Towards Comprehensible ASP Reasoning by Means of Abstraction

Zeynep G. Saribatur

Institute of Logic and Computation
TU Wien, Vienna, Austria

XLoKR @ KR 2020

Answer Set Programming (ASP)

- A knowledge representation and reasoning paradigm widely used in problem solving.
  - Non-monotonic semantics, expressive power
  - Availability of efficient solvers (e.g., Clingo, WASP, DLV)
- Applications in many areas of AI including planning, diagnosis and commonsense reasoning.
- A convenient tool for investigating ways of applying human-inspired problem solving methods.

Background: ASP

- Syntax: rules of form

$$\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_m, not\ \alpha_{m+1}, \ldots, not\ \alpha_n, \quad \text{for} \quad 0 \leq m \leq n$$

Background: ASP

- Syntax: rules of form

$$\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_m, not\ \alpha_{m+1}, \ldots, not\ \alpha_n, \quad \text{for} \quad 0 \leq m \leq n$$

Example: Graph coloring

$color(red).\ \ color(green).\ \ color(blue).\ \ node(1 \ldots 6).$

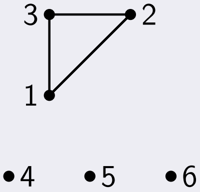
$\{chosenColor(N, C)\} \leftarrow node(N), color(C).$

$colored(N) \leftarrow chosenColor(N, C).$

$\bot \leftarrow not\ colored(N), node(N).$

$\bot \leftarrow chosenColor(N, C_1), chosenColor(N, C_2), C_1 \neq C_2.$

$\bot \leftarrow chosenColor(N_1, C), chosenColor(N_2, C), edge(N_1, N_2).$



Background: ASP

- Syntax: rules of form

$$\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_m, not\ \alpha_{m+1}, \ldots, not\ \alpha_n, \quad \text{for} \quad 0 \leq m \leq n$$

- Semantics:
  - Herbrand universe: constant symbols, Herbrand base: ground literals
  - Stable models [Gelfond and Lifschitz 1991]

Example: Graph coloring

$color(red).\ \ color(green).\ \ color(blue).\ \ node(1 \ldots 6).$

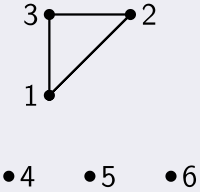
$\{chosenColor(N, C)\} \leftarrow node(N), color(C).$

$colored(N) \leftarrow chosenColor(N, C).$

$\bot \leftarrow not\ colored(N), node(N).$

$\bot \leftarrow chosenColor(N, C_1), chosenColor(N, C_2), C_1 \neq C_2.$

$\bot \leftarrow chosenColor(N_1, C), chosenColor(N_2, C), edge(N_1, N_2).$



Background: ASP

- Syntax: rules of form

$$\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_m, not\ \alpha_{m+1}, \ldots, not\ \alpha_n, \quad \text{for}\ \ 0 \leq m \leq n$$

- Semantics:
  - Herbrand universe: constant symbols, Herbrand base: ground literals
  - Stable models [Gelfond and Lifschitz 1991]

Example: Graph coloring

$color(red).\ \ color(green).\ \ color(blue).\ \ node(1 \ldots 6).$

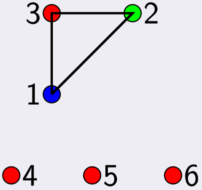
$\{chosenColor(N, C)\} \leftarrow node(N), color(C).$

$colored(N) \leftarrow chosenColor(N, C).$

$\bot \leftarrow not\ \ colored(N), node(N).$

$\bot \leftarrow chosenColor(N, C_1), chosenColor(N, C_2), C_1 \neq C_2.$

$\bot \leftarrow chosenColor(N_1, C), chosenColor(N_2, C), edge(N_1, N_2).$



Answer sets:

$\{chosenColor(1, blue), chosenColor(2, green), chosenColor(3, red), \ldots\}$

Background: ASP

- Syntax: rules of form

$$\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_m, not\ \alpha_{m+1}, \ldots, not\ \alpha_n, \quad \text{for} \ \ 0 \leq m \leq n$$

- Semantics:
  - Herbrand universe: constant symbols, Herbrand base: ground literals
  - Stable models [Gelfond and Lifschitz 1991]

Example: Graph coloring

$color(red).\ \ color(green).\ \ color(blue).\ \ node(1 \ldots 8).$

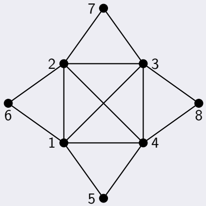
$\{chosenColor(N, C)\} \leftarrow node(N), color(C).$

$colored(N) \leftarrow chosenColor(N, C).$

$\bot \leftarrow not\ \ colored(N), node(N).$

$\bot \leftarrow chosenColor(N, C_1), chosenColor(N, C_2), C_1 \neq C_2.$

$\bot \leftarrow chosenColor(N_1, C), chosenColor(N_2, C), edge(N_1, N_2).$



No answer sets.

