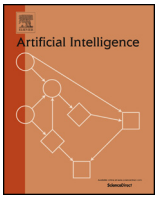
Artificial Intelligence 302 (2022) 103608

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint



A framework for analysing state-abstraction methods

Christer Bäckström *,[1], Peter Jonsson *

Linköping University, Sweden

ARTICLE INFO

Article history:
Received 17 December 2020
Received in revised form 30 September 2021
Accepted 6 October 2021
Available online 13 October 2021

Keywords:
Action planning
Combinatorial search
Heuristic search
Hierarchical abstraction
Refinement
State abstraction

ABSTRACT

Abstraction has been used in combinatorial search and action planning from the very beginning of AI. Many different methods and formalisms for state abstraction have been proposed in the literature, but they have been designed from various points of view and with varying purposes. Hence, these methods have been notoriously difficult to analyse and compare in a structured way. In order to improve upon this situation, we present a coherent and flexible framework for modelling abstraction (and abstraction-like) methods based on graph transformations. The usefulness of the framework is demonstrated by applying it to problems in both search and planning. We model six different abstraction methods from the planning literature and analyse their intrinsic properties. We show how to capture many search abstraction concepts (such as avoiding backtracking between levels) and how to put them into a broader context. We also use the framework to identify and investigate connections between refinement and heuristics—two concepts that have usually been considered as unrelated in the literature. This provides new insights into various topics, e.g. Valtorta's theorem and spurious states. We finally extend the framework with composition of transformations to accommodate for abstraction hierarchies, and other multi-level concepts. We demonstrate the latter by modelling and analysing the merge-and-shrink abstraction method.

1. Introduction

The main idea behind abstraction in problem solving is the following: the original problem instance is transformed into a corresponding abstract instance, this abstract instance is solved and the abstract solution is then used to find a solution to the original instance. The use of abstraction is an old and widespread idea both in AI and in computer science in general. One of the most well-known examples is abstract interpretation of computer programs [21,22]. Abstraction is also used in many other areas, for instance, automated verification, constraint satisfaction, databases, knowledge representation, geographical information systems, planning, agent-based modelling, natural language processing, vision, networks and model-based diagnosis [80]. The literature is consequently vast and we refer the reader to the surveys by Giunchiglia et al. [38], Holte and Choueiry [57] or Zucker [93], and to the book by Saitta and Zucker [80] for a broader and more extensive treatment of the topic.

The focus of this article is on abstraction in action planning and combinatorial search within AI. Abstraction has a long history even if we restrict ourselves in this way; its use dates back to the Abstrips planner [78] and even to the first version

* Corresponding authors.
E-mail addresses: christer.backstrom@liu.se (C. Bäckström), peter.jonsson@liu.se (P. Jonsson).

[1] The work of C. Bäckström was partially supported by the Swedish Research Council (VR) under grant 621-2014-4086.

https://doi.org/10.1016/j.artint.2021.103608

of the General Problem Solver (GPS), which used abstraction in the means-ends analysis [74]. Holte and Choueiry [57] say:

> The earliest and most widespread and successful use of abstraction in AI is in planning and problem solving.

Although our presentation and results have a focus on this area, almost all of our basic theory and our results is generally applicable elsewhere, with or without modification. Hence, we will talk about transformations, rather than abstractions, in the technical part of this article in order to encompass also abstraction-like methods such as reformulation and approximation.

1.1. Abstraction in search and planning

Combinatorial search and action planning are essentially the same problem. Given a description of a *state space* and the possible *transitions*, we ask for a *path* (a *plan*) from some initial state to some goal state. However, contrary to the ordinary graph-searching problem, the graph is usually implicitly specified; the state space is induced by a number of variables, and can thus be of exponential size in the number of variables and their domains, and the transitions are induced by operators (or actions), that may each correspond to an exponential number of transitions. A search (or planning) instance is thus a compact graph representation in the sense of Galperin and Wigderson [35] or Balcázar [12]. Hence, ordinary shortest-path algorithms, like Dijkstra's algorithm, are usually not good enough, so other techniques like abstraction and heuristic search are used to improve the efficiency by exploiting structure in the instance. In some cases, the state spaces are so large that disk-based search is necessary [69]. From a complexity point of view, planning is known to be PSPACE-complete both for variables with binary domains [17] and for variables with arbitrary finite domain [11]. Length-optimal planning is also known to be W[2]-complete under parameterised analysis, using plan length as parameter [8].

Our focus is on *state abstraction*, which is one of the most widespread and important forms of abstraction in planning and search.[2] In general terms, we start with an original instance to solve, the *ground instance*, and create a corresponding *abstract instance* by abstracting the state space. The idea is to first solve this abstract instance and then exploit the abstract solution to solve the ground instance more efficiently.

Two methods dominate for exploiting state abstractions: *abstraction refinement* and *abstraction-based heuristics*. Both approaches will be briefly discussed below. In both cases, it is also common to build hierarchies of abstractions and solve the problems top down, starting on the most abstract level; a method which dates back to Sacerdoti [78]. There are good reasons to consider hierarchical abstractions: Korf [68] showed that hierarchical abstraction can reduce the search space exponentially under ideal assumptions about the relationships between the abstraction layers. The abstraction refinement and abstraction heuristic approaches are usually considered as quite different and unrelated. That is a very superficial analysis, though: both approaches are firmly based on properties of the solutions in the abstract space, so some connections are bound to exist. Yet, the literature is almost void of attempts to investigate these connections; a notable exception can be found in a paper by Holte et al. [60].

1.1.1. Abstraction refinement

In abstraction refinement we first solve the abstract version of the instance, and then refine this into a solution for the original ground instance. One way to do this, that has been common in planning (cf. the articles by Knoblock [65] and Bacchus and Yang [3]), is to map the actions of the abstract plan into a corresponding sequence of ground actions. This is usually not a valid plan for the ground instance, but it can be used as a 'skeleton plan' where we can insert more actions until we get a correct plan for the ground instance. We refer to this method as *label refinement*. Another way, which has been common in search (cf. [59] and [91]), is to instead map the sequence of states along the abstract solution back to a corresponding sequence of ground states. Then we use these ground states as subgoals, thus turning the original problem into a sequence of subproblems which we solve directly in the ground instance. We refer to this method as *state refinement*.

In order for abstraction to be useful, the abstract instance must be easier to solve and the total time spent should be less than without using abstraction. This is a reasonable requirement, yet it has turned out very difficult to guarantee. Abstraction refinement can give huge savings in solution time under ideal circumstances, both for label refinement [64] and for state refinement [59]. However, even if we find an abstract solution, there is in general no guarantee that it can be refined into a ground solution even when one exists. It may be necessary to give up refining it, backtrack to the abstract instance and look for another abstract solution that we can try to refine instead. In unfortunate cases, this process of repeatedly backtracking to higher levels may take much more time than solving the original instance directly. This phenomenon is known as the problem of *backtracking between levels*. It has been a very active research area within planning and various methods have been proposed to remedy the situation. Partial solutions that have been presented include the ordered monotonicity criterion [66], the downward refinement property (DRP) [3], and the simulation-based approach by Bundy

[2] The other major abstraction method is *Hierarchical Task Networks (HTN)*, which originates from the Noah [79] and Nonlin [89] planners. It is based on a hierarchy of *methods* that can be refined by predefined expansion patterns, and it is fundamentally different from state abstraction.

et al. [16]. However, ordered monotonicity is not sufficient to prevent heavy backtracking between levels [86], and even when properties like the DRP guarantee that no such backtracking is needed, the abstraction hierarchy might still force the generation of exponentially suboptimal ground solutions in the worst case [5].

1.1.2. Abstraction-based heuristics

Historically, abstraction refinement was the dominating abstraction method in planning, but it has been largely replaced by *abstraction heuristics*. The heuristic search approach has proven very successful both in planning and in search, and a large number of well-performing domain-independent heuristics have been invented (see the paper by Helmert and Domshlak [47] for a comprehensive survey and comparison). In abstraction heuristics, the abstract solution is not used directly, but the length (or cost) of it is used as a heuristic value to estimate the length (or cost) of an optimal solution in the ground instance. The abstract solutions are thus only used indirectly to guide a heuristic search algorithm. This method dates back to Gaschnig [36] and to Guida and Somalvico [43].

Also the abstraction heuristics method suffers from problems. It was proven by Valtorta [90] that this method cannot improve upon heuristic search directly in the ground state space if the abstraction is an embedding, which was common at the time. However, Holte et al. [61] showed that it is possible to get around Valtorta's theorem if the abstraction is a homomorphism, which is the most common type today. Homomorphisms and embeddings are particular types of structure-preserving maps and their formal definitions can be found in Section 3.2. In analogy to the DRP, Zilles and Holte [92] defined the *downward path preserving (DPP)* criterion, which guarantees that there are no *spurious states*, i.e. states that are reachable in the abstract state space but that have no corresponding reachable states in the ground state space. This is important for heuristics since spurious states can result in overly optimistic estimates [45].

While abstraction-based heuristics is still an active research area (cf. the papers by Geißer et al. [37], Pommerening et al. [77], and Steinmetz and Torralba [87]), it may seem as if refinement has come out of fashion in planning. There are, however, some recent examples of reviving this method. For instance, Seipp and Helmert [83] consider plan refinement using cartesian abstraction functions and Saribatur et al. [81] suggest encoding planning instances as instances of Answer set programming and then use abstraction refinement on the latter. Refinement has also continued to be used in other related areas such as path planning [88] and model checking [19]. Since action planning and path planning must often be performed together in robotics, it may be interesting to consider refinement methods also for action planning in this context. There are also many relationships between action planning and model checking [30] and refinement methods have been very successful in the latter case. Hoffmann et al. [54] suggested that refinement is appropriate for proving that there is no solution, while heuristic search is better for finding solutions. This could explain why model checking focuses on refinement, while planning focuses on heuristic search. However, there is a recent trend in planning to consider also instances that may not have a solution [9,13,32,52], which can be expected to lead to a renewed interest in refinement. Furthermore, even though heuristic search is usually not considered good for proving unsolvability, Hoffmann et al. [52] demonstrate that the merge-and-shrink heuristic [49] may sometimes be successfully applied also to this problem, which begs for a better understanding of the relationships between refinement and heuristics. In addition, a number of new refinement methods for action planning have recently appeared in the literature [42,50,82]. A better understanding of the existing refinement methods is needed to put these newer methods into a historical context. While the literature on abstraction heuristics is fairly coherent, the literature on refinement is not very consistent; most methods are defined within a particular planning language or make very strong assumptions about the abstractions used. It is thus difficult to compare these different methods with each other, or with the abstraction heuristics approach. The work we present in this article aims at providing a formal framework that enables making such comparisons in a more straightforward and transparent way, as described next.

1.2. Our contribution

The purpose of this article is to present and demonstrate a novel theoretical framework for modelling and analysing abstraction and abstraction-like methods in planning and search in a more systematic and formal way than previously possible. This is an important research topic still today, despite the long history of using abstraction:

> Although abstraction and hierarchical planning seem central to controlling complexity in real life, several authors argue that it has not been used as extensively as it could have been. The reason might lie in the fact that there is still much work to be done in order to better understand all the different ways of doing optimal abstraction planning.
>
> [Saitta and Zucker [80, p. 56]]

We want to stress that we do not attempt to create yet another grand theory of abstraction (like the approaches by Giunchiglia and Walsh [39] or Nayak and Levy [73]). Our goal is more pragmatic and somewhat similar in methodology to the theory of reformulation for reasoning about physical systems by Choueiry et al. [18]. We define a framework that is powerful enough to model many, but not all, state-abstraction methods that appear in the literature, with the primary purpose of analysing and comparing them in a transparent way. That is, we aim at a balance between modelling generality and sufficient concreteness for proving results.

The following are the main contributions of this article.

1. In Sec. 2, we define a theoretical framework based on *labelled state-transition graphs (STG)* and a concept of *transformation* between such graphs. Then a *transformation instance* consists of
   (a) two such graphs $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_2, E_2 \rangle$, where the vertex set S_i is the state space and the set E_i of labelled arcs is the set of possible transitions, and
   (b) a transformation $\langle f, R \rangle$ from $\mathbb{G}_1$ to $\mathbb{G}_2$, where f is a function and R is a relation.
   The function f maps states in S_1 to subsets of S_2 such that it partitions S_2. This is in contrast to the usual approach where f maps states to states (such as the approaches by Holte et al. [59] or Helmert et al. [48]). The relation R relates the labels of the two graphs, thus providing a way to couple subsets of transitions in the two graphs to each other. In the case of planning, the labels are typically actions so R specifies the connection between ground and abstract actions.
2. We define a limited number of properties that such graph transformations can have. In Sec. 3, we first present a few very simple properties that are powerful in combination; for instance, they are sufficient to capture concepts such as embeddings, retractions and homomorphisms, but they can also have more powerful implications. The second group of properties are defined in Sec. 4 and they capture different refinement concepts, mainly corresponding to different amounts of backtracking between levels. That is, instead of prescribing a specific method for avoiding, or minimising, such backtracking, we define declarative conditions on transformations. Many of the results we prove concern relationships between such properties.
3. We demonstrate the power of this framework for analysing refinement by formally modelling six different abstraction and abstraction-like methods from the planning literature as transformations in a systematic way. This results in a transparency that allows for an easy and straightforward comparison of the methods, which would otherwise hardly be possible due to the very different formalisms and assumptions used for defining the methods. In particular, our choice of definition for the function f allows both for modelling certain abstraction methods that cannot be modelled in the usual way (e.g. the direct landmark-based surrogate method [27]) as well as modelling some abstractions in a way that is better than or complementary to the usual way (e.g. Abstrips abstraction [78]). For all six methods, we derive exactly which properties these methods have, thus enabling an entirely new type of formal comparison of abstraction methods, based on comparing their transformation properties. For instance, this reveals that variable projection and variable-domain abstraction are essentially equivalent from this perspective, while all other methods deviate from these two in different ways. This is done in three steps: In Sec. 5, we recall the SAS^+ planning language and show how it defines implicit STGs. In Sec. 6, we show that the six different abstraction methods mentioned above can be modelled in SAS^+ in a coherent way in our framework. Finally, in Sec. 7, we analyse the transformation properties of these abstractions and compare them based on which properties they have.
4. In Sec. 8, we extend our framework to handle also abstraction heuristics by additionally defining *admissibility* as a formal property (a heuristic is admissible if it never overestimates the true cost, which is important for most heuristic search algorithms since it allows for finding optimal/cheapest solutions). This enables an analysis of the relationships between abstraction refinement and abstraction heuristics in a formal way, not dependent on any particular abstraction method or formalism. The outcome is that admissibility implies a property that guarantees a strong form of refinement completeness. Otherwise the findings are mostly negative, but the proofs give valuable hints at why there are no more such strong relationships between heuristics and refinement in general, thus suggesting some future directions.
5. In order to take also *abstraction hierarchies* into account, we define *composition* of transformations in Sec. 9. We also define a *transitivity* concept, a property **X** is transitive if whenever two transformations both have property **X**, then also their composition has property **X**. Transitivity is important when forming abstraction hierarchies, since we want a desirable property to hold for the whole hierarchy, not just between some layers. We show that all important properties, and almost all of the abstraction methods in (3) above, are transitive. We also demonstrate the usefulness of transformation composition by modelling and analysing the merge-and-shrink heuristic [49], which requires composing mixed types of transformations.

The paper ends with a discussion of further usage and potential extensions in Sec. 10. Many concepts will be introduced during the course of the article and we have compiled a list of them in Appendix A (Table A.2) for easy access.

We finally want to point out that our purpose is not to invent new abstraction methods but to enable formal analyses and comparisons of various methods—both existing methods and methods yet to be invented. That said, we hope that an increased understanding of state abstraction will inspire to the invention of new and better methods, not only in planning and search.

Parts of the content of this article have appeared in preliminary form in two conference paper [6,7]. This version is substantially extended and it contains, in particular, full proofs of the main results. The content of Section 9 does not appear in any of the two conference papers.

2. STGs and STG transformations

In this section, we introduce our framework for studying abstractions. Before we begin, we first recall some notation and terminology. Let S be a finite set. Then $|S|$ denotes the cardinality of S and 2^S denotes the powerset of S. Let C be a

collection of subsets of S, i.e. $C \subseteq 2^S$. Then C *covers* S if $\bigcup_{X \in C} X = S$. A *partition* of a set S is a set P of non-empty subsets of S such that (1) P covers S and (2) for all $X, Y \in P$, if $X \neq Y$, then $X \cap Y = \varnothing$. The elements of P are called *parts*. Let $f : S \to T$ be a function, then the *range* of f is $\mathrm{Rng}(f) = \{f(x) \mid x \in S\}$.

Definition 1. A *state-transition graph (STG)* over a set L of *labels* is a tuple $\mathbb{G} = \langle S, E \rangle$ where the *state space* S is a finite set of vertices called *states* and $E \subseteq S \times S \times L$ is a finite set of labelled arcs.

We will usually not specify the set of labels explicitly, but let it be implicitly defined as $L(\mathbb{G}) = L(E) = \{\ell \mid \langle s, t, \ell \rangle \in E\}$. Note that the definition allows more than one arc between two states as long as the arcs have different labels or different direction. Also note that loops, i.e. arcs of the type $\langle s, s, \ell \rangle$, are allowed. The labels provide a means to identify a subset of the arcs by assigning the same label to these arcs. This is useful, for instance, in planning where a single action may induce many arcs in an STG. If all arcs have the same label, then the STG collapses to an ordinary directed graph, and we will consider an unlabelled directed graph as equivalent to an STG with a single (often unspecified) label on all arcs. Our focus will be on general properties of STGs, but it is also common in the literature to study specific cases $\langle S, E, I, G \rangle$, where $I \in S$ is a specific *initial state* and $G \subseteq S$ is a set of *goal states*, sometimes known as a *transition system*. Our approach is more general since we are primarily interested in properties that hold for all choices of initial and goal states, and we discuss the connections to transition systems in Section 10. It should also be noted that although we have developed our framework for a particular purpose, almost all of the theory is general and can be applied anywhere it fits. For instance, referring to the vertices as states is just a naming convention and all properties we define work also for undirected graphs.

Definition 2. Let S_1 and S_2 be state spaces. A *transformation function* from S_1 to S_2 is a total function f from $S_1 \to S_2$ such that $\mathrm{Rng}(f)$ is a partition of S_2. Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ be two STGs. A *transformation function* from $\mathbb{G}_1$ to $\mathbb{G}_2$ is a transformation function from S_1 to S_2. A *label relation* from $\mathbb{G}_1$ to $\mathbb{G}_2$ is a binary relation $R \subseteq L(\mathbb{G}_1) \times L(\mathbb{G}_2)$. A *transformation* from $\mathbb{G}_1$ to $\mathbb{G}_2$ is a pair $\tau = \langle f, R \rangle$ where f is a transformation function from $\mathbb{G}_1$ to $\mathbb{G}_2$ and R is a label relation from $\mathbb{G}_1$ to $\mathbb{G}_2$.

The transformation function f specifies how the transformation maps states from S_1 to S_2 while the label relation R provides additional information about how sets of arcs are related between the two STGs. Note that f is a function from S_1 to 2^{S_2}, that is, it maps a state in S_1 to a set of states in S_2. Furthermore, the range of f is a partition of S_2 so all sets in the range are non-empty. We use a function rather than a relation since it makes the theory clearer and simpler. It is also more in line with previous work in the area, where ordinary functions $f : S_1 \to S_2$ are commonly used for abstractions. Also note that $\mathrm{Rng}(f) \subseteq 2^{S_2}$, i.e. it is a set of subsets of S_2, and that $f(s) \neq \varnothing$ for all $s \in S_1$ since $\mathrm{Rng}(f)$ is a partition. Transformation functions are extended to sequences of states such that $f(s_1, \ldots, s_k) = f(s_1), \ldots, f(s_k)$.

Example 3. Consider two STGs $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_2, E_2 \rangle$, where $S_1 = \{00, 01, 10, 11\}$, $E_1 = \{\langle 00, 01, a \rangle, \langle 01, 10, b \rangle, \langle 10, 11, a \rangle\}$, $S_2 = \{0, 1\}$ and $E_2 = \{\langle 0, 1, c \rangle\}$. Also define a function $f_1 : S_1 \to 2^{S_2}$ such that $f_1(xy) = \{x\}$. This is illustrated in Fig. 1 (top). We see immediately that f_1 is a transformation function from $\mathbb{G}_1$ to $\mathbb{G}_2$. Then define $f_2 : S_1 \to 2^{S_2}$ such that $f_2(xy) = \{x, y\}$; this function is not a transformation function since $f_2(00) = \{0\}$ and $f_2(01) = \{0, 1\}$ which implies that $\mathrm{Rng}(f_2)$ is not a partition of S_2. Further define an STG $\mathbb{G}_3 = \langle S_3, E_3 \rangle$, where $S_3 = \{0, \ldots, 7\}$ and $E_3 = \{\langle x, y, d \rangle \mid x \neq y\}$. Finally, the function $f_3(xy) = \{2x + y, 7 - 2x - y\}$ is a transformation function from $\mathbb{G}_1$ to $\mathbb{G}_3$ since $\mathrm{Rng}(f_3)$ partitions $\{0, \ldots, 7\}$ into $\{\{0, 7\}, \{1, 6\}, \{2, 5\}, \{3, 4\}\}$. The function f_3 is illustrated in Fig. 1 (bottom).

Our transformation concept is relational, not functional. A transformation τ is not a function that maps $\mathbb{G}_1$ to $\mathbb{G}_2$, but a relation from $\mathbb{G}_1$ to $\mathbb{G}_2$. Suppose $\tau = \langle f, R \rangle$ is a transformation from $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ to $\mathbb{G}_2 = \langle S_2, E_2 \rangle$. Then the state spaces S_1 and S_2 are fixed by f, but R does not specify the arc sets E_1 and E_2, it only specifies relationships between them. Note that this transformation concept is very general, for all choices of state spaces S_1, S_2, arc sets $E_1 \subseteq 2^{S_1}, E_2 \subseteq 2^{S_2}$ and label sets $L(E_1), L(E_2)$, the pair $\langle f, R \rangle$ is a transformation from $\langle S_1, E_1 \rangle$ to $\langle S_2, E_2 \rangle$ for all possible definitions of transformation function f and label relation R. Instead of restricting which combinations of f and R are transformations, we allow them all and will instead focus on studying which properties a transformation has depending on the choice of f and R. While it is more common to consider a graph transformation as a function from one graph to another, the relational view is a crucial choice of our theory, as will become evident later.

A high degree of symmetry is inherent in this transformation concept, and it is only a conceptual choice to say that one STG is the transformation from another and not the other way around. This symmetry simplifies our exposition considerably: concrete examples are the forthcoming definitions of properties together with some of the proofs.

Definition 4. Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ be two STGs, let f be a transformation function from S_1 to S_2 and let R be a label relation from $\mathbb{G}_1$ to $\mathbb{G}_2$. Then, the *reverse transformation function* $\overline{f} : S_2 \to 2^{S_1}$ is defined such that $\overline{f}(t) = \{s \in S_1 \mid t \in f(s)\}$ for all $t \in S_2$ and the *reverse label relation* $\overline{R} \subseteq L(\mathbb{G}_2) \times L(\mathbb{G}_1)$ is defined such that $\overline{R}(\ell_2, \ell_1)$ if and only if $R(\ell_1, \ell_2)$. Let $\tau = \langle f, R \rangle$ be a transformation, then the *reverse transformation* $\overline{\tau}$ is defined as $\overline{\tau} = \langle \overline{f}, \overline{R} \rangle$.

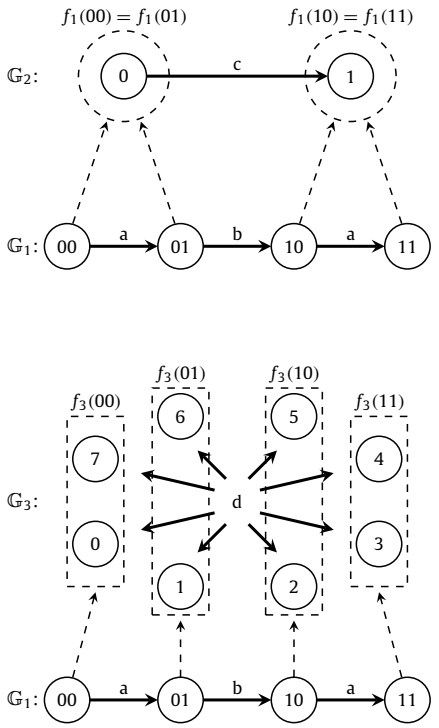


Fig. 1. The functions f_1 (top) and f_3 (bottom) in Example 3.

Example 5. Consider the functions f_1 and f_3 from Example 3 once again. We see that $\overline{f_1}(0) = \{00, 01\}$ and $\overline{f_1}(1) = \{10, 11\}$, while $\overline{f_3}(0) = \overline{f_3}(7) = \{00\}$, $\overline{f_3}(1) = \overline{f_3}(6) = \{01\}$, $\overline{f_3}(2) = \overline{f_3}(5) = \{10\}$ and $\overline{f_3}(3) = \overline{f_3}(4) = \{11\}$.

