Towards Comprehensible ASP Reasoning by Means of Abstraction

Zeynep G. Saribatur

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

Abstract. The inability to understand the complex structures of the recently advanced AI systems urges for more symbolic and rule-based representations geared towards transparency in AI. Answer Set Programming (ASP), with its expressivity and representation power, is a convenient tool for problem-solving and for representing and reasoning about agent behavior. However, when shown the decision-making rules, it can be challenging for humans to get to the core of the behavior, if these rules are of complex nature or contain many details that are distracting. We highlight the potential of the recently introduced abstraction concept in ASP towards tackling this problem, by focusing on the relevant details of the reasoning task that can allow for better human comprehensibility.

1 Introduction

There is an increasing need to have an understanding on the behavior of designed AI agents. Especially with the advance of machine learning systems, obtaining explanations for the behavior is not easy, since these systems rely on highly complex and deep structures, which are far from intuition. So far, the focus of research with such systems has been mostly on *post-hoc interpretability*, that is on explaining the decisions, which can not show how the system actually works, and thus diverge from the goal to understand the behavior itself. This suggests putting more focus on adopting a knowledge representation and reasoning (KR) perspective into the need to understand the core elements of the designed model that plays a role in the behavior and the decision-making.

The expressivity and representation power makes Answer Set Programming (ASP) [8, 3] a convenient tool for problem-solving, with the idea of declaratively representing a problem as a "program" whose models (called "answer sets" [8]) correspond to the solutions of the problem; making it suitable for investigating ways in understanding the problem with its key elements. Studies in explaining how ASP programs find a solution (or none) to a problem have mainly focused on debugging answer sets or finding justifications (see [7] for a recent survey). These approaches can be used to understand the problem at hand and provide explanations, e.g., in planning [13]. However, as noted in [7], the obtained explanations may contain too many details which prevent one from seeing the crucial parts. This is where some notion of abstraction would come in handy.

Humans unwittingly make use of abstraction when reasoning and understanding. Among the several interpretations on its meaning, one that comes up is the capability of *abstract thinking*, achieved by removing irrelevant details and identifying the "essence" of the problem [10]. The notion of *relevance* is especially important in problem-solving, since the problem at hand may become too complex to solve if every detail is taken into account. Another view on abstraction is the *generalization* aspect, which is the process of distinguishing the common properties among the objects. Overall, the general aim of abstraction is to simplify the problem at hand to one that is easier to deal with and to understand.

Recently, we took the initial steps in bringing the notion of abstraction to ASP [14, 15, 6],[1] with the motivation to get rid of irrelevant details for reasoning and to get to the essence of the problems at hand. In this extended abstract, we give a brief overview of the newly introduced concepts and highlight the potential of the results in the comprehensibility of problem-solving in ASP.

2 Abstraction in ASP

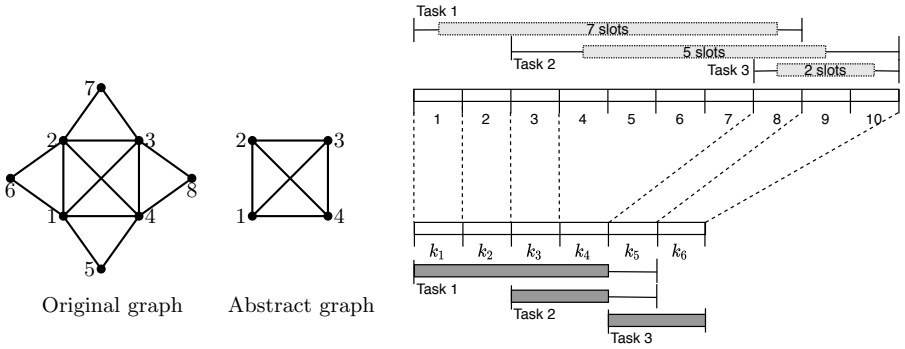
Abstraction is on *over-approximating* a given program Π, with vocabulary $\mathcal{A}$, by constructing a simpler program Π' with a vocabulary reduced to $\mathcal{A}'$, and ensuring that the results of reasoning on the original program are not lost. More formally, Π' is called an *abstraction* of Π, if there exists a mapping $m : \mathcal{A} \to \mathcal{A}'$, where $|\mathcal{A}| \geq |\mathcal{A}'|$, such that for any answer set I of Π, $I' = \{m(\alpha) \mid \alpha \in I\}$ is an answer set of Π'. We refer to m as an *abstraction mapping*, and it gives us the possibility to do clustering over the atoms of Π. The reduced vocabulary simplifies the search for an abstract answer set I' at the abstract program Π', while an additional check is needed to see whether an original answer set I exists in Π that can be mapped to I'. An abstract answer set I' is called *spurious* if it can not be mapped back to some I. Additionally, due to the over-approximation, we can infer that if the abstraction Π' is unsatisfiable (i.e., has no answer sets), then the original program Π is unsatisfiable. This property comes in handy when searching for the cause for unsatisfiability while abstracting away the rest.