Possibilities of Comprehensible ASP

- Simplifications
  - Equivalence-based rewriting [Lifschitz *et al.* 2001, Osorio *et al.* 2002, Eiter *et al.* 2004, Pearce 2004, Eiter and Fink 2003]
  - Forgetting [Eiter and Kern-Isberner 2018, Gonçalves *et al.* 2017, Leite 2017]
- Verifying ASP programs [Lifschitz *et al.* 2020]
- Explanations (see survey [Fandinno and Schulz 2019])
  - Justifications [Pontelli *et al.* 2009, Cabalar *et al.* 2014]
  - Debugging [Brain *et al.* 2007, Gebser *et al.* 2008, Oetsch *et al.* 2010]

Possibilities of Comprehensible ASP

- Simplifications
  - Equivalence-based rewriting [Lifschitz *et al.* 2001, Osorio *et al.* 2002, Eiter *et al.* 2004, Pearce 2004, Eiter and Fink 2003]
  - Forgetting [Eiter and Kern-Isberner 2018, Gonçalves *et al.* 2017, Leite 2017]
- Verifying ASP programs [Lifschitz *et al.* 2020]
- Explanations (see survey [Fandinno and Schulz 2019])
  - Justifications [Pontelli *et al.* 2009, Cabalar *et al.* 2014]
  - Debugging [Brain *et al.* 2007, Gebser *et al.* 2008, Oetsch *et al.* 2010]

$\rightarrow$ The obtained explanations may contain too many details which prevent one from seeing the crucial parts.

Abstraction in KR

- It is commonly agreed that abstraction plays a key role in representing knowledge and in reasoning
  - By focusing on the key details and disregarding the rest
- Solve the problem in the abstract space, then guide the search for an original solution [Newell and Simon 1972, Sacerdoti 1974]
  - Abstraction heuristics [Edelkamp 2001, Helmert *et al.* 2007]
  - Hierarchical planning [Bercher *et al.* 2019]
- Desired properties for abstractions [Giunchiglia and Walsh 1992, Nayak and Levy 1995]
- Abstraction layers in ASP-related languages for robotics [Zhang *et al.* 2015, Sridharan *et al.* 2019]

Abstraction in ASP

- Constructing an over-approximation of a given program
  - through omitting atoms from the vocabulary, or
  - clustering the elements of the domain.
- CEGAR-inspired abstraction-&-refinement methodology
  - Automatically finding an abstraction that gives concrete solutions.
- Implemented prototypical tools and applied to several benchmarks.
- Resulting abstractions can be used to get an understanding of the problem at hand.

Outline

1. Background: Abstraction in ASP
2. Potential in Comprehensibility
   - Removing Irrelevant Details
   - Generalization
3. Conclusion

Outline

1. Background: Abstraction in ASP

2. Potential in Comprehensibility
   - Removing Irrelevant Details
   - Generalization

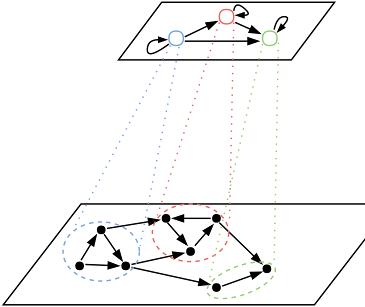
3. Conclusion

Abstraction

- Over-approximate the problem into a smaller or simpler state space.
    - Deliberately lose information.

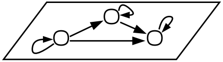
Abstraction in Model Checking

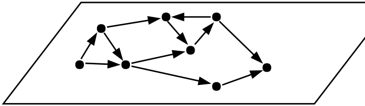
- Over-approximate the problem into a smaller or simpler state space.
  - Deliberately lose information.
- All original transitions must be preserved.
  - The abstract system can simulate the original system.
- Spurious transitions may be introduced.



Abstraction in Model Checking

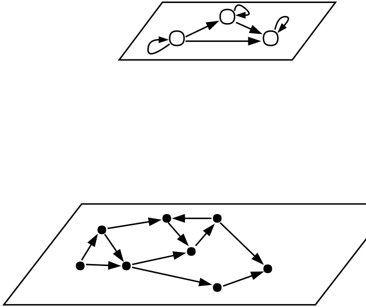
- Over-approximate the problem into a smaller or simpler state space.
  - Deliberately lose information.
- All original transitions must be preserved.
  - The abstract system can simulate the original system.
- Spurious transitions may be introduced.





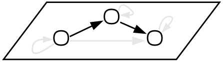
Abstraction in Model Checking

- Over-approximate the problem into a smaller or simpler state space.
  - Deliberately lose information.
- All original transitions must be preserved.
  - The abstract system can simulate the original system.
- Spurious transitions may be introduced.

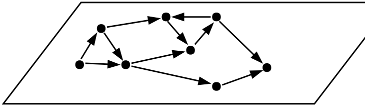


- Refinement of the abstraction is necessary in order to get rid of spurious transitions.
  - E.g., CEGAR method [Clarke *et al.* 2003]

Abstraction in Model Checking



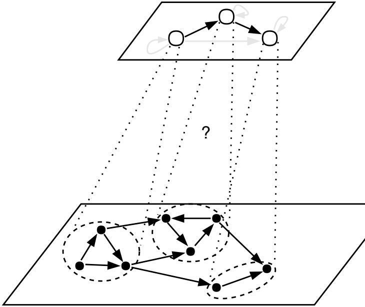
- Over-approximate the problem into a smaller or simpler state space.
    - Deliberately lose information.
- All original transitions must be preserved.
    - The abstract system can simulate the original system.
- Spurious transitions may be introduced.