Theorem 6. *Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ be two STGs. Let f be a transformation function from $\mathbb{G}_1$ to $\mathbb{G}_2$, let R be a label relation from $\mathbb{G}_1$ to $\mathbb{G}_2$ and let $\tau = \langle f, R \rangle$ be a transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$. Then:*

1. *For all $s \in S_1$ and $t \in S_2$, it holds that $s \in \overline{f}(t)$ if and only if $t \in f(s)$.*
2. *$\overline{f}$ is a transformation function from $\mathbb{G}_2$ to $\mathbb{G}_1$.*
3. *$\overline{R}$ is a label relation from $\mathbb{G}_2$ to $\mathbb{G}_1$.*
4. *$\overline{\tau} = \langle \overline{f}, \overline{R} \rangle$ is a transformation from $\mathbb{G}_2$ to $\mathbb{G}_1$.*

Proof. (1) Immediate from the definitions.

(2) We first note that $\mathrm{Rng}(f)$ covers S_2, so it follows from (1) that $\overline{f}$ is a total function. It remains to prove that $\mathrm{Rng}(\overline{f})$ is a partition of S_1.

First suppose that $\overline{f}(t) = \varnothing$ for some $t \in S_2$. Then there is no $s \in S_1$ such that $s \in \overline{f}(t)$, i.e. $t \in f(s)$ by (1), but this is impossible since f covers S_2 by definition. It follows that $\overline{f}(t) \neq \varnothing$ for all $t \in S_2$.

Then suppose there is some $s \in S_1$ such that $s \notin \overline{f}(t)$ for all $t \in S_2$. This implies that there is no $t \in S_2$ such that $t \in f(s)$, which contradicts that f is total. It follows that $\mathrm{Rng}(\overline{f})$ covers S_1.

Finally, suppose there are two distinct elements in $\mathrm{Rng}(\overline{f})$ that are not disjoint. Then there are $t, t' \in S_2$ such that $\overline{f}(t) \neq \overline{f}(t')$ and $\overline{f}(t) \cap \overline{f}(t') \neq \varnothing$, so at least one of $\overline{f}(t) \setminus \overline{f}(t')$ and $\overline{f}(t') \setminus \overline{f}(t)$ must be non-empty. Without losing generality, assume the first of these hold. There are then $s, s' \in S_1$ such that $s \in \overline{f}(t) \cap \overline{f}(t')$ and $s' \in \overline{f}(t) \setminus \overline{f}(t')$, i.e. $s \in \overline{f}(t)$, $s \in \overline{f}(t')$, $s' \in \overline{f}(t)$ and $s' \notin \overline{f}(t')$. It follows that $t \in f(s)$, $t' \in f(s)$, $t \in f(s')$ and $t' \notin f(s')$. However, then $f(s) \neq f(s')$ and $f(s) \cap f(s') \neq \varnothing$, which contradicts that $\mathrm{Rng}(f)$ is a partition of S_1 and, thus, that f is a transformation function. We conclude that $\mathrm{Rng}(\overline{f})$ is a partition of S_1 and it follows that $\overline{f}$ is a transformation function.

(3) Immediate from definition.

(4) Immediate from (1–3) and the definitions. □

It is straightforward from this theorem that $\overline{\overline{f}} = f$.

We extend transformation functions (and other functions that map states to sets of states) in the following way. Let f be a function from S_1 to 2^{S_2}. Then for every subset $S \subseteq S_1$ we define $f(S) = \bigcup_{s \in S} f(s)$. For instance, if $f(0) = \{0, 1\}$ and $f(1) = \{2\}$ we get that $f(\{0, 1\}) = \{0, 1, 2\}$, not $f(\{0, 1\}) = \{\{0, 1\}, \{2\}\}$. This choice is crucial, especially for the generalisation of the

DPP criterion (in Section 4.3) and for composition of transformation functions (in Section 9). This definition respects that $S \subseteq \overline{f}(f(S))$, a property that f shares with an ordinary function and its inverse. While the range $\mathrm{Rng}(f)$ of a transformation function f is a partition of S_2 by definition, f also implicitly defines a partition P of S_1, such that for each part $X \in P$, it holds that $\overline{f}(f(X)) = X$. That is, f induces a bijection from $\mathrm{Rng}(\overline{f})$ to $\mathrm{Rng}(f)$.

As a convention, we will often not specify the STGs in definitions and theorems, tacitly assuming transformations to be from an STG $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ to an STG $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ unless otherwise specified. We also sometimes refer to $\mathbb{G}_1$ as the *ground* graph and $\mathbb{G}_2$ as the *abstract* graph.

Definition 7. Let $\mathbb{G}_1 = \langle S, E \rangle$ be an STG. A sequence $s_0, \ell_1, s_1, \ell_2, \ldots, \ell_n, s_n$, where $s_0, s_1, \ldots, s_n \in S$ and $\ell_1, \ldots, \ell_n \in L(E)$, is a *path* from s_0 to s_n of *length* n if either (1) $n = 0$ or (2) $\langle s_{i-1}, s_i, \ell_i \rangle \in E$ for all i $(1 \leq i \leq n)$. A sequence $s_0, s_1, \ldots, s_n$ of states in S is a *state path* from s_0 to s_n if there are labels $\ell_1, \ldots, \ell_n \in L(E)$ such that $s_0, \ell_1, s_1, \ell_2, \ldots, \ell_n, s_n$ is a path from s_0 to s_n. A sequence $\ell_1, \ldots, \ell_n$ of labels in $L(E)$ is a *label path* from a state $s \in S$ to a state $t \in S$ if there are states $s_0, \ldots, s_n \in S$ such that (1) $s_0 = s$, (2) $s_n = t$ and (3) $s_0, \ell_1, s_1, \ldots, \ell_n, s_n$ is a path from s_0 to s_n. A path is *simple* if no state occurs more than once.

For simplicity, we will usually refer to state paths as paths when it is clear from context which type of path it is. We will also usually refer to label paths as *plans*, since the labels will typically refer to actions (or operators). Also note that zero-length paths are allowed, that is, s is a path for any state s.

Definition 8. Let $\mathbb{G} = \langle S, E \rangle$ be an STG. Then for all $s \in S$, the set $\mathcal{R}(s)$ of *reachable states* from s is defined as

$$\mathcal{R}(s) = \{t \in S \mid \text{There is a path from } s \text{ to } t \text{ in } \mathbb{G}.\}$$

This is extended to subsets of S such that $\mathcal{R}(T) = \bigcup_{s \in T} \mathcal{R}(s)$ for all $T \subseteq S$.

Reachability is reflexive, i.e. $s \in \mathcal{R}(s)$ for all states s, and transitive, i.e. if $t \in \mathcal{R}(s)$ and $u \in \mathcal{R}(t)$, then $u \in \mathcal{R}(s)$, for all states s, t and u. We treat the STG as implicit from context, and when we consider two STGs $\mathbb{G}_1$ and $\mathbb{G}_2$ simultaneously we write $\mathcal{R}_1(\cdot)$ and $\mathcal{R}_2(\cdot)$ to clarify which graph the reachability function refers to.

3. Some basic properties

In this section, we will first define some simple transformation properties and then show that, despite their simplicity, they are powerful enough to capture concepts like embeddings, retractions and homomorphisms.

3.1. Definitions

One of the main purposes of this paper is to model abstractions and abstraction-like methods using classes of transformations with certain properties. In order to describe and analyse such transformations in a general way, we define the following properties.

Definition 9. Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ be two STGs and let $\tau = \langle f, R \rangle$ be a transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$. Then τ can have the following properties:

$\mathbf{M}_\uparrow$: $|f(s)| = 1$ for all $s \in S_1$.
$\mathbf{M}_\downarrow$: $|\overline{f}(s)| = 1$ for all $s \in S_2$.
$\mathbf{R}_\uparrow$: For all $\langle s_1, t_1, \ell_1 \rangle \in E_1$, there is some $\langle s_2, t_2, \ell_2 \rangle \in E_2$ such that $R(\ell_1, \ell_2)$.
$\mathbf{R}_\downarrow$: For all $\langle s_2, t_2, \ell_2 \rangle \in E_2$, there is some $\langle s_1, t_1, \ell_1 \rangle \in E_1$ such that $R(\ell_1, \ell_2)$.
$\mathbf{C}_\uparrow$: For all $\langle s_1, t_1, \ell_1 \rangle \in E_1$, if there is some $\ell_2 \in L(E_2)$ such that $R(\ell_1, \ell_2)$, then there is some $\langle s_2, t_2, \ell_2 \rangle \in E_2$ such that $s_2 \in f(s_1)$ and $t_2 \in f(t_1)$.
$\mathbf{C}_\downarrow$: For all $\langle s_2, t_2, \ell_2 \rangle \in E_2$, if there is some $\ell_1 \in L(E_1)$ such that $R(\ell_1, \ell_2)$, then there is some $\langle s_1, t_1, \ell_1 \rangle \in E_1$ such that $s_1 \in \overline{f}(s_2)$ and $t_1 \in \overline{f}(t_2)$.

Properties $\mathbf{M}_\uparrow/\mathbf{M}_\downarrow$ (*upwards/downwards many-one*) depend only on f and may thus hold also for f itself. The intention of $\mathbf{M}_\uparrow$ is to say that f maps every state in $\mathbb{G}_1$ to a single state in $\mathbb{G}_2$. While this may seem natural we will see examples later on where this property does not hold. We often write $f(s) = t$ instead of $t \in f(s)$ when f is $\mathbf{M}_\uparrow$ and analogously for $\overline{f}$. Properties $\mathbf{R}_\uparrow/\mathbf{R}_\downarrow$ (*upwards/downwards related*) depend only on R and may thus hold also for R itself. The intention behind $\mathbf{R}_\uparrow$ is that if there is a non-empty set of arcs in $\mathbb{G}_1$ with a specific label, then there is at least one arc in $\mathbb{G}_2$ that is explicitly specified via R to correspond to this arc set. Properties $\mathbf{C}_\uparrow/\mathbf{C}_\downarrow$ (*upwards/downwards coupled*) describe the connection between f and R. The intention behind $\mathbf{C}_\uparrow$ is to provide a way to tie up f and R to each other and require that arcs that are related via R must go between states that are related via f. We use a double-headed arrow when a condition

holds both upward and downward. For instance, $\mathbf{C}_\updownarrow$ (*up-down coupled*) means that both $\mathbf{C}_\uparrow$ and $\mathbf{C}_\downarrow$ hold. These classifications retain the symmetric nature of transformations. For instance, $\langle f, R\rangle$ is a $\mathbf{C}_\downarrow$ transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$ if and only if $\langle \overline{f}, \overline{R}\rangle$ is a $\mathbf{C}_\uparrow$ transformation from $\mathbb{G}_2$ to $\mathbb{G}_1$. It should be noted that these properties are only examples that we have chosen to use in this paper since they are simple, yet powerful. It is naturally possible to define other similar properties within our framework.

Example 10. We reconsider Example 3. The transformation function f_1 is $\mathbf{M}_\uparrow$ but not $\mathbf{M}_\downarrow$ while the transformation function f_3 is $\mathbf{M}_\downarrow$ but not $\mathbf{M}_\uparrow$. Define a label relation $R_1 = \{a, b\} \times \{c\}$ and the transformation $\tau_1 = \langle f_1, R_1\rangle$ from $\mathbb{G}_1$ to $\mathbb{G}_2$. Then τ_1 has both property $\mathbf{R}_\uparrow$ and $\mathbf{R}_\downarrow$. It also has property $\mathbf{C}_\downarrow$, but not $\mathbf{C}_\uparrow$ (consider the edge from 00 to 01). Instead define the label relation $R_2 = \{\langle b, c\rangle\}$ and the transformation $\tau_2 = \langle f_1, R_2\rangle$. Then τ_2 is not $\mathbf{R}_\uparrow$, but it is $\mathbf{C}_\uparrow$.

Proposition 11. *If f is an $\mathbf{M}_\updownarrow$ transformation function from S_1 to S_2, then f is a bijection from S_1 to S_2.*

We write $\mathbf{X} \Rightarrow \mathbf{Y}$ to denote that every transformation that has property $\mathbf{X}$ must also have property $\mathbf{Y}$, and we write $\mathbf{X} \not\Rightarrow \mathbf{Y}$ when this is not the case.

3.2. Morphisms

The majority of abstraction functions considered in search and planning can be divided into three groups: embeddings, retractions and homomorphisms [59]. An embedding is typically a mapping from a graph $\langle S, E_1\rangle$ to an abstract graph $\langle S, E_2\rangle$, where $E_1 \subseteq E_2$, i.e. the state spaces are the same but there are additional arcs. Examples of embedding abstractions appear both explicitly [36], as well as implicitly in methods such as precondition relaxation [78] and adding macro operators [67,15]. In graph terms the latter means adding some of the possible transitive arcs to the graph. A retraction is the opposite, a mapping from $\langle S, E_1\rangle$ to $\langle S, E_2\rangle$ such that $E_2 \subseteq E_1$. Examples of retraction abstractions appear both explicitly [36] and implicitly, as in removing redundant actions [46]. Valtorta [90] showed that heuristic search with the A* algorithm [44] will not outperform blind search directly in the ground state space if using a heuristic based on embedding abstractions (under certain assumptions on how to compute the heuristic). Holte et al. [61] showed that this result does not transfer to homomorphic abstractions, where the total number of explored nodes can be fewer with abstraction. Since homomorphisms are also well studied theoretically and have many beneficial properties, they have since been the most common type of abstraction function. They are often used in the form of strong (or faithful) homomorphisms. Examples of homomorphic abstractions appear, for instance, in the publications by Holte et al. [59], Haslum et al. [45], Helmert et al. [48] and Zilles and Holte [92].

We will now formally define embeddings, retractions and homomorphisms within our framework. The standard definitions of these concepts consider only unlabelled graphs, but since we want the definitions to apply to transformations, we also take labels into account, although these are irrelevant for the actual concepts defined. Furthermore, since these concepts have no well-defined meaning for set-valued functions, we restrict the definitions to $\mathbf{M}_\uparrow$ transformation functions. Recall the convention that we write $f(s) = t$ as a shorthand for $t \in f(s)$ when f is $\mathbf{M}_\uparrow$.

Definition 12. An $\mathbf{M}_\uparrow$ transformation function f from an STG $\mathbb{G}_1 = \langle S_1, E_1\rangle$ to an STG $\mathbb{G}_2 = \langle S_2, E_2\rangle$ is

1. a *homomorphism* if for all $\langle s_1, t_1, \ell_1\rangle \in E_1$ there is some $\ell_2 \in L(E_2)$ such that $\langle f(s_1), f(t_1), \ell_2\rangle \in E_2$;
2. a *strong homomorphism* if it is a homomorphism and for all $\langle s_2, t_2, \ell_2\rangle \in E_2$, there is some $\langle s_1, t_1, \ell_1\rangle \in E_1$ such that $f(s_1) = s_2$ and $f(t_1) = t_2$;
3. an *embedding* if it is $\mathbf{M}_\downarrow$ and a homomorphism;
4. a *retraction* if it is an embedding from $\mathbb{G}_2$ to $\mathbb{G}_1$.

Note that if f is $\mathbf{M}_\updownarrow$, then it is a bijection between $\mathrm{Rng}(\overline{f})$ and $\mathrm{Rng}(f)$, which is slightly more general than the usual assumption that f is the identity function in cases (3) and (4).

We will now show that the transformation properties that we have just introduced are sufficient to capture and distinguish between these different abstraction concepts, which strongly indicates that these properties are not arbitrary but express something essential about transformations.

Theorem 13. *Let $\mathbb{G}_1 = \langle S_1, E_1\rangle$ and $\mathbb{G}_2 = \langle S_2, E_2\rangle$ be two STGs and let $\tau = \langle f, R\rangle$ be a transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$. Then:*

1. *If τ is $\mathbf{M}_\uparrow\mathbf{R}_\uparrow\mathbf{C}_\uparrow$, then f is a homomorphism.*
2. *If τ is $\mathbf{M}_\uparrow\mathbf{R}_\updownarrow\mathbf{C}_\updownarrow$, then f is a strong homomorphism.*
3. *If τ is $\mathbf{M}_\updownarrow\mathbf{R}_\uparrow\mathbf{C}_\uparrow$, then f is an embedding.*
4. *If τ is $\mathbf{M}_\updownarrow\mathbf{R}_\downarrow\mathbf{C}_\downarrow$, then f is a retraction.*

Proof. (1) Assume τ is $\mathbf{M}_\uparrow\mathbf{R}_\uparrow\mathbf{C}_\uparrow$. Let $e_1 = \langle s_1, t_1, \ell_1\rangle$ be an arbitrary arc in E_1. Since τ is $\mathbf{R}_\uparrow$ there is some arc $\langle s_2, t_2, \ell_2\rangle \in E_2$ such that $R(\ell_1, \ell_2)$. Since τ is also $\mathbf{C}_\uparrow$ there must be such an arc where $s_2 \in f(s_1)$ and $t_2 \in f(t_1)$, i.e. $s_2 = f(s_1)$ and $t_2 = f(t_1)$ since f is $\mathbf{M}_\uparrow$. It follows that f is a homomorphism, since e_1 was chosen arbitrarily.

(2) Assume that τ is $\mathbf{M}_\uparrow\mathbf{R}_\updownarrow\mathbf{C}_\updownarrow$. It follows from (1) that f is a homomorphism. Let $e_2 = \langle s_2, t_2, \ell_2\rangle$ be an arbitrary arc in E_2. Since τ is $\mathbf{R}_\downarrow$ there is some arc $\langle s_1, t_1, \ell_1\rangle \in E_1$ such that $R(\ell_1, \ell_2)$. Since τ is also $\mathbf{C}_\downarrow$ there must be such an arc where $s_1 \in \overline{f}(s_2)$ and $t_1 \in \overline{f}(t_2)$, i.e. $s_2 = f(s_1)$ and $t_2 = f(t_1)$ since f is $\mathbf{M}_\uparrow$. It follows that f is a strong homomorphism, since e_2 was chosen arbitrarily.

(3) Immediate from (1) and Definition 12.

(4) Immediate from (3) by symmetry. □

For the opposite direction we make the additional assumption of unit-labelled STGs, since the original concepts ignore labels.

Theorem 14. *Let $\mathbb{G}_1 = \langle S_1, E_1\rangle$ and $\mathbb{G}_2 = \langle S_2, E_2\rangle$ be two STGs such that $L(E_1) = L(E_2) = \{\ell\}$. Let $\tau = \langle f, R\rangle$ be a transformation such that $R(\ell, \ell)$ holds. Then:*

1. *If f is a homomorphism, then τ is $\boldsymbol{M}_\uparrow\boldsymbol{R}_\uparrow\boldsymbol{C}_\uparrow$.*
2. *If f is a strong homomorphism, then τ is $\boldsymbol{M}_\uparrow\boldsymbol{R}_\updownarrow\boldsymbol{C}_\updownarrow$.*
3. *If f is an embedding, then τ is $\boldsymbol{M}_\updownarrow\boldsymbol{R}_\uparrow\boldsymbol{C}_\uparrow$.*
4. *If f is a retraction, then τ is $\boldsymbol{M}_\updownarrow\boldsymbol{R}_\downarrow\boldsymbol{C}_\downarrow$.*

Proof. (1) Suppose f is a homomorphism. Then it is implicitly $\mathbf{M}_\uparrow$. Let $e_1 = \langle s_1, t_1, \ell_1\rangle$ be an arbitrary arc in E_1. Since f is a homomorphism there is some label $\ell_2 \in L(E_2)$ such that $\langle f(s_1), f(t_1), \ell_2\rangle \in E_2$. Then $\ell_2 = \ell_1 = \ell$ is the only possibility, so $R(\ell_1, \ell_2)$ holds by definition of R. Since e_1, and thus ℓ_1, was chosen arbitrarily, it follows that τ is $\mathbf{R}_\uparrow$. Now suppose $e_1 = \langle s_1, t_1, \ell_1\rangle \in E_1$ and $R(\ell_1, \ell_2)$ holds for some $\ell_2 \in L(E_2)$. Since f is a homomorphism there is also some $\ell_3 \in L(E_2)$ such that $\langle f(s_1), f(t_2), \ell_3\rangle \in E_2$, but $\ell_3 = \ell_2 = \ell_1 = \ell$ is the only possibility, so it follows that τ is $\mathbf{C}_\uparrow$.

(2) Suppose f is a strong homomorphism. Then f is a homomorphism, so it follows from (1) that τ is $\mathbf{M}_\uparrow\mathbf{R}_\uparrow\mathbf{C}_\uparrow$. Let $e_2 = \langle s_2, t_2, \ell_2\rangle$ be an arbitrary arc in E_2. Since f is a strong homomorphism there is some arc $\langle s_1, t_1, \ell_1\rangle \in E_1$ such that $f(s_1) = s_2$ and $f(t_1) = t_2$. Then $R(\ell_1, \ell_2)$ holds since $\ell_2 = \ell_1 = \ell$ is the only possibility. Since e_2 was chosen arbitrarily, it follows that τ is $\mathbf{R}_\downarrow$. Suppose $\langle s_2, t_2, \ell_2\rangle \in E_2$ and $R(\ell_1, \ell_2)$ holds for some $\ell_1 \in L(E_1)$. Since f is a strong homomorphism there is some arc $\langle s_1, t_1, \ell_3\rangle \in E_1$ such that $f(s_1) = s_2$ and $f(t_1) = t_2$. The only possibility is that $\ell_3 = \ell_1 = \ell_2 = \ell$ so it follows that τ is $\mathbf{C}_\downarrow$.

(3) Suppose f is an embedding. Then τ is a homomorphism, so it follows from (1) that τ is $\mathbf{M}_\uparrow\mathbf{R}_\uparrow\mathbf{C}_\uparrow$. The result follows since τ is also $\mathbf{M}_\downarrow$ by definition.

(4) Immediate from (3) by symmetry. □

4. Refinement properties

In this section, we first discuss and define abstraction refinement within our framework, then we discuss these definitions in the context of the backtracking-between-levels problem. We continue with defining transformation properties that correspond to different strengths of refinement, which we refer to as refinement properties, and then analyse how these properties relate to each other.

4.1. Abstraction refinement

Abstraction refinement refers to the following general technique. Suppose we want to find a path from s to t in a ground graph $\mathbb{G}_1$. Then we map s and t to their corresponding abstractions $f(s)$ and $f(t)$ in an abstract graph $\mathbb{G}_2$, and try to find a path $\sigma = t_0, \ell_1, t_1, \ldots, \ell_m, t_m$ in $\mathbb{G}_2$, where $t_0 \in f(s)$ and $t_m \in f(t)$. There are two major techniques for refining σ: *label refinement* and *state refinement*. Label refinement proceeds by mapping the label sequence of σ back to a sequence $\lambda = \ell'_1, \ldots, \ell'_m$ such that $R(\ell'_i, \ell_i)$ holds for all i. This new label sequence is usually not a valid plan in $\mathbb{G}_1$, so the refinement process has to fill in with additional labels until the result is a plan from s to t that contains λ as a subplan. Examples of label refinement appear in the articles by Knoblock [65] and Bacchus and Yang [3]. State refinement instead ignores the labels entirely and maps the abstract states $t_0, \ldots, t_m$ back to a sequence $s_0, \ldots, s_m$ of corresponding states in S_1. The refinement process then tries to find subpaths from s_{i-1} to s_i for all i. Examples of state refinement are presented by Knoblock [63] and Holte et al. [59]. These two methods were described and compared in principle on a representation-independent graph level by Holte et al. [59].

A refinement abstraction must satisfy two criteria in order to always produce correct solutions:

1. It must be *complete*, i.e. if there is a ground path from s to t in $\mathbb{G}_1$, then there must also be an abstract path in $\mathbb{G}_2$ from $f(s)$ to $f(t)$.

2. It must be *sound*, i.e. if there is an abstract path from $f(s)$ to $f(t)$ in $\mathbb{G}_2$, then there must also be a ground path from s to t in $\mathbb{G}_1$.

One may also consider different degrees of these concepts. For instance, soundness could be strengthened to require that every abstract solution can be refined into a ground solution. We will consider such degrees of refinement later in this section.

One or both of these criteria are often sacrificed in practice. If the abstraction is not complete, then we may fail to find a ground solution even when one exists; since there is no abstract solution, we cannot find the ground solution by refinement. Many abstraction methods are complete, but completeness is no guarantee that an abstraction is useful for refinement; for instance, the empty plan is often a valid abstract plan that can be refined into a ground plan, but it gives no clue at all how to do this refinement.

