{I}$ is spurious means the original program Π has no answer set matching $\hat{I}$. In other words, querying Π for a match to an abstract answer set $\hat{I}$ would return no result exactly if $\hat{I}$ is spurious.

Definition 6. *Let* $\hat{I} \in AS(\Pi^m)$ *for a da-mapping* m*. A* query $Q^m_{\hat{I}}$ *for some* $I \in \Pi$ *that matches* $\hat{I}$ *are the following rules:*

$$\bot \leftarrow \{\alpha \mid m(\alpha) = \hat{\alpha}\} \leq 0. \quad \hat{\alpha} \in \hat{I} \setminus \mathcal{T}_m, \tag{16}$$
$$\bot \leftarrow \alpha. \quad \hat{\alpha} \notin \hat{I} \setminus \mathcal{T}_m, m(\alpha) = \hat{\alpha}. \tag{17}$$

The following is then easily established.

Proposition 4. *For any da-mapping* m *of a program* Π*,* $\hat{I} \in AS(\Pi^m)$ *is spurious iff* $\Pi \cup Q^m_{\hat{I}}$ *is unsatisfiable.*

As existing non-ground debugging tools are not readily applicable, we debug the unsatisfiability in Prop. 4 following the debugging approach based on [Brain *et al.*, 2007] natively. A simplified debugging approach used in [Saribatur *et al.*, 2019], however, cannot address all relevant cases; we thus show an extension of the refinement method by lifting the `spock` [Brain *et al.*, 2007] debugging approach to the non-ground case for a class of programs that subsumes tight programs.

5 Multi-Dimensional Domain Abstraction

The need for multi-dimensionality can be easily seen in grid-cell domains. An abstraction over the grid-cells would be to cluster rows and columns together to define an abstract grid-cell. As the approach above is extendible to *sorts*, i.e., subdomains, abstraction over rows and columns as sorts, one at a time, can achieve certain abstract cell structures. However, for more sophisticated abstractions, these sorts must be *jointly* abstracted. The abstraction method in Section 3.2 keeps built-ins in the abstract program and handles their different behavior in the abstract domain. However, this approach cannot be used to achieve the above mentioned multi-dimensional abstraction.

Existential Abstraction on Relations. We consider an alternative abstraction method that also abstracts over the built-in relations and reasons over the abstracted relation. This leads us to a notion of abstraction that is similar in spirit to so called existential abstraction [Clarke *et al.*, 2003] and allows us to introduce domain mappings over multiple sorts such as

$$m : D_1 \times \ldots \times D_n \to \widehat{D}_1 \times \ldots \times \widehat{D}_n,$$

and to handle relations over different levels of abstraction. For the existential abstraction $\Pi^m_\exists$ of a program Π, we then show:

Theorem 5. *Let m be a da-mapping of a program Π (standardized apart) whose rules have a single relation atom and no cyclic dependencies between non-ground literals. Then $AS(\Pi^m)$ and $AS(\Pi^m_\exists)$ coincide (modulo auxiliary atoms).*

Grid-cell environments are of a type that describes a structure. For problems over grid-cells, certain parts of the environment are often crucial for finding a solution. For a systematic refinement of abstractions on grid-cell environments, we consider a generic *quad-tree* representation, a concept used, e.g., in path planning [Kambhampati and Davis, 1986].

6 Implementation and Evaluation

We have implemented the methodology in the tools DASPAR and mDASPAR based on clingo 5.2.2, Python and the Ouroboros debugging tool [Oetsch *et al.*, 2010], whose Meta-Translator is exploited to obtain a reified program for which the debugging program is constructed. The tools are available at www.kr.tuwien.ac.at/research/systems/abstraction/.

Evaluation on Obtaining Abstract Solutions. The main aim of our evaluation was to see whether the domain abstraction and refinement method from above can find automatically non-trivial domain abstractions that yield concrete answer sets. We also studied the effect of variants of picking abstract answer sets or making refinement decisions.

The results show that with domain abstraction it is possible to achieve concrete solutions while abstracting over some of the details of the program. Reaching faithful abstractions is desired; however it does not occur often, unless a projected concreteness check is considered that only distinguishes the details relevant for a solution of the problem. Obtaining hints from multiple instead from a single abstract spurious answer set yields better decisions and thus coarser abstractions.