- Refinement of the abstraction is necessary in order to get rid of spurious transitions.
    - E.g., CEGAR method [Clarke *et al.* 2003]

Abstraction in Model Checking

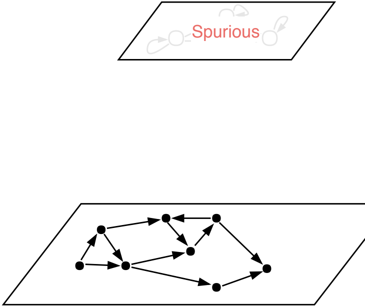
- Over-approximate the problem into a smaller or simpler state space.
    - Deliberately lose information.
- All original transitions must be preserved.
    - The abstract system can simulate the original system.
- Spurious transitions may be introduced.



- Refinement of the abstraction is necessary in order to get rid of spurious transitions.
    - E.g., CEGAR method [Clarke *et al.* 2003]

Abstraction in Model Checking

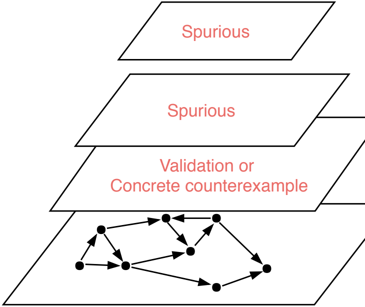
- Over-approximate the problem into a smaller or simpler state space.
  - Deliberately lose information.
- All original transitions must be preserved.
  - The abstract system can simulate the original system.
- Spurious transitions may be introduced.



- Refinement of the abstraction is necessary in order to get rid of spurious transitions.
  - E.g., CEGAR method [Clarke *et al.* 2003]

Abstraction in Model Checking

- Over-approximate the problem into a smaller or simpler state space.
  - Deliberately lose information.
- All original transitions must be preserved.
  - The abstract system can simulate the original system.
- Spurious transitions may be introduced.



- Refinement of the abstraction is necessary in order to get rid of spurious transitions.
  - E.g., CEGAR method [Clarke *et al.* 2003]

Abstraction of Answer Set Programs

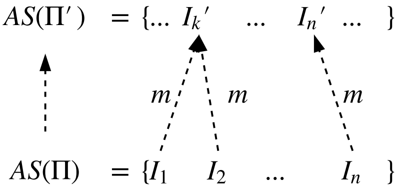
Definition

Π' is an abstraction of Π, where $|\mathcal{A}| \geq |\mathcal{A}'|$, if there exists a mapping $m : \mathcal{A} \rightarrow \mathcal{A}' \cup \{\top\}$ s.t. for any answer set I of Π, $I' = \{m(\alpha) \mid \alpha \in I\}$ is an answer set of Π'.

Abstraction of Answer Set Programs

Definition

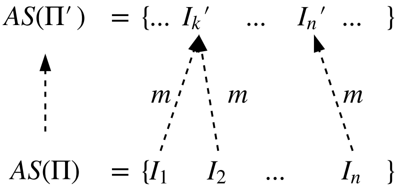
Π' is an abstraction of Π, where $|\mathcal{A}| \geq |\mathcal{A}'|$, if there exists a mapping $m : \mathcal{A} \to \mathcal{A}' \cup \{\top\}$ s.t. for any answer set I of Π, $I' = \{m(\alpha) \mid \alpha \in I\}$ is an answer set of Π'.



Abstraction of Answer Set Programs

Definition

Π' is an abstraction of Π, where $|\mathcal{A}| \geq |\mathcal{A}'|$, if there exists a mapping $m : \mathcal{A} \rightarrow \mathcal{A}' \cup \{\top\}$ s.t. for any answer set I of Π, $I' = \{m(\alpha) \mid \alpha \in I\}$ is an answer set of Π'.

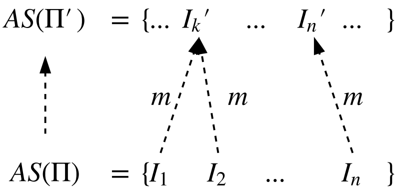


- Size of $I' \leq$ size of I.
- Spurious I' may exist that cannot be mapped back to some I.

Abstraction of Answer Set Programs

Definition

Π' is an abstraction of Π, where $|\mathcal{A}| \geq |\mathcal{A}'|$, if there exists a mapping $m : \mathcal{A} \rightarrow \mathcal{A}' \cup \{\top\}$ s.t. for any answer set I of Π, $I' = \{m(\alpha) \mid \alpha \in I\}$ is an answer set of Π'.



- Size of $I' \leq$ size of I.
- Spurious I' may exist that cannot be mapped back to some I.

- Π' is faithful if it has no spurious answer sets $\rightarrow m(AS(\Pi)) = AS(\Pi')$

Abstraction of Answer Set Programs

Definition

Π' is an abstraction of Π, where $|\mathcal{A}| \geq |\mathcal{A}'|$, if there exists a mapping $m : \mathcal{A} \rightarrow \mathcal{A}' \cup \{\top\}$ s.t. for any answer set I of Π, $I' = \{m(\alpha) \mid \alpha \in I\}$ is an answer set of Π'.

$$AS(\Pi') = \emptyset$$

$$\Downarrow$$

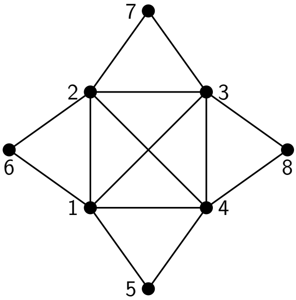
$$AS(\Pi) = \emptyset$$

- Size of $I' \leq$ size of I.
- Spurious I' may exist that cannot be mapped back to some I.
- If no I' exists in Π', then no I exists in Π.