If the abstraction is not sound, then we may find an abstract solution even when there is no ground solution. While this is not disastrous, it may take longer time to find that there is no solution than if searching directly in the ground graph, in particular if there are many abstract solutions and many ways to refine an abstract solution. This is known as the problem of *backtracking between levels*. Knoblock et al. [66] suggested a condition on abstractions called the *ordered monotonicity criterion*, which guarantees that if there is an abstract solution, then there is also a ground solution. It was noted by Smith and Peot [86] that this is still a very weak criterion; there may be a large number of abstract solutions of which only one or a few are refinable, which will still cause extensive backtracking to the abstract level to look for different abstract solutions to try refining. This is thus another case where backtracking between levels occurs. This problem was further analysed and quantified by Bacchus and Yang [3], who suggested the stronger *downward refinement property (DRP)*. The DRP guarantees that every abstract plan can be refined into a ground plan, but there are obviously fewer abstractions satisfying the DRP than the ordered monotonicity criterion. Both these criteria also suffer from the problem that they give no quantitative guarantees on the solutions. In the best case, hierarchical abstraction can give an exponential speed-up over searching in the ground graph [3,64]. However, hierarchical abstraction can be counter-productive in the worst case. Bäckström and Jonsson [5] demonstrated two different abstraction hierarchies that both satisfy the DRP, but where the abstract solutions in one of them can be refined into optimal ground solutions while the abstract solutions of the other one can only be refined into ground solutions that are exponentially longer than the optimal ground solutions.

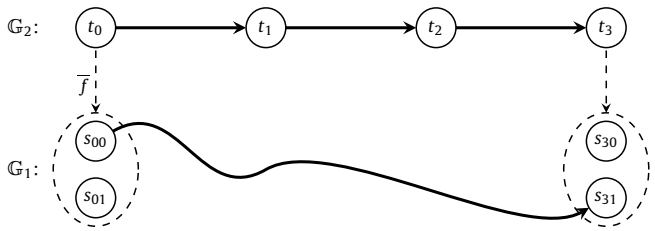
A slightly different approach was taken by Holte et al. [59], who defined the concept *classical refinement*. A classical refinement of an abstract path $t_1, \dots, t_m$ is a state sequence $\sigma = s_1, \dots, s_n$ where there are indices $i_0, \dots, i_m$ such that $i_0 < \cdots < i_m$, $i_0 = 0$, $i_m = n$ and $f(s_{i_{j-1}+1}) = \cdots = f(s_{i_j}) = t_j$ for all j $(1 \leq j \leq m)$. That is, all states in the abstract path must be mapped to states in the ground path, in the same order, all ground states that map to the same abstract state must occur consecutively and all ground states must map to abstract states that appear in the abstract solution.

The concept of *spurious states* [45,92] is more general and declarative, and it uses state refinement. A spurious state is a state that is reachable in the abstract graph, but that does not correspond to any reachable state in the ground graph. For a state s in the ground graph, the corresponding set of spurious states is $\mathcal{S}(s) = \mathcal{R}_2(f(s)) \setminus f(\mathcal{R}_1(s))$. Spurious states are undesirable in an abstract solution since this solution cannot then be refined into a ground solution. A similar notion is used in model-checking, where a spurious state is a counter-example that can be reached in the abstract model, but not in the ground model [19,40]. A necessary and sufficient condition for avoiding spurious states is that an abstraction satisfies the *downward path preserving (DPP)* property [92]. The DPP property guarantees that $\mathcal{S}(s) = \varnothing$ for all s. It was defined for strong homomorphic abstractions using the following two conditions:

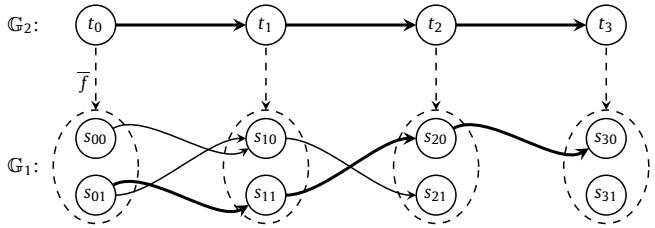
1. $\mathcal{R}_2(f(s)) \subseteq f(\mathcal{R}_1(s))$ for all $s \in S_1$,
2. $f(\mathcal{R}_1(s)) \subseteq \mathcal{R}_2(f(s))$ for all $s \in S_1$.

The DPP property holds if both these conditions are satisfied. We will later use these two conditions under the abbreviations $\mathbf{P}_{\downarrow}$ and $\mathbf{P}_{\uparrow}$, respectively. We note that (1) is a soundness criterion that holds if there are no spurious states, i.e. it guarantees that if we can reach an abstract state in $\mathbb{G}_2$, then we can also reach some corresponding ground state in $\mathbb{G}_1$. Condition (2) is a completeness criterion. Zilles and Holte noted that (2) is inherent in every strong homomorphic abstraction, but not necessarily true in other abstractions.

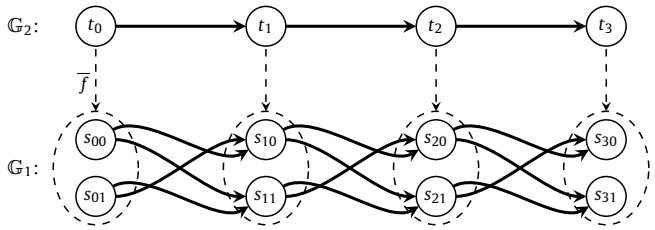
As can be seen from this brief survey of results, the work on abstraction refinement in the literature has been quite disparate, lacking common concepts and notation. Both the ordered monotonicity criterion and the DRP were defined within particular planning languages, thus being difficult to transfer to and compare with other approaches. Classical refinement had a clearer and more general definition, but made some strong assumptions about the abstractions used. The theory on spurious states is the clearest and most general of the approaches, but it has a very weak connection to the other concepts. In order to improve on this situation, we will define a general framework of abstraction refinement, that captures a number of different refinement concepts, as well as spurious states. We will also define a number of declarative properties on transformations that capture these different concepts. Finally, we prove a number of relationships between these properties that allows for drawing conclusions about an abstraction based on what properties it has. In contrast to the other approaches described above, we do not prescribe any particular methods, but focus on capturing various refinement concepts declaratively.



(a) Loosely downwards state refinable



(b) Weakly downwards state refinable



(c) Strongly downwards state refinable

Fig. 2. Downwards state refinements (curly arrows denote paths).

4.2. Formalising refinement

We focus on state refinement in this article; it is simpler and clearer than label refinement, and there are also arguments why it can be expected to be more efficient [59]. We define three types of state refinement, with varying degree of providing guidance for the refinement process.

Definition 15. Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ be two STGs and let $\tau = \langle f, R \rangle$ be a transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$. Let $\sigma = t_0, t_1, \ldots, t_n$ be a state path in $\mathbb{G}_2$. Then:

1. σ is *loosely downwards state refinable* if there are two states $s \in \overline{f}(t_0)$ and $s' \in \overline{f}(t_n)$ such that $s' \in \mathcal{R}_1(s)$.
2. σ is *weakly downwards state refinable* if there is a sequence $s_0, s_1, \ldots, s_n$ of states in S_1 such that $s_i \in \overline{f}(t_i)$ for all i $(0 \leq i \leq n)$ and $s_i \in \mathcal{R}_1(s_{i-1})$ for all i $(1 \leq i \leq n)$.
3. σ is *strongly downwards state refinable* if for every i $(1 \leq i \leq n)$, it holds that $s' \in \mathcal{R}_1(s)$ for all choices of $s \in \overline{f}(t_{i-1})$ and $s' \in \overline{f}(t_i)$.

The corresponding cases of upwards refinability are defined symmetrically.

These concepts are illustrated in Fig. 2. In all cases, a path $\sigma = t_0, t_1, t_2, t_3$ in the abstract graph $\mathbb{G}_2$ is refined into a corresponding path in the ground graph $\mathbb{G}_1$. Curly arrows denote paths, i.e. they may consist of several arcs and pass through states not shown in the figure.

Loose refinement only requires that there is a path corresponding to σ in the ground graph, i.e. there must be a path from some state in $\overline{f}(t_0) = \{s_{00}, s_{01}\}$ to some state in $\overline{f}(t_3) = \{s_{30}, s_{31}\}$. This is satisfied by the path from s_{00} to s_{31}, which is thus a loose refinement of σ. The intermediate states t_1 and t_2 are ignored in the refinement. Weak refinement additionally

```
1  function LPath(s,t)
2    choose a state path σ = t_0, t_1, ..., t_k such that t_0 ∈ f(s) and t_k ∈ f(t)
3    if there is no such σ then fail
4    else return Refine(t_0,t_k)

1  function WSPath(s,t)
2    choose a state path σ = t_0, t_1, ..., t_k such that t_0 ∈ f(s) and t_k ∈ f(t)
3    if there is no such σ then fail
4    else return Refine(t_0, t_1, ..., t_k)

1  function Refine(t_0, t_1, ..., t_k)
2    choose s_0 ∈ f̄(t_0)
3    if k = 0 then return s_0
4    else
5      σ_2 = Refine(t_1, ..., t_k)
6      s_1 = first(σ_2)
7      choose a state path σ_1 from s_0 to s_1
8      if there is no such σ_1 then fail
9      else return the concatenation σ_1; σ_2
```

Fig. 3. Algorithms for path refinement. The **choose** statements are non-deterministic and serve as backtrack points.

requires that we use all states along σ and pass through some state in each of $\overline{f}(t_1)$ and $\overline{f}(t_2)$. This is satisfied by the three subpaths $s_{01}, \ldots, s_{11}$, $s_{11}, \ldots, s_{20}$ and $s_{20}, \ldots, s_{30}$, which together constitute a weak refinement of σ. We cannot, however, combine the subpaths $s_{00}, \ldots, s_{10}$, $s_{10}, \ldots, s_{21}$ and $s_{20}, \ldots, s_{30}$, since this will not even be a path. Weak refinement may be viewed as a generalisation of classical refinement. Finally, strong refinement requires that there is a path for any choice of states in $\overline{f}(t_0), \ldots, \overline{f}(t_3)$, which is satisfied in the last example in the figure. Obviously, loose and strong refinement are the two extreme points of a range of possible definitions, where weak refinement serves as an example of a concept in between. Further variants can be added when required.

The three refinement concepts in Definition 15 correspond to varying degrees of backtracking. This is illustrated by the two algorithms LPath and WSPath in Fig. 3. The **choose** statements are non-deterministic, that is, an actual implementation would use search with the **choose** statements as backtrack points. Algorithm LPath implements loose path refinement. It first tries to find an abstract path σ. If this succeeds, then it calls Refine to find a ground path between the states in the ground graph corresponding to the first and last states of σ. Assume that we somehow know that the path σ is loosely refinable, then we know that there is a corresponding ground solution, so there is no need for Refine to backtrack up to LPath again to look for another abstract solution. Apart from this, Refine is offered no further guidance how to find a ground path; it will have to do the same amount of search as if finding the path from scratch in the ground graph. Algorithm WSPath implements weak and strong path refinement. It first tries to find an abstract path σ. If this succeeds, then it passes the whole path σ to Refine so the states along this path can be used as subgoals. If σ is weakly refinable, then there is no need for Refine to backtrack up to WSPath again, but there may still be a choice of ground subgoals in $\overline{f}(t_i)$ for each state t_i along σ. The amount of backtracking in this search will depend on how these choices are made. Finally, if σ is strongly refinable, then there is not even any need to backtrack to the **choose** points within Refine. It does not matter how we choose the subgoals in each $\overline{f}(t_i)$, there will always be a path to the next one.

In loose and weak refinement, we only require that there is a ground path in $\mathbb{G}_1$ from some state in $\overline{f}(t_0)$, while we may in practice be interested in a path from a specific initial state s. We consider our definitions more general since they allow also for partially unspecified initial states, i.e. a set of initial states, which is sometimes considered in planning (under the name of *conformant planning*). There are several possibilities for handling a single initial state, including the following: Adding a dedicated initial state to the STGs, modifying f such that $\overline{f}(t) = \{s\}$ for all $t \in f(s)$, restricting the STG as described in Sec. 10 or modifying the refinement definitions themselves for actual specific purposes.

4.3. Refinement properties

We now turn our attention to transformation properties that are related to path refinement. We refer to these as *refinement properties*. All of these properties are related to various types of state refinement of paths. Hence, the actual labels on arcs are irrelevant.

Definition 16. Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ be two STGs and let $\tau = \langle f, R \rangle$ be a transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$. Then τ can have the following properties:

$\mathbf{P}_{\mathbf{L}\downarrow}$: Every path in $\mathbb{G}_2$ is loosely downwards state refinable.
$\mathbf{P}_{\mathbf{L}\uparrow}$: Every path in $\mathbb{G}_1$ is loosely upwards state refinable.
$\mathbf{P}_{k\downarrow}$: Every path in $\mathbb{G}_2$ of length k, or less, is weakly downwards state refinable.
$\mathbf{P}_{k\uparrow}$: Every path in $\mathbb{G}_1$ of length k, or less, is weakly upwards state refinable.

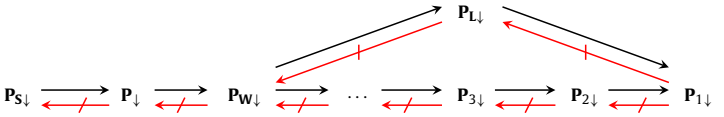


Fig. 4. Hierarchies of downwards refinement properties.

$\mathbf{P_{W\downarrow}}$: $\mathbf{P}_{k\downarrow}$ holds for all $k \geq 0$.
$\mathbf{P_{W\uparrow}}$: $\mathbf{P}_{k\uparrow}$ holds for all $k \geq 0$.
$\mathbf{P_{\downarrow}}$: $\mathcal{R}_2(f(s)) \subseteq f(\mathcal{R}_1(s))$ for all $s \in S_1$.
$\mathbf{P_{\uparrow}}$: $f(\mathcal{R}_1(s)) \subseteq \mathcal{R}_2(f(s))$ for all $s \in S_1$.
$\mathbf{P_{S\downarrow}}$: Every path in $\mathbb{G}_2$ is strongly downwards state refinable.
$\mathbf{P_{S\uparrow}}$: Every path in $\mathbb{G}_1$ is strongly upwards state refinable.

For the $\mathbf{P}_{k\uparrow}$ and $\mathbf{P}_{k\downarrow}$ properties, we first note that $\mathbf{P}_{0\uparrow}$ and $\mathbf{P}_{0\downarrow}$ always hold due to the definition of transformation functions, which matters for technical reasons in some proofs. Then consider the example in Fig. 2(b) and remove the path from s_{01} to s_{11}. The transformation in this example is clearly $\mathbf{P}_{1\downarrow}$. The only paths of length 2 in $\mathbb{G}_2$ are t_0, t_1, t_2 and t_1, t_2, t_3, which can both be weakly refined into paths in $\mathbb{G}_1$ (with two choices of refinements for the first one). The transformation is, thus, $\mathbf{P}_{2\downarrow}$. The path t_0, t_1, t_2, t_3 of length 3 does not have any weak refinement, though, since there is no path from some state in $\overline{f}(t_0)$ to some state in $\overline{f}(t_3)$ that also passes some state in $\overline{f}(t_1)$ and some state in $\overline{f}(t_2)$, in that order. Hence, the transformation is not $\mathbf{P}_{3\downarrow}$ and, thus, not $\mathbf{P_{W\downarrow}}$. Now add a path directly from s_{00} to s_{30}. The transformation is still not $\mathbf{P}_{3\downarrow}$, but the path t_0, t_1, t_2, t_3 can now be loosely refined into this new path in $\mathbb{G}_1$. All shorter paths are already loosely refinable since they are weakly refinable, so the modified transformation is $\mathbf{P_{L\downarrow}}$.

Properties $\mathbf{P_{\downarrow}}$ and $\mathbf{P_{\uparrow}}$ are identical to the conditions for DPP, except that we have generalised them from ordinary abstraction functions to transformations. This is the only property which is not exactly symmetric in its upwards and downwards versions, because in this case it is important which graph is the ground one and which is the abstract one; we cannot just interchange the graphs and redirect the property arrows.[3] Obviously, properties $\mathbf{P_{S\downarrow}}$ and $\mathbf{P_{S\uparrow}}$ are stronger than $\mathbf{P_{W\downarrow}}$ and $\mathbf{P_{W\uparrow}}$, and we will see that they are also stronger than $\mathbf{P_{\downarrow}}$ and $\mathbf{P_{\uparrow}}$. One may view property $\mathbf{P_{L\downarrow}}$ as a generalised variant of the ordered monotonicity property and property $\mathbf{P_{W\downarrow}}$ as a generalised variant of the DRP.

4.4. Relationships between refinement properties

Definition 16 contains many refinement properties so we will attempt to relate these properties to each other in this section. A partial summary of the results can be found in Fig. 4. In the later parts of the section, we will also consider some relations between refinements properties and the transformation properties from Definition 9. In particular, we show that the hierarchies in Fig. 4 collapse when combined with certain transformation properties. The following examples are used in the forthcoming proofs and are collected here for reference.

Example 17.

1. Let $S_1 = \{1, 2, 3, 4\}$, $S_2 = \{1, 2, 3\}$, $E_1 = \{\langle 1, 2, a\rangle, \langle 3, 4, a\rangle\}$ and $E_2 = \{\langle 1, 2, a\rangle, \langle 2, 3, a\rangle\}$. Let $f(1) = 1$, $f(2) = f(3) = 2$, $f(4) = 3$, $R(a, a)$ and $\tau = \langle f, R\rangle$.
2. Same as (1) but add the arc $\langle 1, 4, a\rangle$ to E_1.
3. Let $S_1 = \{1, 2, 3, 4\}$, $S_2 = \{1, 2\}$, $E_1 = \{\langle 1, 2, a\rangle\}$ and $E_2 = \{\langle 1, 2, a\rangle\}$. Let $f(1) = f(3) = 1$, $f(2) = f(4) = 2$, $R(a, a)$ and $\tau = \langle f, R\rangle$.
4. Same as (3) but add the arc $\langle 3, 2, a\rangle$ to E_1.
5. Let $S_1 = S_2 = \{1, 2, 3\}$, $E_1 = \{\langle 1, 2, a\rangle, \langle 2, 3, a\rangle\}$ and $E_2 = \{\langle 1, 3, a\rangle\}$. Let $f(1) = 1$, $f(2) = 2$, $f(3) = 3$, $R(a, a)$ and $\tau = \langle f, R\rangle$.
6. Let $S_1 = \{1, 2\}$, $S_2 = \{1, 2, 3, 4\}$, $E_1 = E_2 = \{\langle 1, 2, a\rangle\}$. Let $f(1) = \{1, 3\}$, $f(2) = \{2, 4\}$, $R(a, a)$ and $\tau = \langle f, R\rangle$.

The $\mathbf{P_S}$, $\mathbf{P}$, $\mathbf{P_W}$ and $\mathbf{P}_1$ properties form strict hierarchies in both directions, as the following theorem shows.

Theorem 18. *The following relationships hold:*
(1) $\boldsymbol{P_{S\downarrow}} \Rightarrow \boldsymbol{P_{\downarrow}} \Rightarrow \boldsymbol{P_{W\downarrow}} \Rightarrow \boldsymbol{P_{L\downarrow}} \Rightarrow \boldsymbol{P}_{1\downarrow}$,
(2) $\boldsymbol{P}_{1\downarrow} \not\Rightarrow \boldsymbol{P_{L\downarrow}} \not\Rightarrow \boldsymbol{P_{W\downarrow}} \not\Rightarrow \boldsymbol{P_{\downarrow}} \not\Rightarrow \boldsymbol{P_{S\downarrow}}$,
(3) $\boldsymbol{P_{S\uparrow}} \Rightarrow \boldsymbol{P_{\uparrow}} \Rightarrow \boldsymbol{P_{W\uparrow}} \Rightarrow \boldsymbol{P_{L\uparrow}} \Rightarrow \boldsymbol{P}_{1\uparrow}$,
(4) $\boldsymbol{P}_{1\uparrow} \not\Rightarrow \boldsymbol{P_{L\uparrow}} \not\Rightarrow \boldsymbol{P_{W\uparrow}} \not\Rightarrow \boldsymbol{P_{\uparrow}} \not\Rightarrow \boldsymbol{P_{S\uparrow}}$.

[3] While it is possible to give these two conditions different names and also define their symmetric counterparts, there seems little use for this since the two conditions are introduced only to make the DPP criterion comparable to our concepts.

Proof. (1) For each case assume that $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ are arbitrary STGs, and that $\tau = \langle f, R \rangle$ is an arbitrary transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$.

$\mathbf{P_{S\downarrow}} \Rightarrow \mathbf{P}_{\downarrow}$: Assume τ is $\mathbf{P_{S\downarrow}}$. Suppose $t \in \mathcal{R}_2(f(s))$ for some $s \in S_1$ and $t \in S_2$. Then there is some $t_0 \in f(s)$ such that $t \in \mathcal{R}_2(t_0)$. Hence, there is a path $\sigma = t_0, \ldots, t_n$ in $\mathbb{G}_2$ where $t_n = t$. Since τ is $\mathbf{P_{S\downarrow}}$ there must be a path in $\mathbb{G}_1$ from any s_0 in $\overline{f}(t_0)$ to any $s_n \in \overline{f}(t_n)$. It follows that $\overline{f}(t_n) \subseteq \mathcal{R}_1(s)$ and, hence, that $t_n \in f(\mathcal{R}_1(s))$. Hence, $\mathcal{R}_2(f(s)) \subseteq f(\mathcal{R}_1(s))$, so τ is $\mathbf{P}_{\downarrow}$.

$\mathbf{P}_{\downarrow} \Rightarrow \mathbf{P_{W\downarrow}}$: Assume τ is $\mathbf{P}_{\downarrow}$. We prove by induction over k that τ is $\mathbf{P}_{k\downarrow}$ for all $k \geq 0$, which proves that τ is $\mathbf{P_{W\downarrow}}$.

Base case: $\mathbf{P}_{0\downarrow}$ holds trivially since $\overline{f}(t) \neq \varnothing$ for all $t \in S_2$.

Induction: Suppose τ is $\mathbf{P}_{m\downarrow}$ for some $m \geq 0$. Let $\sigma = t_0, \ldots, t_{m+1}$ be an arbitrary path in $\mathbb{G}_2$. Then there are states $s_0, \ldots, s_m \in S_1$ such that $s_i \in \overline{f}(t_i)$ for all i $(0 \leq i \leq m)$ and $s_i \in \mathcal{R}_1(s_{i-1})$ for all i $(1 \leq i \leq m)$, since τ is $\mathbf{P}_{m\downarrow}$ by assumption. Furthermore, $t_m \in f(s_m)$ and $t_{m+1} \in \mathcal{R}_2(t_m)$ so $t_{m+1} \in \mathcal{R}_2(f(s_m))$. Hence, $t_{m+1} \in f(\mathcal{R}_1(s_m))$ since τ is $\mathbf{P}_{\downarrow}$. There must thus be some $s_{m+1} \in S_1$ such that $s_{m+1} \in \mathcal{R}_1(s_m)$ and $s_{m+1} \in \overline{f}(t_{m+1})$. It follows that τ is $\mathbf{P}_{(m+1)\downarrow}$ since σ was chosen arbitrarily, which ends the induction.

$\mathbf{P_{W\downarrow}} \Rightarrow \mathbf{P_{L\downarrow}}$: Assume τ is $\mathbf{P_{W\downarrow}}$. Let $\sigma = t_0, \ldots, t_m$ be an arbitrary path in $\mathbb{G}_2$. Then there are states $s_0, \ldots, s_m \in S_1$ such that $s_i \in \overline{f}(t_i)$ for all i $(0 \leq i \leq m)$ and $s_i \in \mathcal{R}_1(s_{i-1})$ for all i $(1 \leq i \leq m)$. It follows that $s_m \in \mathcal{R}_1(s_0)$, so τ is $\mathbf{P_{L\downarrow}}$ since σ was chosen arbitrarily.

$\mathbf{P_{L\downarrow}} \Rightarrow \mathbf{P}_{1\downarrow}$: Assume τ is $\mathbf{P_{L\downarrow}}$. Let $\sigma = t_0, t_1$ be an arbitrary path of length 1 in $\mathbb{G}_2$. Then there are $s_0 \in \overline{f}(t_0)$ and $s_1 \in \overline{f}(t_1)$ such that $s_1 \in \mathcal{R}_1(s_0)$ since τ is $\mathbf{P_{L\downarrow}}$. It follows that τ is $\mathbf{P}_{1\downarrow}$ since σ was chosen arbitrarily.

(2) Proof by counterexamples.

$\mathbf{P}_{1\downarrow} \not\Rightarrow \mathbf{P_{L\downarrow}}$: Example 17(1) is $\mathbf{P}_{1\downarrow}$ but not $\mathbf{P_{L\downarrow}}$.

$\mathbf{P_{L\downarrow}} \not\Rightarrow \mathbf{P_{W\downarrow}}$: Example 17(2) $\mathbf{P_{L\downarrow}}$ but not $\mathbf{P_{W\downarrow}}$.

$\mathbf{P_{W\downarrow}} \not\Rightarrow \mathbf{P}_{\downarrow}$: Example 17(3) is $\mathbf{P_{W\downarrow}}$. However, we have that $2 \in \mathcal{R}_2(f(3))$ but that $2 \notin f(\mathcal{R}_1(3))$, so τ is not $\mathbf{P}_{\downarrow}$.