We introduced methods to construct abstract programs for two kinds of abstraction mappings: abstraction by omission [14] and domain abstraction [15]. The former is about omitting atoms from a program, i.e., clustering them into $\top$, and considering an abstract program over the remaining atoms, while the latter investigates abstraction over non-ground ASP programs given a mapping over their domain (i.e., the Herbrand universe) that singles out the domain elements. In order to automatically obtain abstractions, we proposed a CEGAR-style [5] abstraction refinement approach (implemented in prototypical tools[2]), that starts with an initial highly coarse abstraction and then refines it upon encountering spurious answer sets and getting rid of them, until a non-spurious

[1] In collaboration with Thomas Eiter and Peter Schüller.

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

Fig. 1: Abstractions that remove details irrelevant to (un)solvability



(a) Non-3-colorable graph (on left) remains non-colorable when some nodes are omitted (on right)

(b) A valid task scheduling (for the instance shown on top) can be found even when some time slots are clustered (shown at the bottom)

answer set (or unsatisfiability) is encountered. The resulting abstraction then can be used to get an understanding of the problem at hand.

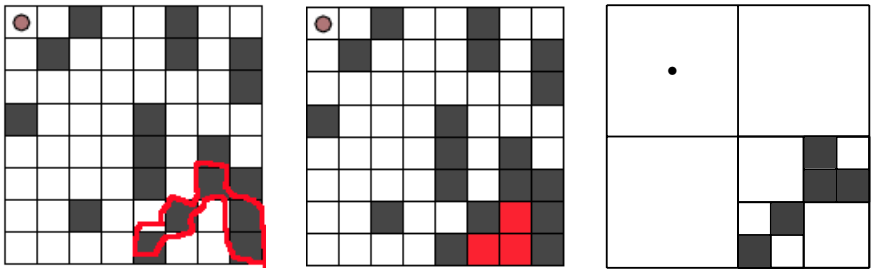
2.1 Removing irrelevant details

Determining relevance is important in understanding a problem with its key elements and abstraction can be a tool to distinguish the relevancy. For problems that are unsolvable, the relevant details of the problem would be the ones that cause the unsolvability. For example, if a graph is not 3-colorable, in order to understand the reason, one would focus on the nodes/edges that cause this and the remaining details would be irrelevant (Fig. 1(a)). Given the corresponding unsatisfiable answer set program, our methodology can automatically construct an abstract program where the atoms related to some nodes/edges are omitted from the vocabulary, while the unsatisfiability is still preserved [14]. The omitted parts of the graph can then be understood as irrelevant to the unsolvability.

Irrelevant details can also exist when solving a problem, since not every detail may need to be taken into account to find a solution. For example, consider a scheduling problem with tasks of certain durations that need to be scheduled within a given time interval, with additional constraints of not having slot intersections among some tasks. Fig. 1(b) shows an instance with 3 tasks, where task 1 and task 2 should not intersect with task 3. We can automatically find a domain abstraction [15] on the time domain that clusters some slots (e.g., slots $4-7$ clustered to k_4) where it is possible to find a schedule that satisfies the constraints and that can be mapped back to the original problem. This abstraction can be used to distinguish the key time slots for solving the problem.

As a possible application on grid-cell problems, we also considered a multi-dimensional domain abstraction to deal with structural aspects, and in particular with hierarchical abstraction over the domain [6]. We were interested to see

Fig. 2: User study on explanatory abstractions for unsatisfiable Reachability problem instances compared with the abstractions found by our tool [6]

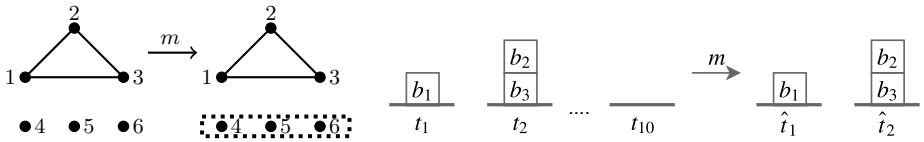


(a) an expected user explanation that marks the obstacles

(b) an unexpected user explanation that marks the cells

(c) abstraction on the grid-cell domain, found by the tool, that preserves unsatisfiability

Fig. 3: Abstractions that distinguish common properties



(a) 3-colorability

(b) Blocksworld with multiple tables

whether the automatically obtained abstractions from our tool matched the intuition behind a human explanation to unsolvability. Ten participants were given different instances of a grid-cell environment where not every cell was reachable from the starting point. They were asked to mark the area which shows the reason for having unreachable cells in these instances (e.g., Fig. 2(a)-(b)). The results showed the "zooming-in" capability of the abstraction method to have a human-like focus on certain parts of the grid to show the unsolvability reason (Fig. 2(c)). Interestingly, we also observed different understandings of *showing a reason* (e.g., the difference of markings in Fig. 2(a) and (b)) which acknowledged the challenge of defining a single meaning to "explanation".

2.2 Generalization

Distinguishing the common properties among the elements would obtain a higher-level view that simplifies the problem, making it easier to understand. Domain abstraction [15] gives us such an ability. For example, in a graph coloring problem the nodes with no edges can be considered as one node and assigned the same color (Fig. 3(a)). We can achieve an abstract program with such an abstraction on the graph, where any coloring at the abstract level will have a corresponding original coloring. We refer to such abstractions with no spurious answer sets as

faithful. These abstractions can be used for reasoning at the abstract level and for understanding the common properties of the clustered elements.