- Π' is faithful if it has no spurious answer sets $\rightarrow m(AS(\Pi)) = AS(\Pi')$

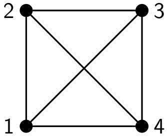
Example: Graph Coloring

- A non-3-colorable graph



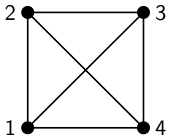
Example: Graph Coloring

- A non-3-colorable graph



Example: Graph Coloring

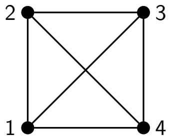
- A non-3-colorable graph



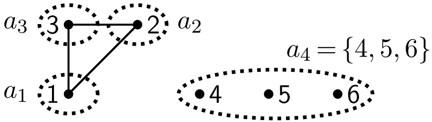
- 3-coloring of a graph

Example: Graph Coloring

- A non-3-colorable graph

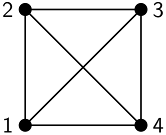


- 3-coloring of a graph

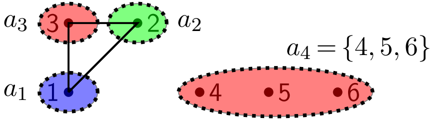


Example: Graph Coloring

- A non-3-colorable graph

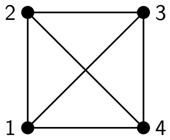


- 3-coloring of a graph

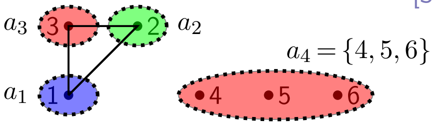


Example: Graph Coloring

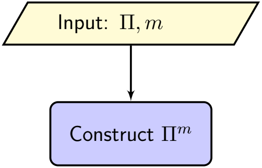
- A non-3-colorable graph → Omission abstraction [Saribatur and Eiter 2020]



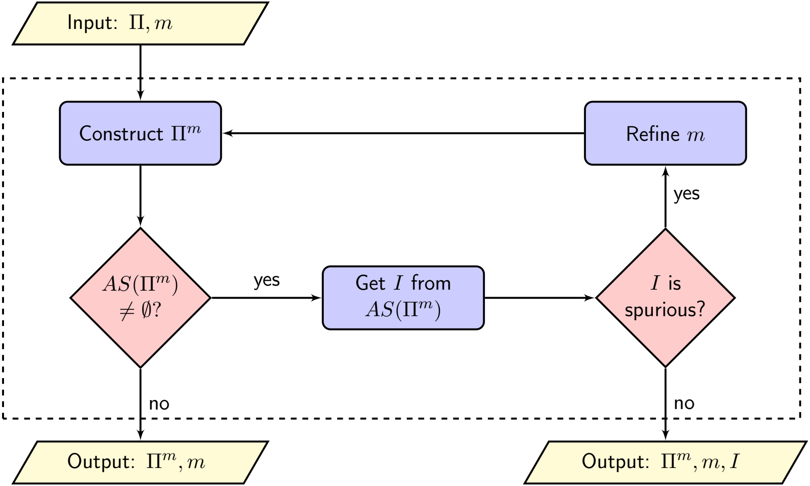
- 3-coloring of a graph → Domain abstraction [Saribatur *et al.* 2019]