$\mathbf{P}_{\downarrow} \not\Rightarrow \mathbf{P_{S\downarrow}}$: Consider Example 17(4). First assume $s \in \{1, 3\}$. Then $\mathcal{R}_2(f(1)) = \mathcal{R}_2(f(3)) = \mathcal{R}_2(\{1\}) = \{1, 2\}$ and we have $f(\mathcal{R}_1(1)) = f(\{1, 2\}) = \{1, 2\}$ and $f(\mathcal{R}_1(3)) = f(\{3, 2\}) = \{1, 2\}$. Then assume $s \in \{2, 4\}$. Then $\mathcal{R}_2(f(2)) = \mathcal{R}_2(f(4)) = \{2\}$ and we have $f(\mathcal{R}_1(2)) = f(\{2\}) = \{2\}$ and $f(\mathcal{R}_1(4)) = f(\{4\}) = \{2\}$. Hence, $\mathcal{R}_2(f(s)) \subseteq f(\mathcal{R}_1(s))$ holds for all $s \in S_1$, and it follows that τ is $\mathbf{P}_{\downarrow}$. Then consider the path $1, 2$ in $\mathbb{G}_2$. This is not strongly refinable since there is no path from $1 \in \overline{f}(1)$ to $4 \in \overline{f}(2)$ in $\mathbb{G}_1$. Hence, the transformation is not $\mathbf{P_{S\downarrow}}$.

(3) and (4) are analogous to (1) and (2), except for the following cases where we sketch the differences.

$\mathbf{P_{S\uparrow}} \Rightarrow \mathbf{P}_{\uparrow}$: Suppose $t \in f(\mathcal{R}_1(s))$. Then there is some path $s_0, \ldots, s_m$ such that $s_0 = s$ and $t \in f(s_m)$. Since τ is $\mathbf{P_{S\uparrow}}$ there is a path from every state in $f(s_0)$ to every state in $f(s_m)$, so it follows that $t \in \mathcal{R}_2(f(s_0))$ and, thus, that τ is $\mathbf{P}_{\uparrow}$.

$\mathbf{P}_{\uparrow} \Rightarrow \mathbf{P_{W\uparrow}}$: Similar to the $\mathbf{P}_{\downarrow} \Rightarrow \mathbf{P_{W\downarrow}}$ case, but the induction step differs as follows. Suppose τ is $\mathbf{P}_{m\uparrow}$ for some $m \geq 0$. Let $\sigma = s_0, \ldots, s_{m+1}$ be an arbitrary path in $\mathbb{G}_1$. Then there are states $t_0, \ldots, t_{m+1} \in \mathbb{G}_2$ such that $t_i \in f(s_i)$ for all i $(1 \leq i \leq m+1)$ and $t_i \in \mathcal{R}_2(t_{i-1})$ for all i $(2 \leq i \leq m+1)$ since τ is $\mathbf{P}_{m\uparrow}$ by assumption. Obviously, $t_1 \in f(\mathcal{R}_1(s_0))$ so $t_1 \in \mathcal{R}_2(f(s_0))$ since τ is $\mathbf{P}_{\uparrow}$. Hence, there must exist some $t_0 \in f(s_0)$ such that $t_1 \in \mathcal{R}_2(t_0)$ and it follows that τ is $\mathbf{P}_{m+1\uparrow}$.

$\mathbf{P_{W\uparrow}} \not\Rightarrow \mathbf{P}_{\uparrow}$: Example 17(6) is $\mathbf{P_{W\uparrow}}$. However, we have that $4 \in f(\mathcal{R}_1(1))$ but that $4 \notin \mathcal{R}_2(f(1))$, so τ is not $\mathbf{P}_{\uparrow}$.

$\mathbf{P}_{\uparrow} \not\Rightarrow \mathbf{P_{S\uparrow}}$: Add the edge $\langle 1, 4, a \rangle$ to Example 17(6), which makes the example $\mathbf{P}_{\uparrow}$ but not $\mathbf{P_{S\uparrow}}$. □

The $\mathbf{P}_k$ properties form an infinite hierarchy between $\mathbf{P_W}$ and $\mathbf{P}_1$ in each direction.

Theorem 19. *For all k, m $(0 \leq m < k)$,*
(1) $\boldsymbol{P}_{k\downarrow} \Rightarrow \boldsymbol{P}_{m\downarrow}$ (2) $\boldsymbol{P}_{m\downarrow} \not\Rightarrow \boldsymbol{P}_{k\downarrow}$ (3) $\boldsymbol{P}_{k\uparrow} \Rightarrow \boldsymbol{P}_{m\uparrow}$ (4) $\boldsymbol{P}_{m\uparrow} \not\Rightarrow \boldsymbol{P}_{k\uparrow}$

Proof. (1,3) Trivial. (2) Example 17(1) is $\mathbf{P}_{1\downarrow}$ but not $\mathbf{P}_{2\downarrow}$. (4) Analogous to (2) by reversing the example. □

However, the $\mathbf{P_L}$ properties are incomparable with these latter hierarchies for all $k > 1$.

Theorem 20.
(1) $\boldsymbol{P}_{2\downarrow} \not\Rightarrow \boldsymbol{P_{L\downarrow}}$, (2) $\boldsymbol{P_{L\downarrow}} \not\Rightarrow \boldsymbol{P}_{2\downarrow}$, (3) $\boldsymbol{P}_{2\uparrow} \not\Rightarrow \boldsymbol{P_{L\uparrow}}$, (4) $\boldsymbol{P_{L\uparrow}} \not\Rightarrow \boldsymbol{P}_{2\uparrow}$.

Proof. (1) Remove the path from s_{01} to s_{11} in Fig. 2(b). The result is $\mathbf{P}_{2\downarrow}$ but not $\mathbf{P_{L\downarrow}}$ since the path t_0, t_1, t_2, t_3 is not loosely downwards state refinable.

(2) Example 17(2) is $\mathbf{P_{L\downarrow}}$ but not $\mathbf{P}_{2\downarrow}$.

(3–4) Analogous to (1–2). □

The two types of hierarchies are illustrated for the downwards direction in Fig. 4 (note that $\mathbf{P_{W\downarrow}} \Rightarrow \mathbf{P}_{k\downarrow}$ for all k by definition), where the arrows denote the $\Rightarrow$ and $\not\Rightarrow$ relationships between the properties. All these hierarchies collapse if the transformation also has property **M** in the same direction.

Theorem 21. *(1) $\boldsymbol{M}_{\downarrow}\boldsymbol{P}_{1\downarrow} \Rightarrow \boldsymbol{P_{S\downarrow}}$, (2) $\boldsymbol{M}_{\uparrow}\boldsymbol{P}_{1\uparrow} \Rightarrow \boldsymbol{P_{S\uparrow}}$.*

Proof. (1) Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ be two arbitrary STGs and let $\tau = \langle f, R \rangle$ be an arbitrary $\mathbf{M}_\downarrow \mathbf{P}_{1\downarrow}$ transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$. Let $\sigma = t_0, \ldots, t_n$ be an arbitrary path in $\mathbb{G}_2$. For each i $(0 \leq i \leq n)$ there is a unique state $s_i \in S_1$ such that $\overline{f}(t_i) = \{s_i\}$, since τ is $\mathbf{M}_\downarrow$. Hence, it follows that $s_i \in \mathcal{R}_1(s_{i-1})$ for all i $(1 \leq i \leq n)$ since τ is $\mathbf{P}_{1\downarrow}$. It follows that τ is $\mathbf{P}_{\mathbf{S}\downarrow}$ since $\overline{f}(t_i) = \{s_i\}$ for all i $(1 \leq i \leq n)$ and σ was chosen arbitrarily.

(2) Analogous. □

Also the other properties in Definition 9 have some implications for refinability. In particular, we can use them for proving general properties for homomorphic abstractions (Corollary 23). We first consider connections between refinement properties and the properties $\mathbf{R}_\uparrow/\mathbf{R}_\downarrow$ and $\mathbf{C}_\uparrow/\mathbf{C}_\downarrow$.

Theorem 22.
(1) $\boldsymbol{R}_\downarrow \boldsymbol{C}_\downarrow \Rightarrow \boldsymbol{P}_{\boldsymbol{1}\downarrow}$, *(2)* $\boldsymbol{P}_{1\downarrow} \not\Rightarrow \boldsymbol{R}_\downarrow \boldsymbol{C}_\downarrow$, *(3)* $\boldsymbol{P}_{\boldsymbol{S}\downarrow} \boldsymbol{R}_\downarrow \not\Rightarrow \boldsymbol{C}_\downarrow$
(4) $\boldsymbol{R}_\uparrow \boldsymbol{C}_\uparrow \Rightarrow \boldsymbol{P}_{\boldsymbol{1}\uparrow}$, *(5)* $\boldsymbol{P}_{1\uparrow} \not\Rightarrow \boldsymbol{R}_\uparrow \boldsymbol{C}_\uparrow$ *(6)* $\boldsymbol{P}_{\boldsymbol{S}\uparrow} \boldsymbol{R}_\uparrow \not\Rightarrow \boldsymbol{C}_\uparrow$

Proof. We prove only (1–3) since (4–6) are analogous.

(1) Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ be two arbitrary STGs and let $\tau = \langle f, R \rangle$ be an arbitrary $\mathbf{R}_\downarrow \mathbf{C}_\downarrow$ transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$. Let $\sigma = t, t'$ be an arbitrary path of length 1 in $\mathbb{G}_2$, i.e. there is some arc $\langle t, t', \ell \rangle \in E_2$. Then there is some label $\ell' \in L(E_1)$ such that $R(\ell', \ell)$, since τ is $\mathbf{R}_\downarrow$. Hence, there is some arc $\langle s, s', \ell' \rangle \in E_1$ such that $s \in \overline{f}(t)$ and $s' \in \overline{f}(t')$ since τ is $\mathbf{C}_\downarrow$. It follows that $s' \in \mathcal{R}_1(s)$ and, thus that τ is $\mathbf{P}_{1\downarrow}$ since σ was chosen arbitrarily.

(2) Example 17(5) is $\mathbf{P}_{1\downarrow}$, but not $\mathbf{R}_\downarrow \mathbf{C}_\downarrow$.

(3) Example 17(5) is $\mathbf{P}_{\mathbf{S}\downarrow}$ and $\mathbf{R}_\downarrow$, but not $\mathbf{C}_\downarrow$. □

Combining Theorems 21 and 22 gives the following corollary.

Corollary 23. *(1)* $\boldsymbol{M}_\downarrow \boldsymbol{R}_\downarrow \boldsymbol{C}_\downarrow \Rightarrow \boldsymbol{P}_{\boldsymbol{S}\downarrow}$, *(2)* $\boldsymbol{M}_\uparrow \boldsymbol{R}_\uparrow \boldsymbol{C}_\uparrow \Rightarrow \boldsymbol{P}_{\boldsymbol{S}\uparrow}$.

Combining this corollary with Theorem 14 yields that all homomorphic abstractions have property $\mathbf{P}_{\mathbf{S}\uparrow}$.

These results will be used later, in Sec. 7, where we will analyse a number of abstraction methods, deriving some properties explicitly and then infer other properties using the results above.

5. SAS$^+$ and implicit state-transition graphs

So far, we have only considered STGs as explicit graphs. The usual case in search and planning is to model an STG implicitly, using variables to define the state space and actions (or operators) to define the arcs. The two most common formalisms for this purpose are propositional STRIPS [17] and SAS$^+$ [10,11]. The major difference between these is that STRIPS models the state space by a set of propositional atoms, i.e. binary variables, while SAS$^+$ uses variables with arbitrary finite domains. Actions are modelled in essentially the same way. Each action has a precondition that tells when the action can be applied and a postcondition that models the effect of applying the action. These formalisms are known to be equivalent under a polynomial reduction that preserves the solution lengths [4]. While STRIPS is modelled using sets of atoms, SAS$^+$ is modelled using vectors allowing undefined values. To be able to mix these formalisms freely and allow for more direct comparisons, we use a different way to model SAS$^+$ here, based on sets of multi-valued atoms. It should be stressed that this is *not* a new planning formalism, but only an alternative definition of an old one; it is still SAS$^+$.

Definition 24. A *variable set* V is a finite set of objects called *variables* and a *domain function* D for V is a function that maps every variable $v \in V$ to a corresponding finite *domain* $D(v)$ of values. An *atom* over V and D is a pair $\langle v, x \rangle$ (usually written as $(v = x)$) such that $v \in V$ and $x \in D(v)$. A *state* is a set of atoms and $V \cdot D = \bigcup_{v \in V} (\{v\} \times D(v))$ denotes the *full state* (the set of all possible atoms over V and D). A state $s \subseteq V \cdot D$ is

1. *consistent* if each $v \in V$ occurs in at most one atom in s,
2. *total* if each $v \in V$ occurs in exactly one atom in s.

The filter functions $\mathcal{T}$ and $\mathcal{C}$ are defined for all $S \subseteq V \cdot D$ as:

1. $\mathcal{C}(S) = \{s \subseteq S \mid s \text{ is consistent}\}$.
2. $\mathcal{T}(S) = \{s \subseteq S \mid s \text{ is total}\}$.

Example 25. Let $V = \{u, v\}$, $D(u) = \{0, 1\}$ and $D(v) = \{0, 1, 2\}$. Then

$$V \cdot D = \{\, (u = 0),\ (u = 1),\ (v = 0),\ (v = 1),\ (v = 2) \,\},$$

$$\begin{aligned}\mathcal{T}(V\cdot D) = \{\,&\{(u=0),(v=0)\},\ \{(u=0),(v=1)\},\ \{(u=0),(v=2)\},\\ &\{(u=1),(v=0)\},\ \{(u=1),(v=1)\},\ \{(u=1),(v=2)\}\,\}\\ \mathcal{C}(V\cdot D) = \{\,&\{(u=0),(v=0)\},\ \{(u=0),(v=1)\},\ \{(u=0),(v=2)\},\\ &\{(u=1),(v=0)\},\ \{(u=1),(v=1)\},\ \{(u=1),(v=2)\},\\ &\{(u=0)\},\ \{(u=1)\},\ \{(v=0)\},\ \{(v=1)\},\ \{(v=2)\},\ \varnothing\,\}.\end{aligned}$$

Definition 26. For arbitrary $s, t \in \mathcal{C}(V\cdot D)$, $U \subseteq V$ and $v \in V$:
(1) $\mathrm{vars}(s) = \{v \mid (v=x) \in s\}$, (2) $s[U] = s \cap (U\cdot D)$,
(3) $s[v] = s[\{v\}]$, (4) $s \ltimes t = s[V \setminus \mathrm{vars}(t)] \cup t$,
(5) $s^{:=c} = \{(v=c) \mid (v=c) \in s\}$ (6) $U^{:=c} = \{(v=c) \mid v \in U\}$.

That is, $\mathrm{vars}(s)$ is the set of variables occurring in atoms in s, $s[U]$ is the projection of s to the subset of atoms with variables in U, $s \ltimes t$ is the *update* operator, $s^{:=c}$ selects those atoms in s that have variable value c and $U^{:=c}$ creates a state with an atom for each variable in U, which are all set to value c. The operator $\ltimes$ is typically used for updating a state s with the postcondition of an action a; this will be made formally clear in Definition 29.

Example 27. Let $s = \{(u=0),(v=1)\}$ and $t = \{(v=2)\}$ be two states. Then $\mathrm{vars}(s) = \{u, v\}$, $s[\{u,v\}] = s$, $s[v] = \{(v=1)\}$, $s \ltimes t = \{(u=0),(v=2)\}$, $s^{:=1} = \{(v=1)\}$ and $\mathrm{vars}(s)^{:=1} = \{(u=1),(v=1)\}$.

Unless otherwise specified, states will be assumed total and we will usually write state rather than total state. As a convention, we will often tacitly assume that binary variables have the domain $\{0,1\}$ and use the shorthands $\overline{v}$ for the *negative atom* $(v=0)$ and v for the *positive atom* $(v=1)$.

The following observations will be tacitly used henceforth.

Proposition 28. *Let V be a variable set, let D be a domain function for V and let $s, t \in \mathcal{C}(V\cdot D)$. Then:*

1. $s[U] \subseteq s$ *for all* $U \subseteq V$.
2. $s \subseteq t \Rightarrow s[U] \subseteq t[U]$ *for all* $U \subseteq V$.
3. $s = t \Rightarrow s[U] = t[U]$ *for all* $U \subseteq V$.
4. $s[V_1] \cup s[V_2] = s[V_1 \cup V_2]$ *for all* $V_1, V_2 \subseteq V$.
5. $s[U] \cup t[U] = (s \cup t)[U]$,
6. $s[U] \ltimes t[U] = (s \ltimes t)[U]$,
7. *If s is total, then $s \ltimes t$ is total.*
8. $s[V_1][V_2] = s[V_1 \cap V_2] = s[V_2][V_1]$ *for all* $V_1, V_2 \subseteq V$.
9. $V_1 \subseteq V_2 \Rightarrow s[V_1][V_2] = s[V_1]$ *for all* $V_1, V_2 \subseteq V$.

We define SAS$^+$ frames and instances as follows.

Definition 29. A *SAS$^+$ frame* is a triple $\mathbb{F} = \langle V, D, A\rangle$ where V is a variable set, D is a domain function for V and A is a finite set of *actions*. Each action $a \in A$ has a *precondition* $\mathrm{pre}(a) \in \mathcal{C}(V\cdot D)$ and a *postcondition* $\mathrm{post}(a) \in \mathcal{C}(V\cdot D)$. The STG $\mathbb{G}(\mathbb{F}) = \langle S, E\rangle$ for $\mathbb{F}$ is defined such that

1. $S = \mathcal{T}(V\cdot D)$ and
2. $E = \{\langle s, t, a\rangle \mid a \in A, \mathrm{pre}(a) \subseteq s \text{ and } t = s \ltimes \mathrm{post}(a)\}$.

A sequence $\omega = a_1, \ldots, a_k$ of actions in A is a *plan* from a state $s \in S$ to a state $t \in S$ if ω is a label path from s to t in $\mathbb{G}(\mathbb{F})$.

A *SAS$^+$ instance* is a 5-tuple $\mathbb{P} = \langle V, D, A, I, G\rangle$ where $\langle V, D, A\rangle$ is a SAS$^+$ frame, $I \in \mathcal{T}(V\cdot D)$ is the *initial state* and $G \in \mathcal{C}(V\cdot D)$ is the *goal*. A sequence $\omega = a_1, \ldots, a_k$ of actions in A is a *plan* for $\mathbb{P}$ if it is a plan from I to some state s such that $G \subseteq s$.

STRIPS can be viewed as a special case of SAS$^+$ where all variables are binary.[4] We will often use the notation $a\colon \mathit{pre} \Rightarrow \mathit{post}$ to compactly define an action a and its pre- and postconditions.

A SAS$^+$ frame $\mathbb{F} = \langle V, D, A\rangle$ is an implicit specification of the STG $\mathbb{G}(\mathbb{F}) = \langle S, E\rangle$, where the actions are used as labels. It is furthermore a compact representation since $\mathbb{G}(\mathbb{F})$ can be exponentially larger than $\mathbb{F}$, because $S = \mathcal{T}(V\cdot D)$ and an

[4] The classical variant of STRIPS only allows positive preconditions, but it is common to also allow negative preconditions.

action in A can induce up to $|S|$ arcs in E. That is, a SAS$^+$ instance is a compact graph representation in the sense of Galperin and Wigderson [35] and of Balcázar [12].

A SAS$^+$ instance $\mathbb{P} = \langle V, D, A, I, G\rangle$ is more specific than the corresponding frame $\mathbb{F} = \langle V, D, A\rangle$, and we may define a corresponding extended STG $\langle S, E, I, G\rangle$, where $\langle S, E\rangle = \mathbb{G}(\mathbb{F})$. Such a structure is sometimes called a *transition system* (cf. [49]), which is briefly discussed in Section 10. Note that the STG concept and most of our results are still useful for more advanced planning formalisms. Conditional actions, where the effect of an action depends on the state is applied in, would still result in a straightforward STG, as would a planning language with axioms for causal effects of actions; in both cases the resulting state is still a function of the state we apply an action in. Even non-deterministic actions can be modelled with STGs; the only difference is that a state may have more than one outgoing arc with the same label, which is not prohibited. Other examples of languages for planning and search are PDDL [71] and PSVN [55].

Since there is a one-to-one correspondence between SAS$^+$ frames and STGs, it is straightforward to say that a transformation τ is a transformation from a SAS$^+$ frame $\mathbb{F}_1$ to a SAS$^+$ frame $\mathbb{F}_2$ if it is a transformation from $\mathbb{G}(\mathbb{F}_1)$ to $\mathbb{G}(\mathbb{F}_2)$.

6. Abstraction in planning

The goal of this section is to model a number of different abstraction and abstraction-like methods from the literature within our framework. We note that even though the methods are quite different, they can all be modelled in a highly uniform and reasonably succinct way. For instance, labels will exclusively be used for keeping track of action names in all examples. This coherent way of defining the methods makes it possible to systematically study their intrinsic properties; something that will be carried out in the next section.

Each of the methods we will study can be viewed as a function that takes a SAS$^+$ frame $\mathbb{F}_1 = \langle V, D, A\rangle$ and some extra information and maps this to a new SAS$^+$ frame $\mathbb{F}_2$. For example, the **ABS** method (that will be formally introduced in Section 6.1) could be viewed as a function α that takes a frame $\mathbb{F}_1$, a subset V_C of the variables and a function g on the set of actions, and then constructs a new frame $\mathbb{F}_2 = \alpha(\mathbb{F}_1, V_C, g)$. However, different methods require different type and amount of such extra information, so this approach would quickly result in a plethora of similar, yet different, transformation concepts, which is rather the opposite of our aims. Hence, we will instead take a relational view where we start with two frames $\mathbb{F}_1$ and $\mathbb{F}_2$ and a transformation τ from $\mathbb{G}(\mathbb{F}_1)$ to $\mathbb{G}(\mathbb{F}_2)$, and then say that τ is an **ABS** transformation from $\mathbb{F}_1$ to $\mathbb{F}_2$ if there exists a choice of the extra information V_C and g such that $\mathbb{F}_2 = \alpha(\mathbb{F}_1, V_C, g)$. Hence, we can focus on the resulting properties of the underlying transformation τ from $\mathbb{G}(\mathbb{F}_1)$ to $\mathbb{G}(\mathbb{F}_2)$. Furthermore, since the extra information is often a free choice, and not a function of $\mathbb{F}_1$, it will be essential later on, when we study compositions of transformations, that we use this relational view. This is also why we have chosen a relational rather than functional view on transformations from the very start.

6.1. Abstrips-style abstraction

The first clearly described case of abstraction in planning is the Abstrips planner [78], which was a version of the Strips planner using state abstraction. The abstraction method used in Abstrips is to identify a subset of the atoms as critical and make an abstraction by restricting the preconditions of all actions to only these critical atoms while leaving everything else unaltered. The intention is that the critical atoms should be more important and that once an abstract plan is found, it should be easy to fill in the missing actions to take all atoms into account. This method has also been used elsewhere, for instance, in the AbTweak planner [3], and it is a special case of the broader class of abstractions known as *precondition relaxation*. We will refer to this type of abstraction as method **ABS**, which we define formally as follows.

Definition 30 (***ABS***). Let $\mathbb{F}_1 = \langle V_1, D_1, A_1\rangle$ and $\mathbb{F}_2 = \langle V_2, D_2, A_2\rangle$ be two SAS$^+$ frames with corresponding STGs $\mathbb{G}_1 = \langle S_1, E_1\rangle$ and $\mathbb{G}_2 = \langle S_2, E_2\rangle$. Let $\tau = \langle f, R\rangle$ be a transformation from $\mathbb{F}_1$ to $\mathbb{F}_2$. Then, τ is an **ABS** transformation from $\mathbb{F}_1$ to $\mathbb{F}_2$ if there is a set of critical variables $V_C \subseteq V_1$ and a bijection $g : A_1 \to A_2$ such that the following holds:

1. $V_2 = V_1$, $D_2 = D_1$,
2. $\mathrm{pre}(g(a)) = \mathrm{pre}(a)[V_C]$ and $\mathrm{post}(g(a)) = \mathrm{post}(a)$ for all $a \in A_1$.
3. $f(s) = \{t \in S_2 \mid s[V_C] = t[V_C]\}$ for all $s \in S_1$.
4. $R = \{\langle a, g(a)\rangle \mid a \in A_1\}$.

The original Abstrips planner used a different set of critical variables for each action.[5] We simplify this to one single set, V_C, both in order to simplify the presentation and analysis somewhat, but also to make the comparison with the following method clearer. This simplification does not alter any of results we show; we could define separate sets $V_C(a)$ of critical atoms for each action and set $V_C = \bigcap_{a\in A} V_C(a)$. We also note that **ABS** is an embedding-style abstraction since $E_1 \subseteq E_2$ must hold, although it is a form of generalised embedding since f is not $\mathbf{M}_\uparrow$.

[5] More precisely, Abstrips uses an abstraction hierarchy and a criticality value is assigned to each precondition atom to tell at which levels it is critical.