Generalization is especially appealing for planning problems, since it allows to compute a plan that can work for multiple instances of a problem. For example, consider the Blocksworld problem with multiple tables [15]. From a given initial state, the aim is to find a plan that piles up the blocks on a chosen table. Fig. 3(b) illustrates an initial state with t_1 being the chosen table. A faithful abstraction can be automatically found where all other tables are clustered into one, showing that distinguishing these tables is not important for the plan computation. No matter on which table the block b_3 is initially located, a plan of 4-steps, that moves b_2 to *some* table, moves b_3 to t_1, and moves b_2 and then b_1 on top of b_3, will be valid. Abstraction allows us to see this commonality of the objects.

3 Discussion

The newly introduced concept of abstraction in ASP shows potential to aid in achieving human-comprehensibility of problem-solving, which would contribute to bringing a KR perspective to the explainability and transparency of AI systems. The obtained results are encouraging and they open up avenues for future research, especially in terms of finding ways to make use of abstraction in order to help the users in understanding the decision-making behavior.

The concept of abstraction in KR is not new (e.g., [9, 12]). Especially for planning problems, reasoning at the abstract level to compute plans and delving into the details when necessary receives ongoing attention, e.g., [1, 2], also for ASP-related languages [17, 16]. However, how to find and decide on a good abstraction to do the reasoning over is a challenge by itself and usually the abstract representations are constructed by the researchers themselves, whereas our methodology can be used as a guide to distinguish the key elements of a problem and to decide on abstractions. Though, one can expect to have differences among the abstractions which are better for comprehensibility vs. for efficiency. Furthermore, there are some recent works in planning that make use of abstraction and focus on giving explanations for choosing a plan or for not finding a plan (see the recent survey [4]). Nevertheless, the potential of abstraction in achieving comprehensibility of rule-based programs remains to explored.

Lastly, as noted in the recent survey on explanation in AI [11], most of the recent research in explanation takes into account what researchers think as "good" explanation, but the human-angle should be considered as well. When providing explanations, humans implicitly make use of their background knowledge, do not need to explicitly state the relations among the objects, and thus achieve various levels of abstraction. Achieving such different abstraction layers in explanations remains an interesting challenge. Our results also acknowledged the need for studying the meaning of "explanation" and the ability to obtain an understanding of a problem using abstraction, when aiming to bring humans into the loop.

References

1. Banihashemi, B., De Giacomo, G., Lespérance, Y.: Abstraction in situation calculus action theories. In: Proc. AAAI. pp. 1048–1055 (2017)
2. Bercher, P., Alford, R., Höller, D.: A survey on hierarchical planning - one abstract idea, many concrete realizations. In: Proc. IJCAI. pp. 6267–6275 (2019)
3. Brewka, G., Eiter, T., Truszczyski, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011)
4. Chakraborti, T., Sreedharan, S., Kambhampati, S.: The emerging landscape of explainable automated planning & decision making. In: Proc. IJCAI. pp. 4803–4811 (2020)
5. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003)
6. Eiter, T., Saribatur, Z.G., Schüller, P.: Abstraction for zooming-in to unsolvability reasons of grid-cell problems. In: Proc. XAI@IJCAI (2019)
7. Fandinno, J., Schulz, C.: Answering the "why" in answer set programming - A survey of explanation approaches. TPLP **19**(2), 114–203 (2019)
8. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9**(3), 365–385 (1991)
9. Giunchiglia, F., Walsh, T.: A theory of abstraction. AIJ **57**(2-3), 323–389 (1992)
10. Kramer, J.: Is abstraction the key to computing? Commun. ACM **50**(4), 36–42 (2007)
11. Miller, T.: Explanation in artificial intelligence: Insights from the social sciences. AIJ (2018)
12. Nayak, P.P., Levy, A.Y.: A semantic theory of abstractions. In: Proc. IJCAI. pp. 196–203 (1995)
13. Nguyen, V., Son, T.C., Stylianos, V.L., Yeoh, W.: Conditional updates of answer set programming and its application in explainable planning. In: Proc. AAMAS. pp. 1954–1956 (2020)
14. Saribatur, Z.G., Eiter, T.: Omission-based abstraction for answer set programs. In: Proc. KR. pp. 42–51. AAAI Press (2018)
15. Saribatur, Z.G., Schüller, P., Eiter, T.: Abstraction for non-ground answer set programs. In: Proc. JELIA, pp. 576–592. LNCS, Springer (2019)
16. Sridharan, M., Gelfond, M., Zhang, S., Wyatt, J.: Reba: A refinement-based architecture for knowledge representation and reasoning in robotics. JAIR **65**, 87–180 (2019)
17. Zhang, S., Yang, F., Khandelwal, P., Stone, P.: Mobile robot planning using action language BC with an abstraction hierarchy. In: Proc. LPNMR. pp. 502–516. Springer (2015)