Abstraction and Refinement Methodology



Abstraction and Refinement Methodology

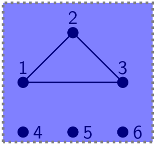


Example: Graph Coloring

- Adding back omitted nodes

→

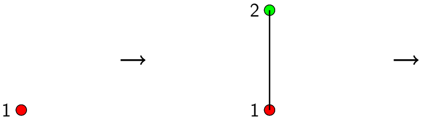
- Dividing abstract node clusters



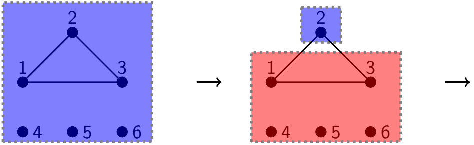
→

Example: Graph Coloring

- Adding back omitted nodes

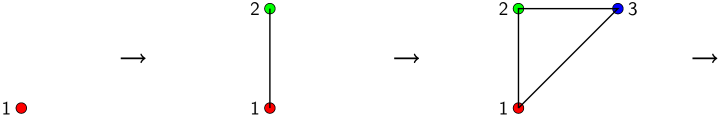


- Dividing abstract node clusters

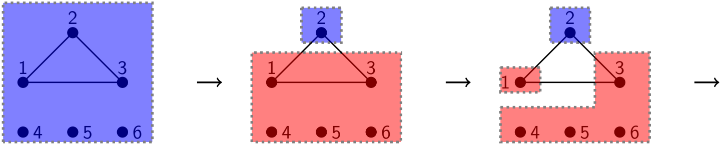


Example: Graph Coloring

- Adding back omitted nodes

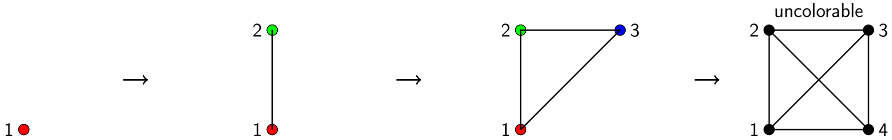


- Dividing abstract node clusters

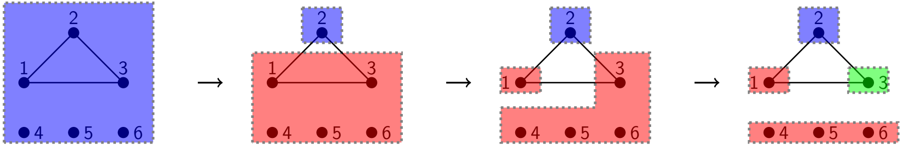


Example: Graph Coloring

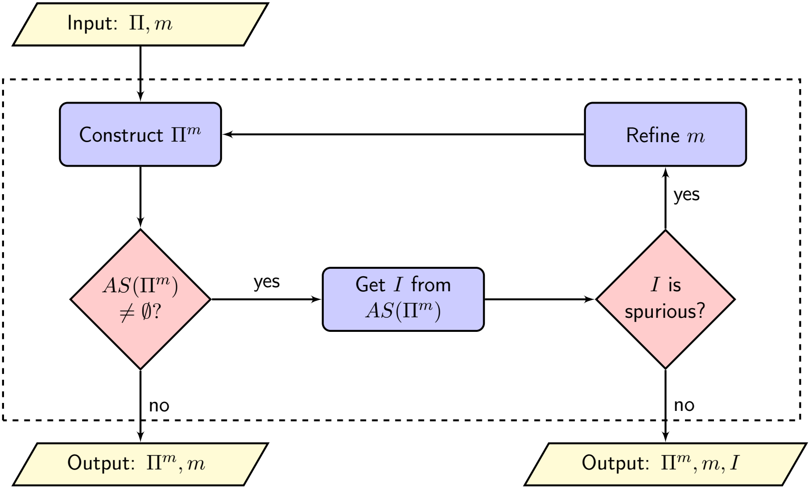
- Adding back omitted nodes



- Dividing abstract node clusters



Abstraction and Refinement Methodology



Outline

1. Background: Abstraction in ASP

2. Potential in Comprehensibility
   - Removing Irrelevant Details
   - Generalization

3. Conclusion

How do Humans use Abstraction?

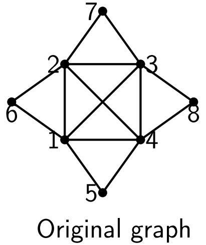
- Abstract Thinking: Removing irrelevant details and identifying the "essence" of the problem [Johnson-Laird 1983, Kramer 2007]
- Approaches to tackle large and complex structures
  - Determining the relevance
  - Distinguishing the common properties among the objects

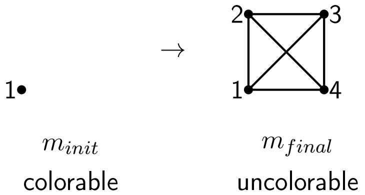
Outline

1. Background: Abstraction in ASP
2. Potential in Comprehensibility
   - Removing Irrelevant Details
   - Generalization
3. Conclusion

Removing Irrelevant Details

Satisfiability blockers





- Focusing on omission abstraction
- $C \subseteq \mathcal{A}$ is an (answer set) blocker set of Π, if $AS(omit(\Pi, \mathcal{A} \setminus C)) = \emptyset$
- $\subseteq$-minimal blocker set → *most relevant* part of the unsat. program

Removing Irrelevant Details

Satisfiability blockers

Initial State

0	3	2
8	5	4
1	6	7

Goal State

0	1	2
3	4	5
6	7	8

- Focusing on omission abstraction
- $C \subseteq \mathcal{A}$ is an (answer set) blocker set of Π, if $AS(omit(\Pi, \mathcal{A} \setminus C)) = \emptyset$
- $\subseteq$-minimal blocker set $\rightarrow$ *most relevant* part of the unsat. program

Removing Irrelevant Details

Satisfiability blockers

Initial State

*	*	*
*	*	*
*	*	*

Goal State

*	*	*
*	*	*
*	*	*

m_{init}

Initial State

0	3	*
*	5	4
*	*	*

Goal State

0	*	*
3	4	5
*	*	*

m_{final}

- Focusing on omission abstraction
- $C \subseteq \mathcal{A}$ is an (answer set) blocker set of Π, if $AS(omit(\Pi, \mathcal{A} \setminus C)) = \emptyset$
- $\subseteq$-minimal blocker set $\rightarrow$ *most relevant* part of the unsat. program

Removing Irrelevant Details

Satisfiability blockers

Initial State

*	*	*
*	*	*
*	*	*

Goal State

*	*	*
*	*	*
*	*	*

m_{init}

Initial State

0	3	*
*	5	4
*	*	*

Goal State

0	*	*
3	4	5
*	*	*

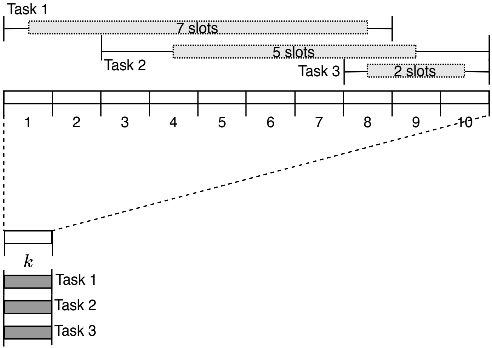
m_{final}

- Focusing on omission abstraction
- $C \subseteq \mathcal{A}$ is an (answer set) blocker set of Π, if $AS(omit(\Pi, \mathcal{A} \setminus C)) = \emptyset$
- $\subseteq$-minimal blocker set $\rightarrow$ *most relevant* part of the unsat. program
- E.g., explain non-acceptability of arguments in abstract argumentation [Saribatur *et al.* 2020]