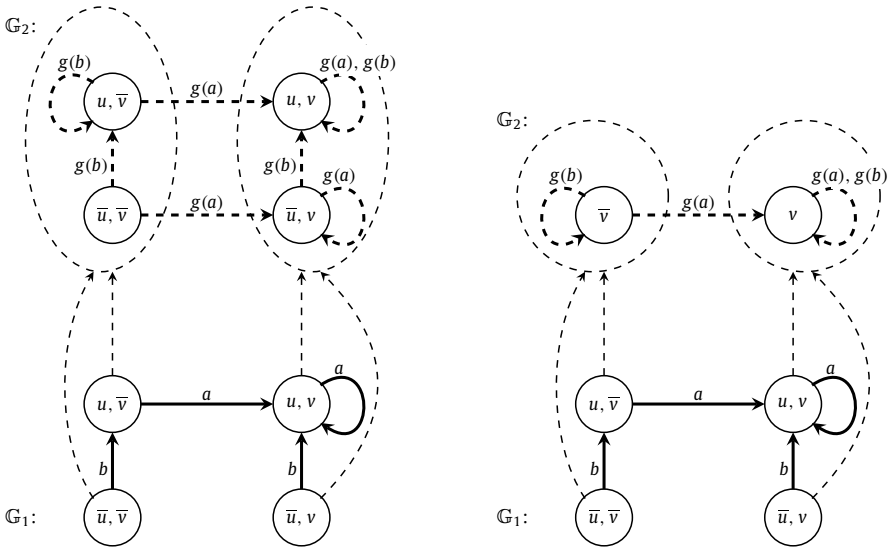


Fig. 5. An **ABS** abstraction (left) and an **VP** abstraction (right) of the same instance ($V_C = \{v\}$).

Example 31. Let $V_1 = \{u, v\}$, where both variables are binary, let $A_1 = \{a\}$, where $a\colon u \Rightarrow v$ and $b\colon \overline{u} \Rightarrow u$. First consider an **ABS** abstraction with $V_C = \{v\}$, i.e. v is the only critical variable. All states in S_1 remain in S_2 and the abstract actions are $g(a)\colon \varnothing \Rightarrow v$ and $g(b)\colon \varnothing \Rightarrow u$. That is, the abstract actions induce twice as many arcs as the ground actions in this case. This is illustrated in Fig. 5 (left).

6.2. Variable projection

Knoblock [65] took this method further in planning by removing the non-critical atoms everywhere, not only in preconditions. This is a general and common technique otherwise known as *variable projection*. It is commonly used also in search (cf. the article by Zilles and Holte [92]), as well as in other areas such as model checking [20]. It is also the technique underlying pattern database heuristics in search and planning [23,45]. We refer to this as method **VP**.

Definition 32 (***VP***). Let $\mathbb{F}_1 = \langle V_1, D_1, A_1\rangle$ and $\mathbb{F}_2 = \langle V_2, D_2, A_2\rangle$ be two SAS$^+$ frames with corresponding STGs $\mathbb{G}_1 = \langle S_1, E_1\rangle$ and $\mathbb{G}_2 = \langle S_2, E_2\rangle$. Let $\tau = \langle f, R\rangle$ be a transformation from $\mathbb{F}_1$ to $\mathbb{F}_2$. Then, τ is a **VP** transformation from $\mathbb{F}_1$ to $\mathbb{F}_2$ if there is a set of critical variables $V_C \subseteq V_1$ and a bijection $g : A_1 \to A_2$ such that the following holds:

1. $V_2 = V_C$, $D_2 = D_{1|V_C}$,
2. $\mathrm{pre}(g(a)) = \mathrm{pre}(a)[V_C]$ and $\mathrm{post}(g(a)) = \mathrm{post}(a)[V_C]$, for all $a \in A_1$.
3. $f(s) = \{s[V_C]\}$ for all $s \in S_1$.
4. $R = \{\langle a, g(a)\rangle \mid a \in A\}$.

We note that this is a homomorphic abstraction where $|E_2| \leq |E_1|$ holds.

Example 33. Consider the same frame as in Example 31. Then consider a **VP** abstraction, also with $V_C = \{v\}$. In this case the state space shrinks, since variable u disappears entirely. We get the abstract actions $g(a)\colon \varnothing \Rightarrow v$ and $g(b)\colon \varnothing \Rightarrow \varnothing$, but no abstract action induces more arcs than its corresponding ground action since the state space shrinks. This is illustrated in Fig. 5 (right). Note that $g(b)$ is redundant and can usually be removed.

6.3. Variable-domain abstraction

Another abstraction technique is *variable-domain abstraction*, which we will refer to as method **VDA**. Instead of removing or disregarding certain variables, this method reduces the domain of one or more variables by collapsing values into new abstract values. This method is used both in planning [26,42], search [58,92] and model checking [20]. One may also note that the technique used in abstract interpretation [21] of abstracting integers to the domain $\{-, 0, +\}$ is domain abstraction. As we will see later, it is also one of the steps in the merge and shrink method [49].

Definition 34 (***VDA***). Let $\mathbb{F}_1 = \langle V_1, D_1, A_1\rangle$ and $\mathbb{F}_2 = \langle V_2, D_2, A_2\rangle$ be two SAS$^+$ frames. Let $\mathbb{G}_1 = \langle S_1, E_1\rangle$ and $\mathbb{G}_2 = \langle S_2, E_2\rangle$ be the corresponding STGs. Let $\tau = \langle f, R\rangle$ be a transformation from $\mathbb{F}_1$ to $\mathbb{F}_2$. Then, τ is a **VDA** transformation if there is a

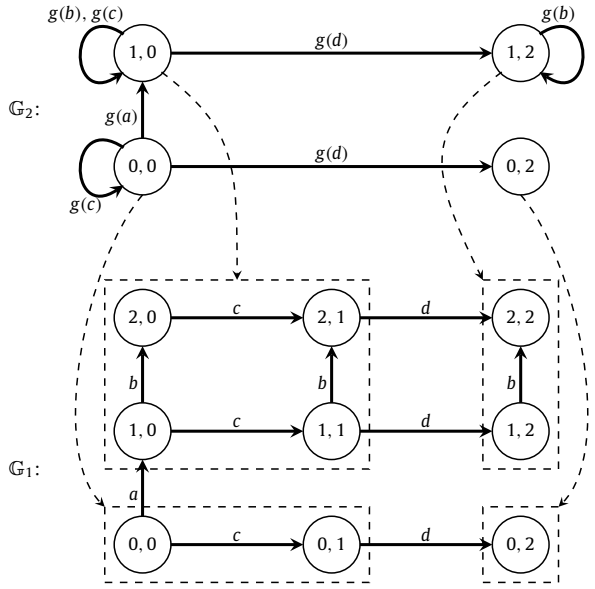


Fig. 6. Domain abstraction. The digits x, y in a state mean $(u = x), (v = y)$. For clarity, the figure shows $\overline{f}$, not f.

family of surjective functions $H = \{h_v : D_1(v) \to D_2(v) \mid v \in V_1\}$ and a bijection $g : A_1 \to A_2$ such that the conditions below hold, where the function $h : S_1 \to S_2$ is defined as $h(s) = \{(v = h_v(x)) \mid (v = x) \in s\}$.

1. $V_2 = V_1$, $D_2(v) = h_v(D_1(v))$ for all $v \in V_1$,
2. $\text{pre}(g(a)) = h(\text{pre}(a))$ and $\text{post}(g(a)) = h(\text{post}(a))$, for all $a \in A_1$.
3. $f(s) = h(s)$ for all $s \in S_1$.
4. $R = \{\langle a, g(a)\rangle \mid a \in A\}$.

Example 35. Let $V_1 = \{u, v\}$, where $D(u) = D(v) = \{0, 1, 2\}$, and let $A = \{a, b, c, d\}$, where $a\colon (u = 0), (v = 0) \Rightarrow (u = 1)$, $b\colon (u = 1) \Rightarrow (u = 2)$, $c\colon (v = 0) \Rightarrow (v = 1)$ and $d\colon (v = 1) \Rightarrow (v = 2)$. Define the abstraction functions h_u and h_v such that $h_u(0) = 0$, $h_u(1) = h_u(2) = 1$, $h_v(0) = h_v(1) = 0$ and $h_v(2) = 2$. This yields the abstract actions $g(a)\colon (u = 0), (v = 0) \Rightarrow (u = 1)$, $g(b)\colon (u = 1) \Rightarrow (u = 1)$, $g(c)\colon (v = 0) \Rightarrow (v = 0)$ and $g(d)\colon (v = 0) \Rightarrow (v = 2)$. This is shown in Fig. 6. Note that this figure shows the reverse mapping $\overline{f}$, rather than f, which is more illustrative here since it shows how the domain abstraction partitions S_1.

6.4. Removing redundant actions

As a response to a common belief that it is good for a planner to have many choices, Haslum and Jonsson [46] showed that it may be more efficient to have as few choices as possible, in other words, they suggested to use retractions rather than embeddings. They proposed removing some, or all, of the redundant actions. While the authors did not present this as an abstraction, it is quite reasonable to view it as such: we abstract away redundant information by removing redundant actions.[6] We refer to this method as *Removing Redundant Actions (**RRA**)*. The original paper considered various degrees of avoiding redundancy so we define two extreme cases, **RRAa** and **RRAb**, differing in condition 3 below.

Definition 36 (***RRA***). Let $\mathbb{F}_1 = \langle V_1, D_1, A_1\rangle$ and $\mathbb{F}_2 = \langle V_2, D_2, A_2\rangle$ be two SAS$^+$ instances with corresponding STGs $\mathbb{G}_1 = \langle S_1, E_1\rangle$ and $\mathbb{G}_2 = \langle S_2, E_2\rangle$. Let $\tau = \langle f, R\rangle$ be a transformation from $\mathbb{F}_1$ to $\mathbb{F}_2$. Then, τ is an **RRA** transformation from $\mathbb{F}_1$ to $\mathbb{F}_2$ if the following holds:

1. $V_2 = V_1$, $D_2 = D_1$,
2. $A_2 \subseteq A_1$.
3. (a) $\{\langle s, t\rangle \mid \exists \ell.\langle s, t, \ell\rangle \in E_1\} = \{\langle s, t\rangle \mid \exists \ell.\langle s, t, \ell\rangle \in E_2\}$ (for **RRAa**).
   (b) $\{\langle s, t\rangle \mid t \in \mathcal{R}_1(s)\} = \{\langle s, t\rangle \mid t \in \mathcal{R}_2(s)\}$ (for **RRAb**).

[6] Heusner et al. [50] suggested a new refinement method that first reduces the available actions to a small subset, and then incrementally extends this until a plan can be found.

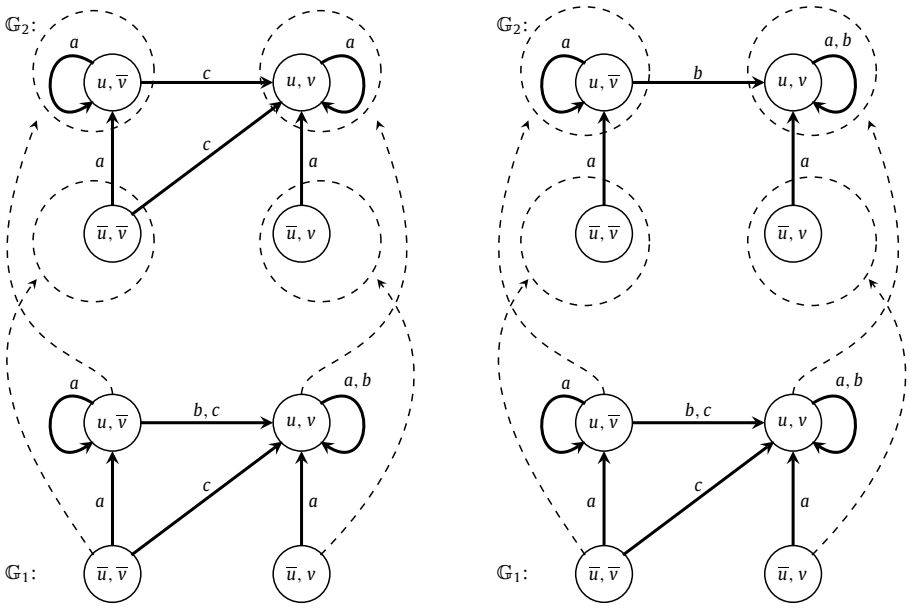


Fig. 7. An **RRAa** abstraction (left) and an **RRAb** abstraction (right) of the same instance.

4. f is the identity function.
5. $R = \{\langle a, a\rangle \mid a \in A_2\}$.

Variant 3a (**RRAa**) says that if an action $a \in A_1$ induces an arc from s to t in the STG and we remove a, then there must be some remaining action that induces an arc from s to t. Variant 3b (**RRAb**), on the other hand, only requires that there is still a path from s to t. Conversely, suppose $\langle s, t, a\rangle \in E_2$ for some action $a \in A_2$. Then it is necessary that also $\langle s, t, a\rangle \in E_1$ due to condition (2). That is, for both variants it holds that $E_2 \subseteq E_1$, so they are retraction-style abstractions. We also note that method **RRAa** preserves the optimal path length between all pairs of states, which is not the case for **RRAb**.

Example 37. Let $V_1 = \{u, v\}$, where both variables are binary, let $A_1 = \{a, b, c\}$, where $a\colon \varnothing \Rightarrow u$, $b\colon u \Rightarrow v$ and $c\colon \overline{v} \Rightarrow u, v$. Fig. 7 (left) illustrates the effect of removing action b (multiple arcs with different labels are shown as one arc with multiple labels). All arcs remain, if we ignore the labels, so this is an **RRAa** abstraction. Fig. 7 (right) illustrates the effect of removing action c instead. Then there is no longer any arc from $\{\overline{u}, \overline{v}\}$ to $\{u, v\}$. However, there is still a path from $\{\overline{u}, \overline{v}\}$ to $\{u, v\}$, so this is an **RRAb** abstraction.

6.5. Ignoring delete lists

The idea of removing the negative postconditions from all actions in STRIPS [14,72] is known as *ignoring delete lists* or *delete relaxation*. This means that false atoms can be set to true, but not vice versa. The method is commonly used as an abstraction in planning, where the length of an optimal plan in this abstraction is used as an estimate for the length of an optimal ground plan, which is known as the h^+ heuristic [51]. While this method was originally defined in the classical variant of STRIPS where actions have only positive preconditions, there is no problem in principle to allow also negative preconditions. To make this distinction clear, we will refer to this latter variant as *Generalised Ignoring Delete Lists* (**GIDL**). Our forthcoming discussion will apply to both variants since we do not make explicit use of negative preconditions.

Definition 38 (***GIDL***). Let $\mathbb{F}_1 = \langle V_1, D_1, A_1\rangle$ and $\mathbb{F}_2 = \langle V_2, D_2, A_2\rangle$ be two SAS$^+$ instances with corresponding STGs $\mathbb{G}_1 = \langle S_1, E_1\rangle$ and $\mathbb{G}_2 = \langle S_2, E_2\rangle$. Let $\tau = \langle f, R\rangle$ be a transformation from $\mathbb{F}_1$ to $\mathbb{F}_2$. Then τ is a **GIDL** transformation from $\mathbb{F}_1$ to $\mathbb{F}_2$ if there is a bijection $g : A_1 \to A_2$ such that the following holds:

1. $V_2 = V_1$, $D_2 = D_1$,
2. $\mathrm{pre}(g(a)) = \mathrm{pre}(a)^{=1}$ and $\mathrm{post}(g(a)) = \mathrm{post}(a)^{=1}$ for all $a \in A_1$.
3. f is the identity function.
4. $R = \{\langle a, g(a)\rangle \mid a \in A_1\}$.

For actual instances $\mathbb{P}_1 = \langle V_1, D_1, A_1, I_1, G_1\rangle$ and $\mathbb{P}_2 = \langle V_2, D_2, A_2, I_2, G_2\rangle$, where $I_2 = I_1$ and $G_2 = {G_1}^{=1}$ since no plan for $\mathbb{P}_2$ can achieve a negative goal on a variable that is set to true at some point in the plan. Keeping negative preconditions

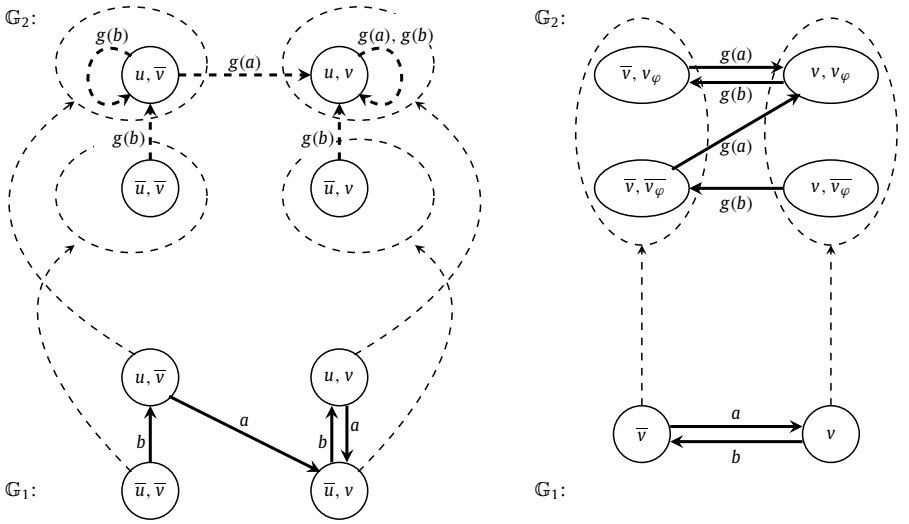


Fig. 8. An **GIDL** abstraction (left) and an **DLBS** abstraction (right).

in the transformation is possible, but the transformation will then lose some favourable properties since abstract plans may be non-refinable and also unusable for admissible heuristics. For a **GIDL** abstraction, there is no clear relationship between E_1 and E_2 at all. They may even be disjoint (consider an action $a : u, \overline{v} \Rightarrow u, \overline{v}$).

Example 39. Let $V_1 = \{u, v\}$, where both variables are binary, and let $A_1 = \{a, b\}$, where $a : u \Rightarrow \overline{u}, v$ and $b : \overline{u} \Rightarrow u$. The **GIDL** abstraction is shown in Fig. 8 (left), where $g(a) : u \Rightarrow v$ and $g(b) : \varnothing \Rightarrow u$. Let $\{\overline{u}, \overline{v}\}$ be the initial state and $\{u, v\}$ the desired goal state. Then b, a, b is a plan in $\mathbb{G}_1$, while the shorter plan $g(b), g(a)$ suffices in the abstraction $\mathbb{G}_2$.

Monotonic abstractions on domains with more than two domain values have recently been intensively studied in the literature, cf. the article by Domshlak et al. [25] and the references therein. While **GIDL** may be viewed as *the* monotonic abstraction for two-valued domains, there are various options for larger domains. In particular, the *accumulation* semantics [28,41] has become very popular and it serves as the basis for interesting approaches such as *red-black relaxation* [25]. Analysing such methods with the aid of our framework appears to be an interesting future research direction.

6.6. Direct landmark-based surrogates

A *landmark* is a condition that is somehow known to be a necessary subgoal for a plan. Landmarks are sometimes added as separate information to planners as guidance for how to solve a particular instance [53]. Domshlak et al. [27] suggested to combine abstraction with landmarks by encoding the landmarks explicitly in the instance—the idea is to guide the abstraction process via the landmarks. Based on this idea, they propose a new way for constructing abstraction heuristics and empirically show that this approach can produce high-quality heuristic functions. They considered *disjunctive landmarks*, i.e. each landmark is a set of atoms interpreted such that at least one of its members must be achieved at some time. This method is known as the *direct landmark-based surrogate* method, which we refer to as **DLBS**.

Definition 40 (***DLBS***). Let $\mathbb{F}_1 = \langle V_1, D_1, A_1 \rangle$ and $\mathbb{F}_2 = \langle V_2, D_2, A_2 \rangle$ be two SAS$^+$ instances with corresponding STGs $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_2, E_2 \rangle$. Let $\tau = \langle f, R \rangle$ be a transformation from $\mathbb{F}_1$ to $\mathbb{F}_2$. Then, τ is a **DLBS** transformation from $\mathbb{F}_1$ to $\mathbb{F}_2$ if there is a set $M \subseteq 2^{V_1 \cdot D_1}$ of *landmarks* and a bijection $g : A_1 \to A_2$ such that conditions (1)–(5) below hold: First define the variable set $V_M = \{v_\varphi \mid \varphi \in M\}$ with domain function $D_M : V_M \to \{0, 1\}$ and for each $a \in A_1$, define $\text{post}_M(a) = \{(v_\varphi = 1) \mid \varphi \in M \cap \text{post}(a)\}$

1. $V_2 = V_1 \cup V_M$, $D_2 = D_1 \cup D_M$,
2. $\text{pre}(g(a)) = \text{pre}(a)$ and $\text{post}(g(a)) = \text{post}(a) \cup \text{post}_M(a)$ for all $a \in A_1$.
3. $f(s) = \{s \cup m \mid m \in \mathcal{T}(V_M \cdot D_M)\}$ for all $s \in S_1$.
4. $R = \{\langle a, g(a) \rangle \mid a \in A_1\}$.

For actual instances $\mathbb{P}_1 = \langle V_1, D_1, A_1, I_1, G_1 \rangle$ and $\mathbb{P}_2 = \langle V_2, D_2, A_2, I_2, G_2 \rangle$, we must also make the transformations $I_2 = I \cup \{(v_\varphi = 1) \mid \varphi \in M \cap I\}$ and $G_2 = G_1 \cup V_M{}^{:=1}$, if we want to enforce that every plan achieves all landmarks in M. Note that landmarks may contain conflicting atoms, and that even a single disjunctive landmark may do so. Consider a landmark $\varphi = \{(v = 1), (v = 3)\}$, which requires that every plan achieves at least one of the values $(v = 1)$ and $(v = 3)$ at

Table 1
Properties of the planning abstraction methods.

Type of method	Properties		
	Theorems 42 & 43	Theorem 44	Theorem 45
ABS	$\mathbf{R}_{\updownarrow}\mathbf{C}_{\updownarrow}$	$\mathbf{P}_{1\downarrow}$, $\mathbf{P}_{1\uparrow}$	not $\mathbf{P}_{\mathbf{L}\downarrow}$, $\mathbf{P}_{\uparrow}$, not $\mathbf{P}_{\mathbf{S}\uparrow}$
VP	$\mathbf{M}_{\uparrow}\mathbf{R}_{\updownarrow}\mathbf{C}_{\updownarrow}$	$\mathbf{P}_{1\downarrow}$, $\mathbf{P}_{\mathbf{S}\uparrow}$	not $\mathbf{P}_{\mathbf{L}\downarrow}$
VDA	$\mathbf{M}_{\uparrow}\mathbf{R}_{\updownarrow}\mathbf{C}_{\updownarrow}$	$\mathbf{P}_{1\downarrow}$, $\mathbf{P}_{\mathbf{S}\uparrow}$	not $\mathbf{P}_{\mathbf{L}\downarrow}$
RRAa/RRAb	$\mathbf{M}_{\updownarrow}\mathbf{R}_{\downarrow}\mathbf{C}_{\updownarrow}$	$\mathbf{P}_{\mathbf{S}\downarrow}$	$\mathbf{P}_{\mathbf{S}\uparrow}$
GIDL	$\mathbf{M}_{\updownarrow}\mathbf{R}_{\updownarrow}$	–	not $\mathbf{P}_{1\downarrow}$, not $\mathbf{P}_{1\uparrow}$
DLBS	$\mathbf{M}_{\downarrow}\mathbf{R}_{\updownarrow}\mathbf{C}_{\updownarrow}$	$\mathbf{P}_{\mathbf{S}\downarrow}$, $\mathbf{P}_{1\uparrow}$	$\mathbf{P}_{\mathbf{W}\uparrow}$, not $\mathbf{P}_{\uparrow}$

some time. Then suppose M contains two landmarks $\varphi_1 = \{(v = 1)\}$ and $\varphi_2 = \{(v = 3)\}$. This is not inconsistent either. It specifies that every plan must achieve both $(v = 1)$ and $(v = 3)$, but not that they must hold in the same state (which is impossible). The new landmark variables we introduce in the transformation can be viewed as flags that are used to remember if a landmark has ever been achieved so far. It is set to true the first time the landmark is achieved, and remains true ever after.

Example 41. Let $V_1 = \{v\}$, $D(v) = \{0, 1\}$ and $A_1 = \{a, b\}$, where $a\colon \overline{v} \Rightarrow v$ and $b\colon v \Rightarrow \overline{v}$. Consider an **DLBS** abstraction with $M = \{\{(v = 1)\}\}$, i.e. there is only one landmark and which has a single atom. Let φ denote the landmark $\{(v = 1)\}$. This is illustrated in Fig. 8 (right), where $g(a)\colon \overline{v} \Rightarrow v, v_\varphi$ and $g(b)\colon v \Rightarrow \overline{v}$. We can move forth and back freely between the states $\{\overline{v}\}$ and $\{v\}$ in $\mathbb{G}_1$, but in $\mathbb{G}_2$ the variable v_φ will always be set to 1 the first time we execute action $g(a)$ and it will thereafter remain at this value, whatever the value of v.