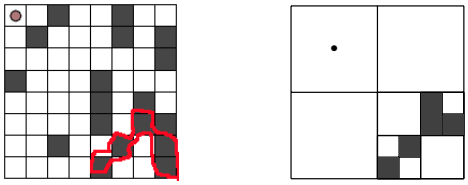

Figure 3: User explanation by marked obstacles (left) vs. mDASPAR abstraction (right) that preserves unsolvability of reaching each cell

Evaluation on Unsolvable Grid-Cells Problems. We considered explaining unsatisfiability of grid-cell problems by obtaining an abstraction that focuses on the troubling areas. Our interest was to see whether the abstractions match the intuition behind human explanations. We thus conducted a small user study to obtain the regions on which humans focus to realize unsolvability of a problem instance.

As a result, our abstraction method can exhibit a human-like focus on certain parts of the grid. Fig. 3 shows an instance where not every cell is reachable from the upper left corner, due to the obstacles surrounding the unreachable cells. However, humans implicitly also use background knowledge and need not to state all relations among objects explicitly. Empowering machines with such capabilities is challenging.

7 Conclusion

We have introduced domain abstraction to ASP, which shrinks the domain size while over-approximating the answer sets of an ASP program, and it preserves the rule structure, features multi-sorted domains, and can handle multi-dimensionality. One variant keeps built-in relations in rules, which then must be lifted to the abstract domain, while another (existential abstraction) loses their original format but can handle different levels of abstractions as needed for hierarchical abstraction. Using ASP debugging techniques, we further gave a method to obtain hints for refining abstractions to eliminate spurious abstract answer sets, along with a CEGAR-style methodology of iterated abstraction refinement. Experiments with our DASPAR and mDASPAR prototypes showed the potential of the approach for understanding the core parts of an ASP program.

The use of domain abstraction remains to be explored for applications. Different from common debugging techniques, domain abstraction aims to not just show the rules themselves that effect a certain behavior, but can moreover be used to identify the gist of the domain that is responsible for the latter and thus aids in gaining more insight into a program at hand. We believe that domain abstraction has potential for building systems that explain matters to a human end user, and thus can be a useful tool for realizing explainable AI. On the technical side, studying "good" abstractions, improving the abstraction and refinement methodology, and addressing scalability are relevant issues for future research.

References

[Brain *et al.*, 2007] Martin Brain, Martin Gebser, Jörg Pührer, Torsten Schaub, Hans Tompits, and Stefan Woltran. Debugging asp programs by means of asp. In *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2007)*, pages 31–43. Springer, 2007.

[Clarke *et al.*, 2003] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP 1988)*, volume 88, pages 1070–1080, 1988.

[Kambhampati and Davis, 1986] Subbarao Kambhampati and Larry Davis. Multiresolution path planning for mobile robots. *IEEE Journal on Robotics and Automation*, 2(3):135–145, 1986.

[Oetsch *et al.*, 2010] Johannes Oetsch, Jörg Pührer, and Hans Tompits. Catching the ouroboros: On debugging non-ground answer-set programs. *Theory and Practice of Logic Programming*, 10(4-6):513–529, 2010.

[Saribatur and Eiter, 2018] Zeynep G. Saribatur and Thomas Eiter. Omission-based abstraction for answer set programs. In *Proceedings of the 16th International Conference on Principles of Knowledge Representation and Reasoning (KR 2018)*, pages 42–51. AAAI Press, 2018.

[Saribatur *et al.*, 2019] Zeynep G. Saribatur, Peter Schüller, and Thomas Eiter. Abstraction for non-ground answer set programs. In *Proceedings of the 16th European Conference on Logics in Artificial Intelligence (JELIA 2019)*, Lecture Notes in Computer Science, pages 576–592. Springer, 2019.

[Saribatur *et al.*, 2021] Zeynep G. Saribatur, Thomas Eiter, and Peter Schüller. Abstraction for non-ground answer set programs. *Artif. Intell.*, 300:103563, 2021.