Removing Irrelevant Details

Finding only relevant details for solvability

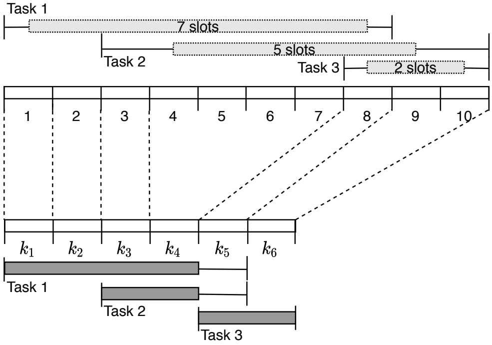
- Focusing on domain abstraction
- E.g., distinguishing the key time slots in scheduling



Removing Irrelevant Details

Finding only relevant details for solvability

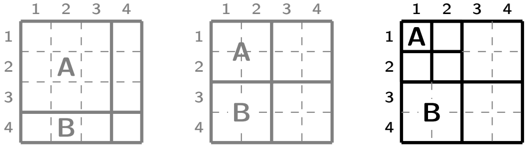
- Focusing on domain abstraction
- E.g., distinguishing the key time slots in scheduling



Removing Irrelevant Details

Finding only relevant details for unsolvability

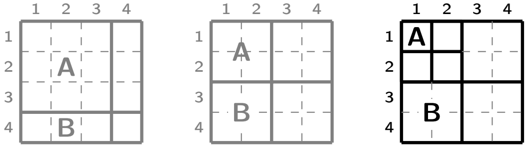
- Focusing on multi-dimensional domain abstraction [Eiter *et al.* 2019]
  - Achieve a hierarchical abstraction over the domain for *zooming-in*



Removing Irrelevant Details

Finding only relevant details for unsolvability

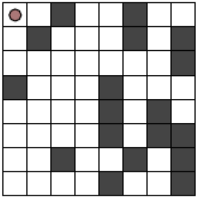
- Focusing on multi-dimensional domain abstraction [Eiter *et al.* 2019]
  - Achieve a hierarchical abstraction over the domain for *zooming-in*

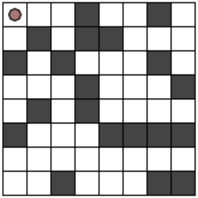


- Can the automatically obtained abstractions match the intuition behind a human explanation to unsolvability?

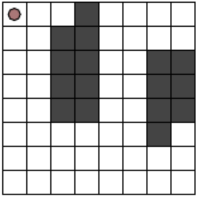
User Study on Unsatisfiable Grid-cell Problems

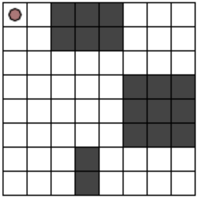
- Reachability: Mark the area which shows *the reason for having unreachable cells*



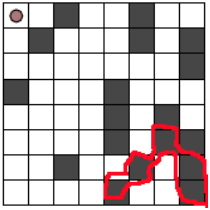


- Visitall: Mark the area which shows *the reason for not finding a solution* that visits all the cells

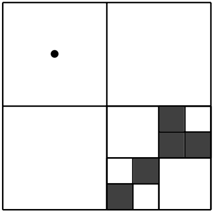




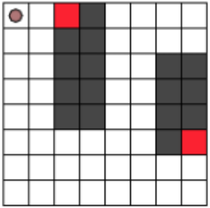
User Study Results for Reachability and Visitall (1/2)



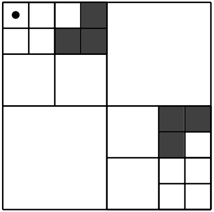
(a) #6 : expected



(c) #6 - mDASPAR

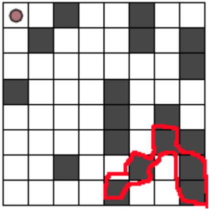


(d) #1: expected

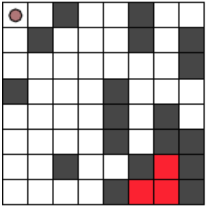


(f) #1 - mDASPAR

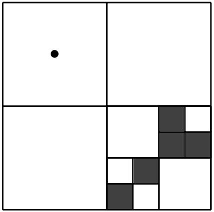
User Study Results for Reachability and Visitall (1/2)



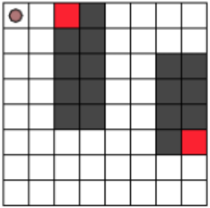
(a) #6 : expected



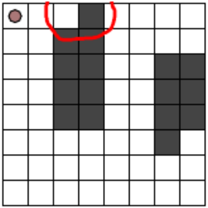
(b) #6 : unexpected



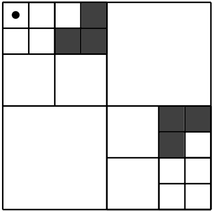
(c) #6 - mDASPAR



(d) #1: expected

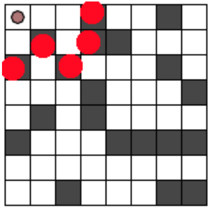


(e) #1: unexpected

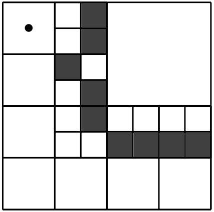


(f) #1 - mDASPAR

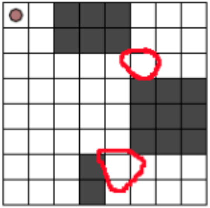
User Study Results for Reachability and Visitall (2/2)