7. Analysis of planning abstractions

We can now analyse the methods in the previous section with respect to their intrinsic transformation properties, i.e. properties that all transformations of a certain type must have. We note that condition (4) on f in all definitions in the previous section enforces that f is a transformation function, so it is not necessary to first assume that τ is a transformation.

The following theorem tells which of the properties in Definition 9 are inherent in the methods in the previous section. The results are also summarised in column 2 of Table 1.

Theorem 42.
(1) $\textbf{\textit{ABS}} \Rightarrow \boldsymbol{R_{\updownarrow}C_{\updownarrow}}$, *(2)* $\textbf{\textit{VP}} \Rightarrow \boldsymbol{M_{\uparrow}R_{\updownarrow}C_{\updownarrow}}$,
(3) $\textbf{\textit{VDA}} \Rightarrow \boldsymbol{M_{\uparrow}R_{\updownarrow}C_{\updownarrow}}$, *(4)* $\textbf{\textit{RRAa}} \Rightarrow \boldsymbol{M_{\updownarrow}R_{\downarrow}C_{\updownarrow}}$,
(5) $\textbf{\textit{RRAb}} \Rightarrow \boldsymbol{M_{\updownarrow}R_{\downarrow}C_{\updownarrow}}$, *(6)* $\textbf{\textit{GIDL}} \Rightarrow \boldsymbol{M_{\updownarrow}R_{\updownarrow}}$,
(7) $\textbf{\textit{DLBS}} \Rightarrow \boldsymbol{M_{\downarrow}R_{\updownarrow}C_{\updownarrow}}$

Proof. We will tacitly make frequent use of Proposition 28. For each of the cases below, assume $\tau = \langle f, R\rangle$ is a transformation from $\mathbb{F}_1 = \langle V_1, D_1, A_1\rangle$ to $\mathbb{F}_2 = \langle V_2, D_2, A_2\rangle$, and that $\mathbb{G}(\mathbb{F}_1) = \langle S_1, E_1\rangle$ and $\mathbb{G}(\mathbb{F}_2) = \langle S_2, E_2\rangle$.

(1) Assume τ is **ABS**. Let $V_C \subseteq V_1$ be the set of critical variables. Note that $S_1 = S_2$.

Suppose $\langle s_1, t_1, a\rangle \in E_1$. Then it must hold that $\mathrm{pre}(a) \subseteq s_1$, so we get $\mathrm{pre}(g(a)) = \mathrm{pre}(a)[V_C] \subseteq \mathrm{pre}(a) \subseteq s_1$ and, hence, $\langle s_1, t_2, g(a)\rangle \in E_2$, where $t_2 = s_1 \ltimes \mathrm{post}(g(a))$. Since $R(a, g(a))$ holds by definition it follows that τ is $\mathbf{R}_{\uparrow}$. Then also suppose that $R(a, \ell)$ holds for some ℓ. By definition of R, the only possibility is that $\ell = g(a)$, so we already know that $\langle s_1, t_2, g(a)\rangle \in E_2$. We get $t_2 = s_1 \ltimes \mathrm{post}(g(a)) = s_1 \ltimes \mathrm{post}(a) = t_1$, and, thus, that $\langle s_1, t_1, g(a)\rangle \in E_2$. Since $s_1 \in f(s_1)$ and $t_1 \in f(t_1)$ it follows that τ is $\mathbf{C}_{\uparrow}$.

Suppose instead that $\langle s_2, t_2, g(a)\rangle \in E_2$. Then $\mathrm{pre}(g(a)) \subseteq s_2$, so we get $\mathrm{pre}(a)[V_C] = \mathrm{pre}(g(a)) \subseteq s_2$. Hence, there must be some state $s_1 \in S_1$ such that $\mathrm{pre}(a) \subseteq s_1$ and $s_1[V_C] = s_2[V_C]$, i.e. $s_1 \in \overline{f}(s_2)$. Let $t_1 = s_1 \ltimes \mathrm{post}(a)$. Then $\langle s_1, t_1, a\rangle \in E_1$ so τ is $\mathbf{R}_{\downarrow}$ since $R(a, g(a))$ holds by definition. Then also suppose that $R(\ell, g(a))$ holds for some ℓ. The only possibility is that $\ell = a$. We have, $t_1[V_C] = (s_1 \ltimes \mathrm{post}(a))[V_C] = s_1[V_C] \ltimes \mathrm{post}(a)[V_C] = s_2[V_C] \ltimes \mathrm{post}(g(a))[V_C] = (s_2 \ltimes \mathrm{post}(g(a)))[V_C] = t_2[V_C]$, i.e. $t_1 \in \overline{f}(t_2)$. Hence, $\langle s_1, t_1, a\rangle \in E_1$, where $s_1 \in \overline{f}(s_2)$ and $t_1 \in \overline{f}(t_2)$. It follows that τ is $\mathbf{C}_{\downarrow}$. (Note, that this proof still holds if using action-dependent sets of critical atoms by replacing $\mathrm{pre}(a)[V_C]$ with $\mathrm{pre}(a)[V_C(a)]$ everywhere and defining V_C as previously explained.)

(2) Assume τ is **VP**. Let $V_C \subseteq V_1$ be the set of critical variables. It is immediate from the definition τ is $\mathbf{M}_{\uparrow}$.

Suppose $\langle s_1, t_1, a\rangle \in E_1$. Let $s_2 = s_1[V_C] = f(s_1)$. Since $\mathrm{pre}(a) \subseteq s_1$, we get $\mathrm{pre}(g(a)) = \mathrm{pre}(a)[V_C] \subseteq s_1[V_C] = s_2$, so $\langle s_2, t_2, g(a)\rangle \in E_2$, where $t_2 = s_2 \ltimes \mathrm{post}(g(a))$. Since also $R(a, g(a))$ holds by definition it follows that $\mathbf{R}_{\uparrow}$ holds. Then also suppose that $R(a, \ell)$ holds for some ℓ. The only possibility is that $\ell = g(a)$, so we already know that $\langle s_2, t_2, g(a)\rangle \in E_2$. Hence, we get that $t_2 = s_2 \ltimes \mathrm{post}(g(a)) = s_1[V_C] \ltimes \mathrm{post}(a)[V_C] = (s_1 \ltimes \mathrm{post}(a))[V_C] = t_1[V_C] = f(t_1)$. It follows that $\langle f(s_1), f(t_1), g(a)\rangle \in E_2$ and, thus, that τ is $\mathbf{C}_{\uparrow}$.

Suppose instead that $\langle s_2, t_2, g(a)\rangle \in E_2$. Then it holds that $\mathrm{pre}(g(a)) \subseteq s_2$ and $t_2 = s_2 \ltimes \mathrm{post}(g(a))$. Since $\mathrm{pre}(a)[V_C] = \mathrm{pre}(g(a)) \subseteq s_2$ there must be some $s_1 \in S_1$ such that $\mathrm{pre}(a) \subseteq s_1$ and $s_1[V_C] = s_2$, i.e. $s_1 \in \overline{f}(s_2)$. Then $\langle s_1, t_1, a\rangle \in E_1$, where

$t_1 = s_1 \ltimes \text{post}(a)$. Since $R(a, g(a))$ holds by definition, it follows that τ is $\mathbf{R}_\downarrow$. Then also suppose that $R(\ell, g(a))$ holds for some ℓ. The only possibility is that $\ell = a$, so we know that $\langle s_1, t_1, a\rangle \in E_1$. Furthermore, $t_1[V_C] = (s_1 \ltimes \text{post}(a))[V_C] = s_1[V_C] \ltimes \text{post}(a)[V_C] = s_2 \ltimes \text{post}(g(a)) = t_2$, i.e. $t_1 \in \overline{f}(t_2)$. It follows that τ is $\mathbf{C}_\downarrow$.

(3) Assume τ is **VDA**. Then τ is $\mathbf{M}_\uparrow$ since $f(s) = h(s)$ and $h(s) \in S_2$ for $s \in S_1$. Before continuing, we first note that for all $s, t \in S_1$ it holds that if $s \subseteq t$, then $h(s) \subseteq h(t)$, and that $h(s \ltimes t) = h(s) \ltimes h(t)$.

Suppose $\langle s_1, t_1, a\rangle \in E_1$. Then $\text{pre}(a) \subseteq s_1$ must hold, so we get $\text{pre}(g(a)) = h(\text{pre}(a)) \subseteq h(s_1)$. Hence, $\langle h(s_1), t_2, g(a)\rangle \in E_2$, where $t_2 = h(s_1) \ltimes \text{post}(g(a))$. Since also $R(a, g(a))$ holds by definition it follows that τ is $\mathbf{R}_\uparrow$. Then also suppose $R(a, \ell)$ holds for some ℓ. The only possibility is $\ell = g(a)$. We have that $h(s_1) = f(s_1)$, by definition, and $t_2 = h(s_1) \ltimes \text{post}(g(a)) = h(s_1) \ltimes h(\text{post}(a)) = h(s_1 \ltimes \text{post}(a)) = h(t_1) = f(t_1)$. It follows that $\langle f(s_1), f(t_1), g(a)\rangle \in E_2$ and, thus, that τ is $\mathbf{C}_\uparrow$.

Suppose instead that $\langle s_2, t_2, g(a)\rangle \in E_2$. Then $\text{pre}(g(a)) \subseteq s_2$ and $t_2 = s_2 \ltimes \text{post}(g(a))$. Since $\text{pre}(g(a)) = h(\text{pre}(a))$, there must exist some $s_1 \in S_1$ such that $\text{pre}(a) \subseteq s_1$ and $s_2 = h(s_1) = f(s_1)$, i.e. $s_1 \in \overline{f}(s_2)$. Hence, $\langle s_1, t_1, a\rangle \in E_2$, where $t_1 = s_1 \ltimes \text{post}(a)$. Since $R(a, g(a))$ holds by definition it follows that τ is $\mathbf{R}_\downarrow$. Then also suppose $R(\ell, g(a))$ holds for some ℓ. The only possibility is $\ell = a$. We get $f(t_1) = h(t_1) = h(s_1 \ltimes \text{post}(a)) = h(s_1) \ltimes h(\text{post}(a)) = s_2 \ltimes \text{post}(g(a)) = t_2$, i.e. $t_1 \in \overline{f}(t_2)$. It follows that τ is $\mathbf{C}_\downarrow$.

(4–5) Assume τ is **RRA** (the proofs hold both for **RRAa** and **RRAb**). Then τ is $\mathbf{M}_\updownarrow$ since f is the identity function.

Suppose $\langle s, t, a\rangle \in E_2$. Then $a \in A_2$ and, thus, also $a \in A_1$ since $A_2 \subseteq A_1$. Hence, $\langle s, t, a\rangle \in E_1$ since $S_1 = S_2$. It follows that τ is $\mathbf{R}_\downarrow$ since $R(a, a)$ holds.

Suppose $\langle s, t, a\rangle \in E_1$ and $R(a, \ell)$ holds. Then $\ell = a$. Since $f(s) = s$ and $f(t) = t$ we get that $\langle f(s), f(t), a\rangle \in E_2$. Hence, τ is $\mathbf{C}_\uparrow$. The $\mathbf{C}_\downarrow$ case is analogous.

(6) Assume τ is **GIDL**. Then τ is $\mathbf{M}_\updownarrow$ since f is the identity function.

Suppose $\langle s_1, t_1, a\rangle \in E_1$. Then $\text{pre}(a) \subseteq s_1$. Hence, $\text{pre}(g(a)) = \text{pre}(a)^{=1} \subseteq \text{pre}(a) \subseteq s_1$, so $\langle s_1, s_1 \ltimes \text{post}(g(a)), g(a)\rangle \in E_2$. Since $R(a, g(a))$ holds by definition we get that τ is $\mathbf{R}_\uparrow$.

Suppose $\langle s_2, t_2, g(a)\rangle \in E_2$. There must exist some $s_1 \in S_1$ such that $\text{pre}(a) \subseteq s_1$. Then $\langle s_1, s_1 \ltimes \text{post}(a), a\rangle \in E_1$. Since $R(a, g(a))$ holds by definition we get that τ is $\mathbf{R}_\downarrow$.

(7) Assume τ is **DLBS**. Then it is immediate from the definition that τ is $\mathbf{M}_\downarrow$.

Suppose $\langle s_1, t_1, a\rangle \in E_1$. Then $\text{pre}(a) \subseteq s_1$. We have $\text{pre}(g(a)) = \text{pre}(a)$ and $s_1 \in S_2$, since $S_1 \subseteq S_2$, so $\langle s_1, t_2, g(a)\rangle \in E_2$, where $t_2 = s_1 \ltimes \text{post}(g(a))$. Since also $R(a, g(a))$ holds by definition it follows that τ is $\mathbf{R}_\uparrow$. Then also suppose that $R(a, \ell)$ holds for some ℓ. Then $\ell = g(a)$, so we know that $\langle s_1, t_2, g(a)\rangle \in E_2$. Obviously, $s_1 \in f(s_1)$. Furthermore, $t_2 = s_1 \ltimes \text{post}(g(a)) = s_1 \ltimes (\text{post}(a) \cup \text{post}_M(a))$. Since it holds that $\text{post}(a) \in \mathcal{C}(V_1 \cdot D_1)$ and that $\text{post}_M(a) \in \mathcal{C}(V_M \cdot D_M)$, we get $t_2 = (s_1 \ltimes \text{post}(a)) \cup \text{post}_M(a) = t_1 \cup \text{post}_M(a)$. Hence, $t_2 \in f(t_1)$ and it follows that τ is $\mathbf{C}_\uparrow$.

Suppose instead that $\langle s_2, t_2, g(a)\rangle \in E_2$. It then holds $\text{pre}(g(a)) \subseteq s_2$, but $\text{pre}(g(a)) = \text{pre}(a)$ so $\text{pre}(a) \subseteq s_2[V_1]$. Let $s_1 = s_2[V_1]$, i.e. $s_1 \in \overline{f}(s_2)$. Then $\langle s_1, t_1, a\rangle \in E_1$, where $t_1 = s_1 \ltimes \text{post}(a)$. Since $R(a, g(a))$ holds by definition it follows that τ is $\mathbf{R}_\downarrow$. Then also suppose $R(\ell, g(a))$ holds for some ℓ. The only possibility is that $\ell = a$. We already know that $\langle s_1, t_1, a\rangle \in E_1$. We get $t_1 = s_1 \ltimes \text{post}(a) = s_2[V_1] \ltimes (\text{post}(a) \cup \text{post}_M(a))[V_1] = s_2[V_1] \ltimes \text{post}(g(a))[V_1] = (s_2 \ltimes \text{post}(g(a)))[V_1] = t_2[V_1]$. Hence, $t_1 \in \overline{f}(t_2)$. It follows that τ is $\mathbf{C}_\downarrow$. □

We then show that the remaining properties from Definition 9 do not generally hold.

Theorem 43.

(1) $\textbf{\textit{ABS}} \not\Rightarrow \boldsymbol{M}_\uparrow$, *(2)* $\textbf{\textit{ABS}} \not\Rightarrow \boldsymbol{M}_\downarrow$, *(3)* $\textbf{\textit{VP}} \not\Rightarrow \boldsymbol{M}_\downarrow$,
(4) $\textbf{\textit{VDA}} \not\Rightarrow \boldsymbol{M}_\downarrow$, *(5)* $\textbf{\textit{RRAa}} \not\Rightarrow \boldsymbol{R}_\uparrow$, *(6)* $\textbf{\textit{RRAb}} \not\Rightarrow \boldsymbol{R}_\uparrow$,
(7) $\textbf{\textit{GIDL}} \not\Rightarrow \boldsymbol{C}_\uparrow$, *(8)* $\textbf{\textit{GIDL}} \not\Rightarrow \boldsymbol{C}_\downarrow$, *(9)* $\textbf{\textit{DLBS}} \not\Rightarrow \boldsymbol{M}_\uparrow$

Proof. (1–6,9) Examples 31, 33, 35, 37 and 41 constitute counterexamples.

(7–8) Consider the transformation τ in Example 39. We have $R(a, g(a))$ and $\langle\{u, \overline{v}\}, \{\overline{u}, v\}, a\rangle \in E_1$, but there is no ℓ such that $\langle\{u, \overline{v}\}, \{\overline{u}, v\}, \ell\rangle \in E_2$ Hence, τ is not $\mathbf{C}_\uparrow$. We also have $\langle\{u, \overline{v}\}, \{u, v\}, g(a)\rangle \in E_2$ but there is no ℓ such that $\langle\{u, \overline{v}\}, \{u, v\}, \ell\rangle \in E_1$. Hence, τ is not $\mathbf{C}_\downarrow$ either. □

The following refinement properties can be immediately deduced from the preceding theorems. They are summarised in column 3 of Table 1.

Theorem 44.

(1) $\textbf{\textit{ABS}} \Rightarrow \boldsymbol{P}_{1\updownarrow}$, *(2)* $\textbf{\textit{VP}} \Rightarrow \boldsymbol{P}_{1\downarrow}\boldsymbol{P}_{S\uparrow}$, *(3)* $\textbf{\textit{VDA}} \Rightarrow \boldsymbol{P}_{1\downarrow}\boldsymbol{P}_{S\uparrow}$,
(4) $\textbf{\textit{RRAa}} \Rightarrow \boldsymbol{P}_{S\downarrow}$, *(5)* $\textbf{\textit{RRAb}} \Rightarrow \boldsymbol{P}_{S\downarrow}$, *(6)* $\textbf{\textit{DLBS}} \Rightarrow \boldsymbol{P}_{S\downarrow}\boldsymbol{P}_{1\uparrow}$.

Proof. Combine Theorem 42 with Theorem 22 and Corollary 23. □

We now explicitly derive the remaining refinement properties of the methods, to achieve tight separations. These are summarised in column 4 of Table 1.

Theorem 45.
(1) $\mathbf{ABS} \Rightarrow \mathbf{P}_{\uparrow}$, (2) $\mathbf{ABS} \not\Rightarrow \mathbf{P}_{\mathbf{S}\uparrow}$, (3a) $\mathbf{ABS} \not\Rightarrow \mathbf{P}_{\mathbf{L}\downarrow}$,
(3b) $\mathbf{ABS} \not\Rightarrow \mathbf{P}_{2\downarrow}$, (4a) $\mathbf{VP} \not\Rightarrow \mathbf{P}_{\mathbf{L}\downarrow}$, (4b) $\mathbf{VP} \not\Rightarrow \mathbf{P}_{2\downarrow}$,
(5a) $\mathbf{VDA} \not\Rightarrow \mathbf{P}_{\mathbf{L}\downarrow}$, (5b) $\mathbf{VDA} \not\Rightarrow \mathbf{P}_{2\downarrow}$, (6) $\mathbf{RRAa} \Rightarrow \mathbf{P}_{\mathbf{S}\uparrow}$,
(7) $\mathbf{RRAb} \Rightarrow \mathbf{P}_{\mathbf{S}\uparrow}$, (8) $\mathbf{GIDL} \not\Rightarrow \mathbf{P}_{1\downarrow}$, (9) $\mathbf{GIDL} \not\Rightarrow \mathbf{P}_{1\uparrow}$,
(10) $\mathbf{DLBS} \Rightarrow \mathbf{P}_{\mathbf{W}\uparrow}$, (11) $\mathbf{DLBS} \not\Rightarrow \mathbf{P}_{\uparrow}$.

Sketch. (1) Suppose $t \in f(\mathcal{R}_1(s))$. Then there is a path $\sigma = s_0, a_1, \ldots, a_m, s_m$ such that $s_0 = s$ and $t \in f(s_m)$. Proof by induction over the length of σ that $t \in f(\mathcal{R}_1(s_0))$ implies $t \in \mathcal{R}_2(f(s_0))$.

Base case: For $m = 0$, we have $t \in f(s_0)$ and, thus, $t \in f(\mathcal{R}_1(s_0))$ and $t \in \mathcal{R}_2(f(s_0))$.

Induction: Suppose the claim holds for paths of length k for some $k \geq 0$. Suppose there is a path $\sigma = s_0, a_1, s_1, \ldots, a_k, s_k, a_{k+1}, s_{k+1}$ in $\mathbb{G}_1$ such that $t \in f(s_{k+1})$, i.e. $t[V_C] = s_{k+1}[V_C]$. There must exist some state $t' \in S_2$ such that $t' \ltimes g(a_{k+1}) = t$, that is, t' satisfies both
(a) $t'[V_C] \ltimes \mathrm{post}(g(a_{k+1}))[V_C] = t[V_C]$ and
(b) $t'[V \setminus V_C] \ltimes \mathrm{post}(g(a_{k+1}))[V \setminus V_C] = t[V \setminus V_C]$.
We can choose t' such that $t'[V_C] = s_k[V_C]$, since $\mathrm{post}(g(a_{k+1})) = \mathrm{post}(a_{k+1}))$ and $t[V_C] = s_{k+1}[V_C]$, i.e. condition (a) is still satisfied by this choice. With this choice of t' we also have $\mathrm{pre}(g(a_{k+1})) \subseteq t'$, since $\mathrm{pre}(g(a_{k+1})) = \mathrm{pre}(a_{k+1})[V_C]$ and $\mathrm{pre}(a_{k+1}) \subseteq s_k$. It follows that $g(a_{k+1})$ is from t' to t, i.e. $t \in \mathcal{R}_2(t')$. We further have $t' \in f(\mathcal{R}_1(s_0))$ since $t' \in f(s_k)$ and it follows from the induction hypothesis that also $t' \in \mathcal{R}_2(f(s_0))$ since s_k is reachable from s_0 in k steps. Transitivity of reachability yields that $t \in \mathcal{R}_2(f(s_0))$. This ends the induction, and it follows that $t \in \mathcal{R}_2(f(s))$, so τ is $\mathbf{P}_{\uparrow}$.

(2) Consider the **ABS** transformation in Example 31. There is a path $\{u, \overline{v}\}, a, \{u, v\}$. We also have $\{u, \overline{v}\} \in f(\{u, \overline{v}\})$ and $\{\overline{u}, v\} \in f(\{u, v\})$, but $\{\overline{u}, v\} \notin \mathcal{R}_2(\{u, \overline{v}\})$ so τ is not $\mathbf{P}_{\mathbf{S}\uparrow}$.

(3–4) Let $V_1 = \{u, v\}$, where $D(u) = \{0, 1\}$ and $D(v) = \{0, 1, 2\}$. Let $a\colon (u = 0), (v = 0) \Rightarrow (v = 1)$ and $b\colon (u = 1), (v = 1) \Rightarrow (v = 2)$ be the only actions. Let $V_C = \{v\}$. Then $g(a)\colon (v = 0) \Rightarrow (v = 1)$ and $g(b)\colon (v = 1) \Rightarrow (v = 2)$ for both **ABS** and **VP**. For **ABS**, $\{(u = 0), (v = 0)\}, g(a), \{(u = 0), (v = 1)\}, g(b), \{(u = 0), (v = 2)\}$ is a path in $\mathbb{G}_2$, but there is no path from any state in $f(\{(u = 0), (v = 0)\})$ to any state in $f(\{(u = 0), (v = 2)\})$ in $\mathbb{G}_1$. For **VP**, $\{(v = 0)\}, g(a), \{(v = 1)\}, g(b), \{(v = 2)\}$ is a path in $\mathbb{G}_2$, but there is no path from any state in $f(\{(v = 0)\})$ to any state in $f(\{(v = 2)\})$ in $\mathbb{G}_1$. Hence, neither method is $\mathbf{P}_{\mathbf{L}\downarrow}$ or $\mathbf{P}_{2\downarrow}$.

(5) Let $V_1 = \{v\}$, where $D_1(v) = \{0, 1, 2, 3\}$. Let $a\colon (v = 0) \Rightarrow (v = 1)$ and $b\colon (v = 2) \Rightarrow (v = 3)$ be the only actions. Define h_v as $h_v(0) = 0$, $h_v(1) = h_v(2) = 1$ and $h_v(3) = 2$. Then $g(a)\colon (v = 0) \Rightarrow (v = 1)$ and $g(b)\colon (v = 1) \Rightarrow (v = 2)$, so $\{(v = 0)\}, g(a), \{(v = 1)\}, g(b), \{(v = 2)\}$ is a path in $\mathbb{G}_2$, but there is no path from $\{(v = 0)\}$ to $\{(v = 3)\}$ in $\mathbb{G}_1$. Hence, **VDA** is neither $\mathbf{P}_{\mathbf{L}\downarrow}$ nor $\mathbf{P}_{2\downarrow}$.

(6–7) For both **RRAa** and **RRAb**, if $\langle s, t, a\rangle \in E_1$, then $t \in \mathcal{R}_2(s)$. Hence, **RRAa** and **RRAb** are $\mathbf{P}_{1\uparrow}$ and, thus, $\mathbf{P}_{\mathbf{S}\uparrow}$ by Theorem 21.

(8–9) Consider Example 39. There is a path $\{u, \overline{v}\}, a, \{\overline{u}, v\}$ in $\mathbb{G}_1$, but $\{\overline{u}, v\} \notin \mathcal{R}_2(\{u, \overline{v}\})$. Hence, **GIDL** is not $\mathbf{P}_{1\uparrow}$. If we remove action b, then there is a path $\{u, \overline{v}\}, g(a), \{u, v\}$ in $\mathbb{G}_2$, but $\{u, v\} \notin \mathcal{R}_1(\{u, \overline{v}\})$. Hence, **GIDL** is not $\mathbf{P}_{1\downarrow}$ either.

(10) Let $s_0, a_1, s_1, a_2, \ldots, a_m, s_m$ be a path in $\mathbb{G}_1$. Let $t_0 = s_0$. Then there is a path $t_0, g(a_1), t_1, g(a_2), \ldots, g(a_m), t_m$ in $\mathbb{G}_2$ such that $t_i \in f(s_i)$ for all i. Hence, **DLBS** is $\mathbf{P}_{\mathbf{W}\uparrow}$.

(11) In Example 41 we have $\{v, \overline{v_\varphi}\} \in f(\mathcal{R}_1(\{\overline{v}\}))$, but $\{v, \overline{v_\varphi}\} \notin \mathcal{R}_2(f(\{\overline{v}\}))$. Hence, **DLBS** is not $\mathbf{P}_{\uparrow}$. □

Without transformation functions, the only reasonable modelling of method **ABS** would be to define $f(s) = s$, which is the traditional way. This works, but results in the properties $\mathbf{M}_{\updownarrow}\mathbf{R}_{\updownarrow}\mathbf{C}_{\uparrow}$ instead. If using different sets of critical variables for the actions, then f collapses to an ordinary $\mathbf{M}_{\updownarrow}$ function whenever there are two actions a and a' such that $V_C(a)$ and $V_c(a')$ are disjoint.

Consulting Theorem 13, we immediately see that **VP** and **VDA** are strong homomorphisms, while **RRAa** and **RRAb** are both retractions. Neither of the other three methods can be immediately classified as any of the four 'usual' methods in Definition 12, although in the case of **DLBS** it is evident that the reverse transformation is a strong homomorphism. The remaining two methods, **ABS** and **GIDL**, remain unclassified with respect to Definition 12, yet they can be easily classified and compared to the other methods in our framework. This is particularly interesting since **ABS** was the first abstraction method used in planning and **GIDL** is one of the most influential methods in domain-independent heuristics for planning today. It is already known that **VP** and **VDA** are essentially equivalent in the sense that a **VP** instance can be transformed into a **VDA** instance. However, our results demonstrate a strong similarity by analysing their transformation properties only, not using any method-specific constructions. We finally conclude that **RRA** is the only one of the methods that also satisfies the **DPP** property (since **DPP** is equivalent to $\mathbf{P}_{\updownarrow}$).

8. Metric properties

In this section we will augment our framework with metric properties for admissibility, in addition to the previous qualitative ones for refinement, and investigate how these properties relate to each other. The results we will prove in this section are summarized in Fig. 9, where the arrows denote the $\Rightarrow$ and $\not\Rightarrow$ relationships between the properties. (Note that the figure contains also the new properties that are yet to be defined and that we have applied in Theorem 21.)

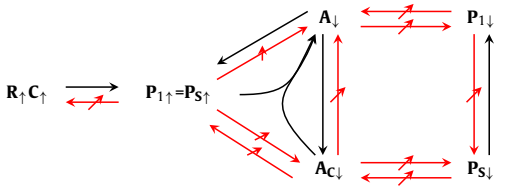


Fig. 9. Relationships between additional properties of $\mathbf{M}_\uparrow$ transformations. Arrows denote the $\Rightarrow$ and $\not\Rightarrow$ relationships between the properties.

8.1. Metrics and heuristics

Heuristic search attempts to find an optimal solution faster than blind search by using a heuristic function that approximates the true cost to guide the search. It is desirable that this function is admissible, i.e. that it never overestimates the true cost, and consistent, i.e. that it satisfies the triangle inequality. For instance, the A* search algorithm [44] is near-optimal under these conditions in the sense of expanding as few nodes as possible for finding an optimal solution [24,56]. This is important since A* and derivatives of it are commonly used in search and planning today.

More precisely, let $\mathbb{R}_0$ be the non-negative reals[7] and define $\mathbb{R}_0^\infty = \mathbb{R}_0 \cup \{\infty\}$. Extend $=$, $<$ and $+$ to $\mathbb{R}_0^\infty$ in the obvious way. Let $\mathbb{G} = \langle S, E\rangle$ be an STG. Generally, a *cost function* for $\mathbb{G}$ is a function $c : S \times S \to \mathbb{R}_0^\infty$, with the restriction that for all $s, t \in S$, $c(s,t) = \infty$ if and only if $t \notin \mathcal{R}(s)$. A *heuristic function* h for a cost function c is itself a cost function. The heuristic h is *admissible* for c if $0 \leq h(s,t) \leq c(s,t)$, for all $s, t \in S$ and h is *consistent* for c if $h(s,t) \leq c(s,u) + h(u,t)$ for all $s, t, u \in S$. It is also common to instead define the heuristic function with respect to a particular set G of goal states as a function h^G of one state, i.e. $h^G(s) = \min_{t \in G} h(s,t)$. Our variant is more general and better suited for our purpose of comparing refinement with heuristics, and it is a reasonable assumption for domain-independent heuristics since h^G can be "simulated" by adding zero-weight arcs from all states in G to a single dedicated goal state [91].

Gaschnig [36] suggested to view both the ground and abstract search spaces as graphs and to define the abstraction such that the length of paths in the abstract graph is an admissible heuristic for the length or cost of the corresponding paths in the ground graph. That is, abstraction is viewed as a graph transformation. This is a very common way to view abstraction heuristics today [31,59]. It is even common to preprocess the abstract graph and store the path lengths in a pattern database [23], or to compute a heuristic function from several such databases [45]. We will consider this method of using an abstract graph to define the heuristic function for a ground graph as follows. Let $\mathbb{G} = \langle S, E\rangle$ be an STG. First define a *weight* function $w : S \times S \to \mathbb{R}_0^\infty$, which must respect that $w(s,s) = 0$ for all $s \in S$ and that $w(s,t) = \infty$ if and only if there is no arc from s to t in E. The exact definition of w does not matter; it could be an arbitrary assignment, it could be based on a weight function on the labels such that $w(s,t)$ is the minimum weight of all labels on the arcs from s to t etc. We extend the weight function to paths such that $w(s_0, \ldots, s_k) = \sum_{i=1}^{k} w(s_{i-1}, s_i)$. Although it is common that the arc weights are given by the labels (i.e. the actions), it is usually only the weights that are considered when computing heuristics, so in that sense, the actual labels are irrelevant. Following common practice in this context, we only consider cost functions that are implicitly defined by the weight function w such that $c(s,t)$ is the minimum of $w(\sigma)$ over all paths σ from s to t, i.e. $c(s,t)$ is the cost of the cheapest path from s to t. Note that $c(s,t) = \infty$ if and only if there is no path from s to t. We also define the specific weight function d, the *distance function*, which is defined such that $d(s,t) = 1$ if there is an arc from s to t in E and otherwise $d(s,t) = \infty$, i.e. d represents the *unit cost assumption*. When considering two STGs $\mathbb{G}_1$ and $\mathbb{G}_2$ simultaneously we index their corresponding c, d and w functions analogously, for instance, w_1 is the weight function for $\mathbb{G}_1$.

We extend transformations with metric information, and we write $\tau = \langle f, R, w_1, w_2\rangle$ when $\langle f, R\rangle$ is a transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$ and w_1 and w_2 are the weight functions for $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. We could alternatively extend the STGs with weight functions. There is no difference in principle between the choices, but our approach allows for a simpler analysis. The cost functions c_1 and c_2 are implicitly defined by w_1 and w_2. For instance, using the path length in the abstract graph as a heuristic estimate for the path length in the ground graph corresponds to an $\langle f, R, d_1, d_2\rangle$ transformation, while an $\langle f, R, w_1, d_2\rangle$ transformation estimates path costs in the ground graph with path lengths in the abstract graph. In order to accommodate cost functions in our framework we will introduce some new metric properties in this section. We only consider such properties for $\mathbf{M}_\uparrow$ transformation functions, since abstraction heuristics usually assume that f is an ordinary function and there is no consensus on how to define heuristics for set-valued abstraction functions. We similarly only define downwards metric properties since heuristics are typically only used in that direction. Corresponding upwards properties could be defined symmetrically, if needed.

It is common to consider admissible heuristics that are also consistent, although exceptions are possible [34]. In our case, the heuristic function c_2 is based on minimum lengths or costs of paths in an abstract graph, so it always satisfies the triangle inequality, i.e. $c_2(s,t) \leq c_2(s,u) + c_2(u,t)$ for all states $s, t, u \in S_2$. Suppose τ is admissible, i.e. $c_2(f(s), f(t)) \leq c_1(s,t)$ for all $s, t \in S_1$. Then, for all $s, t, u \in S_1$ we get that

[7] We use reals for generality of the results. In practice we are, of course, restricted to integers and rationals, and the choice of the numeric domain influences the computational complexity of finding cost-optimal paths [1].

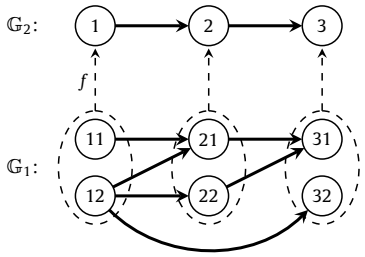


Fig. 10. A transformation that is $\mathbf{P}_{\mathbf{S}\uparrow}$ but not $\mathbf{A}_{\downarrow}$. Unit label ℓ assumed.

$$c_2(f(s), f(t)) \leq c_2(f(s), f(u)) + c_2(f(u), f(t)) \leq c_1(s,u) + c_2(f(u), f(t)),$$

that is, c_2 must also be a consistent heuristic for c_1. Hence, there is no need to consider consistency explicitly.

Definition 46. An $\mathbf{M}_{\uparrow}$ transformation $\tau = \langle f, R, w_1, w_2 \rangle$ can have the following *metric properties*:

$\mathbf{A}_{\downarrow}$: $c_2(f(s), f(t)) \leq c_1(s,t)$ for all $s, t \in S_1$.
$\mathbf{A}_{\mathbf{C}\downarrow}$: $c_2(f(s), f(t)) \leq c_1(s,t)$ or $c_2(f(s), f(t)) = \infty$ for all $s, t \in S_1$.

Property $\mathbf{A}_{\downarrow}$ corresponds to admissibility and $\mathbf{A}_{\mathbf{C}\downarrow}$ is a conditional variant of $\mathbf{A}_{\downarrow}$ that is incomplete, admissibility is only required to hold in the cases where there actually is a path in $\mathbb{G}_2$. The following proposition is immediate.

Proposition 47. *(1)* $\boldsymbol{M_{\uparrow}A_{\downarrow}} \Rightarrow \boldsymbol{A_{\mathbf{C}\downarrow}}$ *(2)* $\boldsymbol{M_{\uparrow}A_{\mathbf{C}\downarrow}} \not\Rightarrow \boldsymbol{A_{\downarrow}}$

The results in the remainder of this section will fill in the rest of the arrows in Fig. 9.

8.2. Metric properties and upwards refinement

The following theorem formalizes that admissibility implies completeness, but the opposite is false; not even the strongest form of completeness, $\mathbf{P}_{\mathbf{S}\uparrow}$, guarantees admissibility. Furthermore, conditional admissibility is not strong enough to imply even the weakest form of completeness.

Theorem 48.
(1) $\boldsymbol{M_{\uparrow}A_{\downarrow}} \Rightarrow \boldsymbol{P_{S\uparrow}}$ *(2)* $\boldsymbol{M_{\uparrow}P_{S\uparrow}} \not\Rightarrow \boldsymbol{A_{\mathbf{C}\downarrow}}$ *(3)* $\boldsymbol{M_{\uparrow}A_{\mathbf{C}\downarrow}} \not\Rightarrow \boldsymbol{P_{1\uparrow}}$

Proof. In each case below, let $\tau = \langle f, R, w_1, w_2 \rangle$ be an $\mathbf{M}_{\uparrow}$ transformation from $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ to $\mathbb{G}_2 = \langle S_2, E_2 \rangle$.

(1) Choose $\mathbb{G}_1$, $\mathbb{G}_2$ and τ arbitrarily. Suppose τ is $\mathbf{A}_{\downarrow}$. Let $e = \langle s, t, \ell \rangle$ be an arbitrary arc in E_1. Then $c_1(s,t) < \infty$, so $c_2(f(s), f(t)) < \infty$ since τ is $\mathbf{A}_{\downarrow}$. Hence, $f(t) \in \mathcal{R}_2(f(s))$. Since e was chosen arbitrarily, it follows that τ is $\mathbf{P}_{1\uparrow}$ and, thus, also $\mathbf{P}_{\mathbf{S}\uparrow}$ by Theorem 21.

(2) Let $S_1 = S_2 = \{1,2,3\}$, $E_2 = \{\langle 1,2,a \rangle, \langle 2,3,a \rangle\}$ and $E_1 = E_2 \cup \{\langle 1,3,a \rangle\}$ Let f be the identity function, $R = \{\langle a,a \rangle\}$ and $\tau = \langle f, R, d_1, d_2 \rangle$. Then, τ is clearly $\mathbf{P}_{\mathbf{S}\uparrow}$, but not $\mathbf{A}_{\mathbf{C}\downarrow}$ since $c_1(1,3) = d_1(1,3) = 1$ but $c_2(f(1), f(3)) = d_2(f(1), f(2)) + d_2(f(2), f(3)) = 2$.

(3) Let $S_1 = S_2 = \{1,2\}$, $E_1 = \{\langle 1,2,a \rangle\}$, $E_2 = \varnothing$, $f(1) = 1$, $f(2) = 2$, $R(a,a)$ and $\tau = \langle f, R, d_1, d_2 \rangle$. Then τ is vacuously $\mathbf{A}_{\mathbf{C}\downarrow}$, but it is not $\mathbf{P}_{1\uparrow}$. □

We obviously also get $\mathbf{M}_{\uparrow}\mathbf{A}_{\downarrow} \Rightarrow \mathbf{P}_{1\uparrow}$, $\mathbf{M}_{\uparrow}\mathbf{P}_{\mathbf{S}\uparrow} \not\Rightarrow \mathbf{A}_{\downarrow}$ and other immediate consequences. Fig. 10 is an example of a $\mathbf{P}_{\mathbf{S}\uparrow}$ transformation (the labelling is omitted for readability since all arcs have label ℓ). However, there is a one-arc path $12, 32$ in $\mathbb{G}_1$ but the shortest path from $f(12)$ to $f(32)$ in $\mathbb{G}_2$ is of length 2. If we apply unit cost to both graphs, then the transformation cannot be $\mathbf{A}_{\downarrow}$.

It is sufficient to combine conditional admissibility with the weakest form of completeness to get full admissibility, as the following theorem demonstrates.

Theorem 49. $\boldsymbol{M_{\uparrow}P_{1\uparrow}A_{\mathbf{C}\downarrow}} \Rightarrow \boldsymbol{A_{\downarrow}}$

Proof. Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ be arbitrary STGs and let $\tau = \langle f, R, w_1, w_2 \rangle$ be an arbitrary $\mathbf{M}_{\uparrow}\mathbf{P}_{1\uparrow}\mathbf{A}_{\mathbf{C}\downarrow}$ transformation. Let s, s' be two arbitrary states in S_1. If $c_1(s,s') = \infty$, then $c_2(f(s), f(s')) \leq c_1(s,s')$ holds trivially, so suppose $c_1(s,s') < \infty$. Then there is a path $s_0, \ldots, s_n$ in $\mathbb{G}_1$, where $s_0 = s$ and $s_n = s'$. Since τ is $\mathbf{M}_{\uparrow}$ and $\mathbf{P}_{1\uparrow}$ there are states $t_0, \ldots, t_n \in S_2$ such that $t_i = f(s_i)$ for all i $(0 \leq i \leq n)$ and $t_i \in \mathcal{R}_2(t_{i-1})$ for all i $(1 \leq i \leq n)$. Hence, $t_n \in \mathcal{R}_2(t_0)$, i.e.

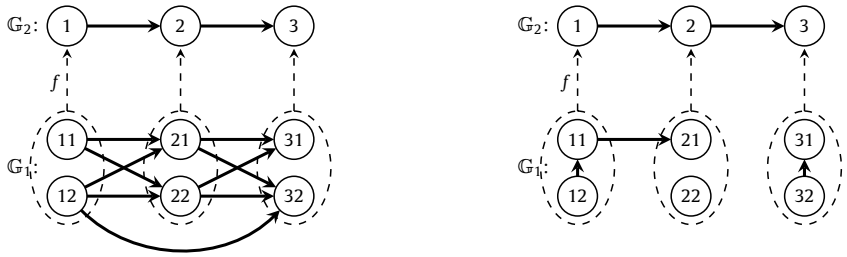


Fig. 11. Transformations that are $\mathbf{P_{S}}_{\downarrow}$ but not $\mathbf{A_{C}}_{\downarrow}$ (left) and $\mathbf{A}_{\downarrow}$ but not $\mathbf{P}_{1\downarrow}$ (right). Unit label ℓ assumed.

$f(s') \in \mathcal{R}_2(f(s))$, so it must hold that $c_2(f(s), f(s')) \leq c_1(s, s')$ since τ is $\mathbf{A_{C}}_{\downarrow}$. It follows that τ is $\mathbf{A}_{\downarrow}$ since s and s' were chosen arbitrarily. □

8.3. Metric properties and downward refinement

Admissibility enforces completeness but not soundness; that is, there can be an abstract path with no corresponding ground path. In fact, admissibility and downward refinement are largely orthogonal and incomparable concepts.

Theorem 50. *(1)* $\boldsymbol{M}_{\uparrow}\boldsymbol{P_{S}}_{\downarrow} \not\Rightarrow \boldsymbol{A_{C}}_{\downarrow}$ *(2)* $\boldsymbol{M}_{\uparrow}\boldsymbol{A}_{\downarrow} \not\Rightarrow \boldsymbol{P}_{1\downarrow}$

Proof. (1) Use the same counterexample as in the proof of Theorem 48(2) and note that τ is also $\mathbf{P_{S}}_{\downarrow}$. However, $c_1(1,3) = 1$ but $c_2(f(1), f(3)) = 2$, so τ is not $\mathbf{A_{C}}_{\downarrow}$.

(2) Let $S_1 = S_2 = \{1, 2\}$, $E_1 = \varnothing$ and $E_2 = \{\langle 1, 2, a\rangle\}$. Also let f be the identity function, $R = \varnothing$ and $\tau = \langle f, R, d_1, d_2\rangle$. Then τ is $\mathbf{A}_{\downarrow}$ but not $\mathbf{P}_{1\downarrow}$ since $E_1 = \varnothing$. □

We can also prove similar results for the hierarchy of refinement properties.

Theorem 51.
(1) $\boldsymbol{M}_{\uparrow}\boldsymbol{P}_{1\downarrow}\boldsymbol{A}_{\downarrow} \not\Rightarrow \boldsymbol{P_{L}}_{\downarrow}$ *(2)* $\boldsymbol{M}_{\uparrow}\boldsymbol{P_{L}}_{\downarrow}\boldsymbol{A}_{\downarrow} \not\Rightarrow \boldsymbol{P_{W}}_{\downarrow}$
(3) $\boldsymbol{M}_{\uparrow}\boldsymbol{P_{W}}_{\downarrow}\boldsymbol{A}_{\downarrow} \not\Rightarrow \boldsymbol{P}_{\downarrow}$ *(4)* $\boldsymbol{M}_{\uparrow}\boldsymbol{P}_{\downarrow}\boldsymbol{A}_{\downarrow} \not\Rightarrow \boldsymbol{P_{S}}_{\downarrow}$

Proof. (1-4) Follows from Theorem 18 since Examples 17(1, 3, 4) are $\mathbf{A}_{\downarrow}$ for $\tau = \langle f, R, d_1, d_2\rangle$ and Example 17(2) is $\mathbf{A}_{\downarrow}$ for $\tau = \langle f, R, w_1, d_2\rangle$, where $w_1 = d_1$ except that $w_1(1,4) = 2$. □

The major reason for these results is that refinement is defined without any metrics; even if we have a guarantee that an abstract plan can be refined into a ground plan, we have no guarantee that there is no shorter ground plan.

To illustrate Theorem 51, consider the transformation in Fig. 11(left), which is obviously $\mathbf{P_{S}}_{\downarrow}$. It does, however, have a path in $\mathbb{G}_1$ of length one from 12 to 32, while the shortest path from $f(12) = \{1\}$ to $f(32) = \{3\}$ is of length 2. It follows that the transformation is not $\mathbf{A}_{\downarrow}$, or even $\mathbf{A_{C}}_{\downarrow}$. That is, it is sound in the strongest sense but d_2 is not even a conditionally admissible heuristic for d_1.

Then consider the transformation in Fig. 11(right). There is a path of length one, i.e. an arc, in $\mathbb{G}_2$ from 2 to 3 but there is no path of whatever length in $\mathbb{G}_1$ from some state in $\overline{f}(2) = \{21, 22\}$ to some state in $\overline{f}(3) = \{31, 32\}$. Hence, the transformation, is not $\mathbf{P}_{1\downarrow}$. It is $\mathbf{A}_{\downarrow}$, though. Note, for instance, that $d_2(2,3) = 1$ because of the arc from 2 to 3 while $d_1(s, s') = \infty$ for all $s \in \overline{f}(2)$ and $s' \in \overline{f}(3)$. That is, d_2 is an admissible heuristic for d_1 but the transformation is not sound even in the trivial sense.

8.4. Relating metric and non-metric properties

Also the properties in Definition 9 have connections with the metric properties. We know that $\mathbf{M}_{\uparrow}\mathbf{R}_{\uparrow}\mathbf{C}_{\uparrow} \Rightarrow \mathbf{P_{S}}_{\uparrow}$ and $\mathbf{P_{S}}_{\uparrow} \Rightarrow \mathbf{P}_{1\uparrow}$, so it follows from Theorem 49 that $\mathbf{M}_{\uparrow}\mathbf{R}_{\uparrow}\mathbf{C}_{\uparrow}\mathbf{A_{C}}_{\downarrow} \Rightarrow \mathbf{A}_{\downarrow}$. However, we can also derive the following more direct relationship between metric and non-metric properties, since $\mathbf{M}_{\uparrow}\mathbf{R}_{\uparrow}\mathbf{C}_{\uparrow}$ transformations are homomorphisms.

Proposition 52. *Let* $\tau = \langle f, R, w_1, w_2\rangle$ *be an* $\boldsymbol{M}_{\uparrow}\boldsymbol{R}_{\uparrow}\boldsymbol{C}_{\uparrow}$ *transformation. Then:*

1. *If* $w_2(f(s), f(t)) \leq w_1(s, t)$ *for all* $s, t \in S_1$ *such that* $\langle s, t, \ell\rangle \in E_1$ *for some* $\ell \in L(E_1)$ *and* $\langle f(s), f(t), \ell'\rangle \in E_2$ *for some* $\ell' \in L(E_2)$*, then* τ *is* $\boldsymbol{A}_{\downarrow}$.
2. *Otherwise,* τ *need not be* $\boldsymbol{A}_{\downarrow}$.

8.5. Examples with metric properties

In this section we give examples of how the extended framework can be used to express new things or express old things in new ways. The examples are deliberately quite different from each other in order to demonstrate the breadth of applicability of an abstract framework of this kind.

8.5.1. Admissibility and homomorphisms

Homomorphisms, i.e. $\mathbf{M}_\uparrow\mathbf{R}_\uparrow\mathbf{C}_\uparrow$ transformations, are known to be very suitable as abstraction functions in heuristic search, cf. the articles by Holte et al. [59], Helmert et al. [48] and Zilles and Holte [92]. One reason for this is that they are admissible. The following result shows that also the opposite holds, admissibility implies that the abstraction function is a homomorphism, if all arcs in both graphs have unit cost. Without unit costs, this relationship breaks down. This is a most relevant observation in the context of using the path length in the abstract graph as the heuristic estimate for the path length or path cost in the ground graph.

Theorem 53. *Let $\tau = \langle f, R, w_1, d_2\rangle$ be an $\boldsymbol{M}_\uparrow\boldsymbol{A}_\downarrow$ transformation. Then:*

1. *If $w_1(s,t) < 2$ or $w_1(s,t) = \infty$ for all $s, t \in S_1$, then f is a homomorphism.*
2. *Otherwise f need not be a homomorphism.*

Proof. (1) Suppose that τ is $\mathbf{M}_\uparrow\mathbf{A}_\downarrow$ and $w_1(s,t) < 2$ or $w_1(s,t) = \infty$ for all $s, t \in S_1$. Then suppose that f is not a homomorphism. There must then be some arc $\langle s, t, \ell\rangle \in E_1$ such that $\langle f(s), f(t), \ell'\rangle \notin E_2$ for any $\ell' \in L(E_2)$. That is, $c_1(s,t) \leq w_1(s,t) < 2$, but every path from $f(s)$ to $f(t)$, must be of length 2 or more. Hence, $c_2(f(s), f(t)) \geq 2$, so $c_2(f(s), f(t)) > c_1(s,t)$ and τ cannot, thus, be $\mathbf{A}_\downarrow$. This contradicts the assumption, so f must be a homomorphism.