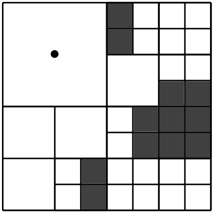
(a) #10: expected



(c) #10 - mDASPAR

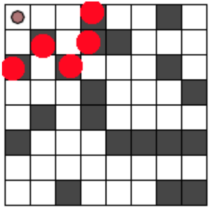


(d) #10: expected

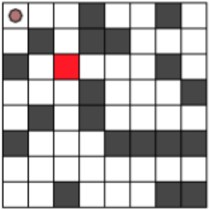


(f) #10 - mDASPAR

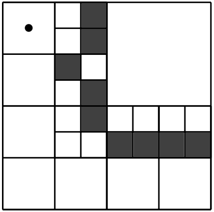
User Study Results for Reachability and Visitall (2/2)



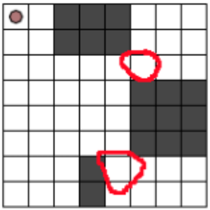
(a) #10: expected



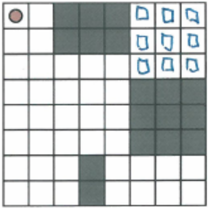
(b) #10: unexpected



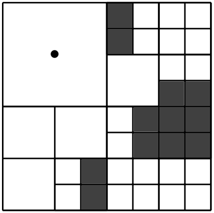
(c) #10 - mDASPAR



(d) #10: expected



(e) #10: unexpected



(f) #10 - mDASPAR

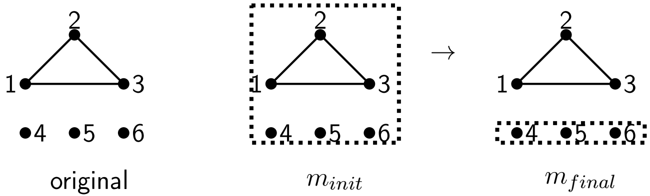
Outline

1 Background: Abstraction in ASP

2 Potential in Comprehensibility
- Removing Irrelevant Details
- Generalization

3 Conclusion

Generalization

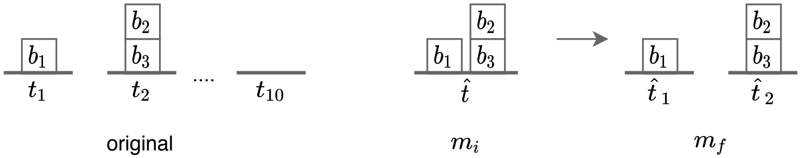


m_{init}
spurious colorings

m_{final}
no spurious colorings

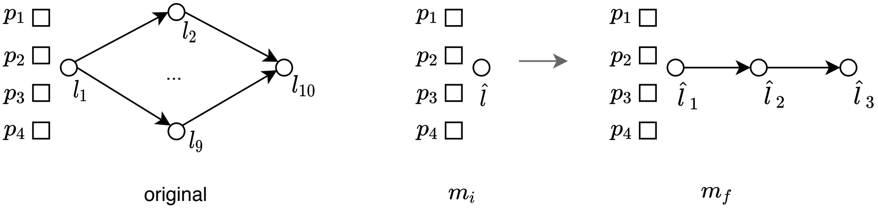
- Make use of faithful abstractions to reason at the abstract level by distinguishing the common properties of the clustered elements.

Generalization



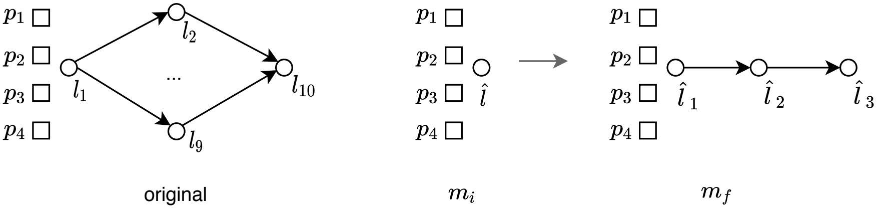
- Make use of faithful abstractions to reason at the abstract level by distinguishing the common properties of the clustered elements.

Generalization



- Make use of faithful abstractions to reason at the abstract level by distinguishing the common properties of the clustered elements.

Generalization



- Make use of faithful abstractions to reason at the abstract level by distinguishing the common properties of the clustered elements.
- Relation to generalized planning [Srivastava *et al.* 2011], [Illanes and McIlraith 2019] remains to be investigated.

Outline

1. Background: Abstraction in ASP
2. Potential in Comprehensibility
   - Removing Irrelevant Details
   - Generalization
3. Conclusion

Conclusion

- Abstraction shows potential in finding the "essense" of problem solving in ASP, useful for human-comprehensibility.
- We have an automated way of starting with an initial abstraction and achieving an abstraction with a concrete answer.
- Demonstrates a human-like focus to the key elements in the problem.
- Can be used as a guide to decide on good abstractions for reasoning.

Challenges:

- Finding ways to make use of such abstractions to help the users in understanding the decision-making behavior.
- Achieving the various levels of abstraction that humans unwittingly use.

References I

Pascal Bercher, Ron Alford, and Daniel Höller.
A survey on hierarchical planning - one abstract idea, many concrete realizations.
In *Proc. IJCAI*, pages 6267–6275, 2019.

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.

Pedro Cabalar, Jorge Fandinno, and Michael Fink.
Causal graph justifications of logic programs.
Theory and Practice of Logic Programming (TC), 14(4-5):603–618, 2014.

References II

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.

Stefan Edelkamp.
Planning with pattern databases.
In *Proceedings of the 6th European Conference on Planning (ECP 2001)*, pages 13–24, 2001.

Thomas Eiter and Michael Fink.
Uniform equivalence of logic programs under the stable model semantics.
In *International Conference on Logic Programming*, pages 224–238. Springer, 2003.

References III

- Thomas Eiter and Gabriele Kern-Isberner.
  A brief survey on forgetting from a knowledge representation and reasoning perspective.
  KI – Künstliche Intelligenz, 33(1):9–33, 2018.

- Thomas Eiter, Michael Fink, Hans Tompits, and Stefan Woltran.
  Simplifying logic programs under uniform and strong equivalence.
  In Vladimir Lifschitz and Ilkka Niemelä, editors, *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2004)*, pages 87–99. Springer, 2004.

- Thomas Eiter, Zeynep G. Saribatur, and Peter Schüller.
  Abstraction for zooming-in to unsolvability reasons of grid-cell problems.
  In *Proc. XAI@IJCAI*, 2019.

References IV

Jorge Fandinno and Claudia Schulz.
Answering the "why" in answer set programming - A survey of explanation approaches.
Theory and Practice of Logic Programming, 19(2):114–203, 2019.

Martin Gebser, Jörg Pührer, Torsten Schaub, and Hans Tompits.
A meta-programming technique for debugging answer-set programs.
In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008)*, volume 8, pages 448–453, 2008.

Michael Gelfond and Vladimir Lifschitz.
Classical negation in logic programs and disjunctive databases.
New Generation Computing, 9(3):365–385, 1991.

Fausto Giunchiglia and Toby Walsh.
A theory of abstraction.
Artificial Intelligence, 57(2-3):323–389, 1992.

References V

Ricardo Gonçalves, Matthias Knorr, João Leite, and Stefan Woltran.
When you must forget: Beyond strong persistence when forgetting in answer set programming.
Theory and Practice of Logic Programming, 17(5-6):837–854, 2017.

Malte Helmert, Patrik Haslum, Jörg Hoffmann, et al.
Flexible abstraction heuristics for optimal sequential planning.
In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007)*, pages 176–183, 2007.

León Illanes and Sheila A. McIlraith.
Generalized planning via abstraction: Arbitrary numbers of objects.
In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, 2019.

References VI

- Philip Nicholas Johnson-Laird.
  Mental models: Towards a cognitive science of language, inference, and consciousness.
  Number 6. Harvard University Press, 1983.
- Jeff Kramer.
  Is abstraction the key to computing?
  Communications of the ACM, 50(4):36–42, 2007.
- João Leite.
  A bird's-eye view of forgetting in answer-set programming.
  In Marcello Balduccini and Tomi Janhunen, editors, *Proceedings of the 14th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2017)*, volume 10377 of *Lecture Notes in Computer Science*, pages 10–22. Springer, 2017.

References VII

- Vladimir Lifschitz, David Pearce, and Agustín Valverde.
  Strongly equivalent logic programs.
  ACM Transactions on Computational Logic, 2(4):526–541, October 2001.

- Vladimir Lifschitz, Patrick Lühne, and Torsten Schaub.
  Towards verifying logic programs in the input language of clingo.
  In *Fields of Logic and Computation III*, pages 190–209. Springer, 2020.

- P. Pandurang Nayak and Alon Y. Levy.
  A semantic theory of abstractions.
  In *Proceedings of the 14th International Joint conference on Artificial intelligence (IJCAI 1995)*, pages 196–203, 1995.

- Allen Newell and Herbert A. Simon.
  Human problem solving, volume 104.
  Prentice-Hall Englewood Cliffs, NJ, 1972.

- 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.

- Mauricio Osorio, Juan A. Navarro, and José Arrazola.
  Equivalence in answer set programming.
  In Alberto Pettorossi, editor, *Logic Based Program Synthesis and Transformation*, pages 57–75. Springer Berlin Heidelberg, 2002.

- David Pearce.
  Simplifying logic programs under answer set semantics.
  In Bart Demoen and Vladimir Lifschitz, editors, *Logic Programming*, pages 210–224, 2004.

References IX

- Enrico Pontelli, Tran Cao Son, and Omar Elkhatib.
  Justifications for logic programs under answer set semantics.
  Theory and Practice of Logic Programming, 9(1):1–56, 2009.
- Earl D. Sacerdoti.
  Planning in a hierarchy of abstraction spaces.
  Artificial Intelligence, 5(2):115–135, 1974.
- Zeynep G. Saribatur and Thomas Eiter.
  Omission-based abstraction for answer set programs.
  TPLP, 2020.
  To appear. Available at `https://arxiv.org/abs/2004.01410`.

References X

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.

Zeynep G Saribatur, Johannes P Wallner, and Stefan Woltran.
Explaining non-acceptability in abstract argumentation.
In *Proc. ECAI*, 2020.

Mohan Sridharan, Michael Gelfond, Shiqi Zhang, and Jeremy Wyatt.
Reba: A refinement-based architecture for knowledge representation and reasoning in robotics.
JAIR, 65:87–180, 2019.

References XI

Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein.
A new representation and associated algorithms for generalized planning.
Artificial Intelligence, 175(2):615–647, 2011.

Shiqi Zhang, Fangkai Yang, Piyush Khandelwal, and Peter Stone.
Mobile robot planning using action language BC with an abstraction hierarchy.
In *Proceedings of the 13th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2015)*, pages 502–516. Springer, 2015.