(2) Let $S_1 = S_2 = \{1,2,3\}$, $E_1 = \{\langle 1,3,a\rangle\}$, $E_2 = \{\langle 1,2,a\rangle, \langle 2,3,a\rangle\}$. Let $f(s) = s$ for all $s \in S_1$ and let $w_1(1,3) = 2$. Clearly, τ is $\mathbf{A}_\downarrow$ for this example. However, f is not a homomorphism since $\langle 1,3,a\rangle \in E_1$ but $\langle f(1), f(3), a'\rangle \notin E_2$ for any $a' \in L(E_2)$. □

8.5.2. Spurious states

We have earlier noted that the **DPP** criterion [92] is equivalent to $\mathbf{P}_\updownarrow$. Since $\mathbf{A}_\downarrow \Rightarrow \mathbf{P}_{\mathbf{S}\uparrow}$ and $\mathbf{P}_{\mathbf{S}\uparrow} \Rightarrow \mathbf{P}_\uparrow$, we can alternatively define the **DPP** property as $\mathbf{P}_\downarrow\mathbf{A}_\downarrow$, instead of $\mathbf{P}_\updownarrow$. Although $\mathbf{P}_\downarrow\mathbf{A}_\downarrow$ is a stronger criterion than $\mathbf{P}_\updownarrow$ it means that we need not verify independently that $\mathbf{P}_\uparrow$ holds if we already know that the heuristic is admissible, which is often the case.

8.5.3. Globally admissible heuristics

Karpas and Domshlak [62] considered optimal solutions with non-admissible heuristics. One example is so called *globally admissible heuristics*, which need only be admissible for the states along some optimal plan. Let $\mathbb{G} = \langle S, E\rangle$ be an STG, c a cost function for $\mathbb{G}$ and h a heuristic for c. Then, for arbitrary $s, t \in S$, h is *globally admissible* for c from s to t if there is an optimal path $s_0, \ldots, s_n$ such that $s_0 = s$, $s_n = t$ and $h(s_i, t) \leq c(s_i, t)$ for all i, $0 \leq i \leq n$. One may view this concept in the following alternative way.

Theorem 54. *Let $\tau = \langle f, R, w_1, w_2\rangle$ be an $\boldsymbol{M}_\uparrow\boldsymbol{A}_{\mathbf{C}\downarrow}$ transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$. For all states s and t in $\mathbb{G}_1$, if there is an optimal path σ from s to t in $\mathbb{G}_1$ such that $f(\sigma)$ is a path in $\mathbb{G}_2$, then c_2 is a globally admissible heuristic for c_1 from s to t.*

Proof. Since σ is a path from s to t in $\mathbb{G}_1$ and $f(\sigma)$ is a path in $\mathbb{G}_2$, the latter must be a path from $f(s)$ to $f(t)$. Hence, $c_2(f(s), f(t)) < \infty$, so $c_2(f(u), f(t)) < \infty$ for all states u along σ. The theorem then follows by definition of $\mathbf{A}_{\mathbf{C}\downarrow}$. □

This makes use of properties $\mathbf{M}_\uparrow$ and $\mathbf{A}_{\mathbf{C}\downarrow}$ and a type of completeness property that is even weaker than $\mathbf{P}_{1\uparrow}$.

8.5.4. Valtorta's theorem

Valtorta [90] proved that when using embeddings as abstraction functions, it is not possible to explore fewer nodes in total, counting both ground and abstract nodes, when using the A* search algorithm [44] with path length in the abstract graph as heuristic estimate, than if using Dijkstra's algorithm directly in the ground graph. This was later generalised to abstraction functions in general by Holte et al. [61]. This is known as the generalised version of Valtorta's theorem and can be stated as follows using our notation and formalism.

Theorem 55 *(Generalised Valtorta's theorem, [31,61]). Assume $\tau = \langle f, R, d_1, d_2\rangle$ is an $\boldsymbol{M}_\uparrow$ transformation. Let u be any state in $\mathbb{G}_1$ that is necessarily expanded when the instance $\langle s, t\rangle$ is solved by blind search in $\mathbb{G}_1$ and let the heuristic function h be $h(u,t) = d_2(f(u), f(t))$, computed by blind search in $\mathbb{G}_2$. If A* solves this instance, then either u or $f(u)$ will be expanded during the search.*

We follow Holte et al. [61] and view blind search as A* search without a heuristic, i.e. a kind of breadth-first search. Holte et al. noted that this does not rule out that some abstractions might explore fewer nodes in total. (The theorem is somewhat weak, though, since it only tells us that *u or* $f(u)$ is expanded, allowing the possibility that both are expanded.) It is well

known that this requires that there are abstract nodes corresponding to two or more ground nodes; the expansion of an abstract node can then result in a heuristic estimate that prevents A* from exploring the corresponding ground nodes. An alternative characterization of this is that f is not $\mathbf{M}_{\downarrow}$. While this is hardly an interesting new result itself, it demonstrates that the framework defined so far is sufficient to express this important criterion for A* search.

We may also step outside the $\mathbf{M}_{\uparrow}$ assumption. Suppose we have some concept of expanding $f(u)$ for a state u when f is not $\mathbf{M}_{\uparrow}$. We need not have a precise definition of this concept to see that Theorem 55 still holds, and that f must still not be $\mathbf{M}_{\downarrow}$ to have any chance of exploring fewer nodes.

8.5.5. Previous abstraction methods and admissibility

All of the methods **VP**, **VDA**, **RRAa** and **GIDL** are $\mathbf{M}_{\uparrow}$ and they also satisfy that if $a_1, \ldots, a_n$ is a plan for $\mathbb{P}_1$, then $g(a_1), \ldots, g(a_n)$ is a plan for $\mathbb{P}_2$. Hence, these methods are all suitable for defining admissible heuristics, if $w_2(f(s), f(t)) \leq w_1(s,t)$ for all s, t. Also **RRAb** is $\mathbf{M}_{\uparrow}$, but it is not $\mathbf{A}_{\downarrow}$ in general since the costs in the two graphs have no suitable relationship. The usual admissibility concept does not even apply to methods **ABS** and **DLBS**, since they are not $\mathbf{M}_{\uparrow}$.

Since **VP** and **VDA** are $\mathbf{M}_{\uparrow}\mathbf{R}_{\uparrow}\mathbf{C}_{\uparrow}$, we can alternatively apply Theorem 52 to derive that they are $\mathbf{A}_{\downarrow}$ under the stated assumptions about the graph weights. Also, **GIDL** is known to be admissible, which is essential for its success as a heuristic. Yet, this does not follow automatically from any inherent properties in our framework, but must be explicitly derived. This indicates that it differs from methods like **VP** and **VDA** in some fundamental way.

9. Transformation composition and abstraction hierarchies

In this section, we define composition of transformations and define a transitivity concept for transformation properties. This is important in cases where we have to combine transformations, either the same type, as in hierarchical abstraction, or mixed types, as in computing the Merge and Shrink heuristic. We thus prove that essentially all properties, as well as the previous planning abstraction methods, are transitive.

9.1. Composition of transformations

Let S_1, S_2 and S_3 be sets of states and let $f_1 : S_1 \to 2^{S_2}$ and $f_2 : S_2 \to 2^{S_3}$ be functions. Then the *composition* $f_1 \circ f_2$ of f_1 with f_2 is defined in the usual way,[8] i.e. $(f_1 \circ f_2)(s) = f_2(f_1(s))$ for all $s \in S_1$. Note that $f_1(s)$ is a set of states, so $f_2(f_1(s)) = \bigcup_{t \in f_1(s)} f_2(t)$ according to our previous definitions. That is, $(f_1 \circ f_2)(s)$ is a set of states in S_3, not a set of subsets of S_3. This is essential, since $f_1 \circ f_2$ could otherwise never be a transformation function from S_1 to S_3.

The composition $f_1 \circ f_2$ is not necessarily a transformation function, even if both f_1 and f_2 are transformation functions, as the following example shows.

Example 56. Let $S_1 = S_3 = \{0, 1\}$ and $S_2 = \{0, 1, 2\}$. Define two transformation functions $f_1 : S_1 \to 2^{S_2}$ and $f_2 : S_2 \to 2^{S_3}$ such that $f_1(0) = \{0, 1\}$, $f_1(1) = \{2\}$, $f_2(0) = \{0\}$ and $f_2(1) = f_2(2) = \{1\}$. Then $(f_1 \circ f_2)(0) = \{0, 1\}$ and $(f_1 \circ f_2)(1) = \{1\}$, so $f_1 \circ f_2$ is not a transformation function.

Conversely, $f_1 \circ f_2$ can be a transformation function even if neither f_1 nor f_2 is a transformation function.

Example 57. Let $S_1 = S_3 = \{0, 1\}$, $S_2 = \{0, 1, 2\}$ and let the functions $f_1 : S_1 \to 2^{S_2}$ and $f_2 : S_2 \to 2^{S_3}$ be defined such that $f_1(0) = \{0\}$, $f_1(1) = \{1\}$, $f_2(0) = \{0\}$, $f_2(1) = \{1\}$ and $f_2(2) = \{0, 1\}$. Then neither f_1 nor f_2 is a transformation function. However, $(f_1 \circ f_2)(0) = \{0\}$ and $(f_1 \circ f_2)(1) = \{1\}$, so $f_1 \circ f_2$ is a transformation function.

We will frequently say that a composition $f_1 \circ f_2$ is a transformation function without specifying the domain and range, since these are implicitly defined by f_1 and f_2. We will not give a precise characterisation of when a composition is a transformation function or not, but the following is a sufficient condition.

Definition 58. Let S_1, S_2 and S_3 be state spaces, let f_1 be a transformation function from S_1 to S_2 and let f_2 be a transformation function from S_2 to S_3. Then the tuple $\langle f_1, f_2 \rangle$ has the *intermediate subset property* if for all $s \in S_1$ and $u \in (f_1 \circ f_2)(s)$ either (1) $f_1(s) \subseteq \overline{f_2}(u)$ or (2) $\overline{f_2}(u) \subseteq f_1(s)$.

Theorem 59. *Let S_1, S_2 and S_3 be state spaces, let f_1 be a transformation function from S_1 to S_2 and let f_2 be a transformation function from S_2 to S_3. Then $f_1 \circ f_2$ is a transformation function if $\langle f_1, f_2 \rangle$ has the intermediate subset property.*

Proof. Let $f_3 = f_1 \circ f_2$. Suppose $\langle f_1, f_2 \rangle$ has the intermediate subset property. Then suppose that f_3 is not a transformation function. We first note that f_3 must be a total function since f_1 and f_2 are total, so it must be the case that $\mathrm{Rng}(f_3)$ is not a partition of S_3. Obviously, $f_3(S_1)$ covers S_3 and $f_3(s) \neq \varnothing$ for all $s \in S_1$. Hence, there must be $s, s' \in S_1$ such

[8] Note that both definitions $(f_1 \circ f_2)(s) = f_2(f_1(s))$ and $(f_1 \circ f_2)(s) = f_1(f_2(s))$ occur in the literature.

that $f_3(s) \neq f_3(s')$ and $f_3(s) \cap f_3(s') \neq \varnothing$, since $\mathrm{Rng}(f_3)$ is not a partition of S_3. There must then be $u, u' \in S_3$ such that $u \in f_3(s) \cap f_3(s')$ and either $u' \in f_3(s) \setminus f_3(s')$ or $u' \in f_3(s') \setminus f_3(s)$. Without losing generality, assume that $u' \in f_3(s) \setminus f_3(s')$. Let $T = f_1(s)$ and $T' = f_1(s')$. Then $T \neq T'$ since $f_2(T) = f_3(s) \neq f_3(s') = f_2(T')$. It follows that $s \neq s'$ and $T \cap T' = \varnothing$, since f_1 is a transformation function. It also follows that $u \in f_2(T) \cap f_2(T')$ and $u' \in f_2(T) \setminus f_2(T')$. There are, thus, $t, t' \in T$ such that $u \in f_2(t)$ and $u' \in f_2(t')$. There is also some $t'' \in T'$ such that $u \in f_2(t'')$. Suppose that $t = t'$. Then $u' \in f_2(t)$, so we have $\{u, u'\} \subseteq f_2(t)$ and $\{u\} \subseteq f_2(t'')$. However, it must then hold that also $u' \in f_2(t'')$ since f_2 is a transformation function. This contradicts the assumptions and it hence follows that $t \neq t'$. We now have $t, t' \in f_1(s)$, $t'' \notin f_1(s)$, $t, t'' \in \overline{f_2}(u)$ and $t' \notin \overline{f_2}(u)$. It follows that neither $f_1(s) \subseteq \overline{f_2}(u)$ nor $\overline{f_2}(u) \subseteq f_1(s)$ can hold, but this contradicts the assumptions so we conclude that f_3 must be a transformation function. □

The following condition is an immediate consequence.

Corollary 60. *A composition $f_1 \circ f_2$ of two transformation functions f_1 and f_2 is a transformation function if either f_1 is $\boldsymbol{M}_\uparrow$ or f_2 is $\boldsymbol{M}_\downarrow$.*

While Theorem 59 is a sufficient condition, it is not a necessary one, as the following example shows.

Example 61. Let $S_1 = S_3 = \{1, 2\}$ and $S_2 = \{1, 2, 3, 4\}$. Define the transformation functions f_1 from S_1 to S_2 and f_2 from S_2 to S_3 such that $f_1(1) = \{1, 2\}$, $f_1(2) = \{3, 4\}$, $f_2(1) = f_2(3) = \{1\}$ and $f_2(2) = f_2(4) = \{2\}$. Let $f_3 = f_1 \circ f_2$. We get $f_3(1) = f_3(2) = \{1, 2\}$, so f_3 is a transformation function. However, $\langle f_1, f_2 \rangle$ does not have the intermediate subset property.

We define composition of label relations in the usual way. Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$, $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ and $\mathbb{G}_3 = \langle S_3, E_3 \rangle$ be STGs. Let R_1 be a label relation from E_1 to E_2 and let R_2 be a label relation from E_2 to E_3. Then the *composition* $R_1 \circ R_2$ of R_1 with R_2 is $R_1 \circ R_2 = \{\langle \ell_1, \ell_3 \rangle \mid \exists \ell_2 \in L(E_2).R(\ell_1, \ell_2) \text{ and } R(\ell_2, \ell_3)\}$. It is immediate that $R_1 \circ R_2$ is always a label relation. Furthermore, let $\tau_1 = \langle f_1, R_1 \rangle$ be a transformation from S_1 to S_2 and let $\tau_2 = \langle f_2, R_2 \rangle$ be a transformation from S_2 to S_3. Then the *composition* $\tau_1 \circ \tau_2$ of τ_1 with τ_2 is defined as $\tau_1 \circ \tau_2 = \langle f_1 \circ f_2, R_1 \circ R_2 \rangle$. It is immediate that $\tau_1 \circ \tau_2$ is a transformation from $\mathbb{G}_1$ to $\mathbb{G}_3$ if and only if $f_1 \circ f_2$ is a transformation function from S_1 to S_3. We will frequently say that $\tau_1 \circ \tau_2$ is a transformation without explicitly specifying the STGs, meaning that $f_1 \circ f_2$ is a transformation function.

9.2. Transitivity of transformation properties

One of the properties that make homomorphic abstractions attractive is transitivity, i.e. the composition of two homomorphisms is itself a homomorphism [48]. This is particularly interesting when forming hierarchies of abstractions [65,3,61]; if we have a hierarchy $\tau_1, \tau_2, \ldots, \tau_n$ of transformations that all have a property **X**, then it is desirable that also the composite transformation $\tau_1 \circ \tau_2 \circ \ldots \circ \tau_n$ has property **X**. Since we do not restrict ourselves to homomorphisms, or any other particular type of abstraction, we instead show that almost all transformation properties in this article are transitive in the following sense.

Definition 62. A transformation property **X** is *transitive* if for all STGs $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_3$, and for all transformations τ_1 from $\mathbb{G}_1$ to $\mathbb{G}_2$ and τ_2 from $\mathbb{G}_2$ to $\mathbb{G}_3$ the following holds:

if both τ_1 and τ_2 are **X** and $\tau_1 \circ \tau_2$ is a transformation, then $\tau_1 \circ \tau_2$ is **X**.

All properties in Definition 9 are transitive.

Theorem 63. *Properties $\boldsymbol{M}_\uparrow$, $\boldsymbol{M}_\downarrow$, $\boldsymbol{R}_\uparrow$, $\boldsymbol{R}_\downarrow$, $\boldsymbol{C}_\uparrow$ and $\boldsymbol{C}_\downarrow$ are transitive.*

Proof. We show only the upwards cases. The downwards cases are analogous. Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$, $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ and $\mathbb{G}_3 = \langle S_3, E_3 \rangle$ be arbitrary STGs. Let $\tau_1 = \langle f_1, R_1 \rangle$ from $\mathbb{G}_1$ to $\mathbb{G}_2$ and $\tau_2 = \langle f_2, R_2 \rangle$ from $\mathbb{G}_2$ to $\mathbb{G}_3$ be two arbitrary transformations such that $\tau_1 \circ \tau_2$ is a transformation, i.e. $f_1 \circ f_2$ is a transformation function. Define $f_3 = f_1 \circ f_2$, $R_3 = R_1 \circ R_2$ and $\tau_3 = \tau_1 \circ \tau_2 = \langle f_3, R_3 \rangle$.

$\mathbf{M}_\uparrow$: Suppose both τ_1 and τ_2 are $\mathbf{M}_\uparrow$. Choose $s \in S_1$ arbitrarily. By definition, $f_3(s) = f_2(f_1(s)) = \bigcup_{t \in f_1(s)} f_2(t)$. However, f_1 is $\mathbf{M}_\uparrow$ so there is some $t \in S_2$ such that $f_1(s) = t$. Hence, $f_3(s) = f_2(t)$ and it follows that $|f_3(s)| = 1$ since f_2 is $\mathbf{M}_\uparrow$. Hence, f_3 is $\mathbf{M}_\uparrow$ since s was chosen arbitrarily.

$\mathbf{R}_\uparrow$: Immediate from the definitions of composition.

$\mathbf{C}_\uparrow$: Suppose both τ_1 and τ_2 are $\mathbf{C}_\uparrow$. Let $\langle s_1, t_1, \ell_1 \rangle$ be an arbitrary edge in E_1 and let $\ell_3 \in L(\mathbb{G}_3)$ be an arbitrary label such that $R_3(\ell_1, \ell_3)$. By definition of R_3 there must be some label $\ell_2 \in L(\mathbb{G}_2)$ such that $R_1(\ell_1, \ell_2)$ and $R_2(\ell_2, \ell_3)$. Since τ_1 is $\mathbf{C}_\uparrow$ there must, thus, be some edge $\langle s_2, t_2, \ell_2 \rangle \in E_2$ such that $s_2 \in f_1(s_1)$ and $t_2 \in f_1(t_1)$. Furthermore, since also τ_2 is $\mathbf{C}_\uparrow$ there must be some edge $\langle s_3, t_3, \ell_3 \rangle \in E_3$ such that $s_3 \in f_2(s_2)$ and $t_3 \in f_2(t_2)$. It then follows from the definition of f_3 that $s_3 \in f_3(s_1)$ and $t_3 \in f_3(t_1)$. Hence, τ_3 is also $\mathbf{C}_\uparrow$ since s_1, s_2, ℓ_1 and ℓ_3 were chosen arbitrarily. □

Also all important refinement properties are transitive.

Theorem 64. *Properties* $\mathbf{P_{L\downarrow}}$, $\mathbf{P_{L\uparrow}}$, $\mathbf{P_{W\downarrow}}$, $\mathbf{P_{W\uparrow}}$, $\mathbf{P_{\downarrow}}$, $\mathbf{P_{\uparrow}}$, $\mathbf{P_{S\downarrow}}$ *and* $\mathbf{P_{S\uparrow}}$ *are transitive.*

Proof. We show only the downwards cases. The upwards cases are analogous. Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$, $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ and $\mathbb{G}_3 = \langle S_3, E_3 \rangle$ be arbitrary STGs. Let $\tau_1 = \langle f_1, R_1 \rangle$ from $\mathbb{G}_1$ to $\mathbb{G}_2$ and $\tau_2 = \langle f_2, R_2 \rangle$ from $\mathbb{G}_2$ to $\mathbb{G}_3$ be two arbitrary transformations such that $\tau_1 \circ \tau_2$ is a transformation, i.e. $f_1 \circ f_2$ is a transformation function. Define $f_3 = f_1 \circ f_2$, $R_3 = R_1 \circ R_2$ and $\tau_3 = \tau_1 \circ \tau_2 = \langle f_3, R_3 \rangle$.

$\mathbf{P_{L\downarrow}}$: Suppose both τ_1 and τ_2 are $\mathbf{P_{L\downarrow}}$. Let $\sigma = u_0, \ldots, u_m$ in $\mathbb{G}_3$ be an arbitrary path in $\mathbb{G}_3$. Since τ_2 is $\mathbf{P_{L\downarrow}}$ there are states $t_0 \in \overline{f_2}(u_0)$ and $t_n \in \overline{f_2}(u_n)$ such that $t_n \in \mathcal{R}_2(t_0)$. Hence, there must be a path $t_0, \ldots, t_n$ in $\mathbb{G}_2$. Since also τ_1 is $\mathbf{P_{L\downarrow}}$ there are states $s_0 \in \overline{f_1}(t_0)$ and $s_\ell \in \overline{f_1}(t_n)$ such that $s_\ell \in \mathcal{R}_1(s_0)$. It follows that $s_0 \in \overline{f_3}(u_0)$ and $s_\ell \in \overline{f_3}(u_m)$. Hence, τ_3 is $\mathbf{P_{L\downarrow}}$ since σ was chosen arbitrarily.

$\mathbf{P_{W\downarrow}}$: Suppose both τ_1 and τ_2 are $\mathbf{P_{W\downarrow}}$. Let $\sigma = u_0, \ldots, u_m \in S_3$ be an arbitrary path in $\mathbb{G}_3$. Since τ_2 is $\mathbf{P_{W\downarrow}}$ there are states $t_0, \ldots, t_m \in S_2$ such that $t_i \in \overline{f_2}(u_i)$ for all i $(0 \leq i \leq m)$ and $t_i \in \mathcal{R}_2(t_{i-1})$ for all i $(1 \leq i \leq m)$. That is, there is a path $v_0, \ldots, v_n \in S_2$, where $m \leq n$, and integers $i_0, \ldots, i_m$ such that $0 = i_0 < i_1 < i_2 < \ldots < i_m = n$ and $t_j = v_{i_j}$ for all j $(0 \leq j \leq m)$. Since also τ_1 is $\mathbf{P_{W\downarrow}}$ there are states $s_0, \ldots, s_n \in S_1$ such that $s_i \in \overline{f_1}(v_i)$ for all i $(0 \leq i \leq n)$ and $s_i \in \mathcal{R}(s_{i-1})$ for all i $(1 \leq i \leq n)$. For all j $(0 \leq j \leq m)$ it thus holds that $v_{i_j} \in \overline{f_2}(u_j)$ and that $s_{i_j} \in \overline{f_1}(v_{i_j})$, i.e. $s_{i_j} \in \overline{f_3}(u_j)$. It also holds for all j $(1 \leq j \leq m)$ that $s_{i_j} \in \mathcal{R}_1(s_{i_{j-1}})$ since reachability is transitive. It follows that τ_3 is $\mathbf{P_{W\downarrow}}$, since σ was chosen arbitrarily.

$\mathbf{P_{\downarrow}}$: Suppose both τ_1 and τ_2 are $\mathbf{P_{\downarrow}}$. Let $s_1 \in S_1$ and $t_3 \in S_3$ be arbitrary states such that $t_3 \in \mathcal{R}_3(f_3(s_1))$, i.e. there is some state $s_3 \in f_3(s_1)$ such that $t_3 \in \mathcal{R}_3(s_3)$. By definition of f_3 there must also be some state $s_2 \in S_2$ such that $s_2 \in f_1(s_1)$ and $s_3 \in f_2(s_2)$. Then $t_3 \in \mathcal{R}_3(f_2(s_2))$, so $t_3 \in f(\mathcal{R}_2(s_2))$ since τ_2 is $\mathbf{P_{\downarrow}}$. That is, there is some $t_2 \in \mathcal{R}_2(s_2)$ such that $t_3 \in f(t_2)$. It follows that $t_2 \in \mathcal{R}_2(f_1(s_1))$, so $t_2 \in f_1(\mathcal{R}_1(s_1))$ since τ_1 is $\mathbf{P_{\downarrow}}$. Hence, $t_3 \in f_2(f_1(t_1)) = f_3(t_1)$ so $t_3 \in f_3(\mathcal{R}_1(s_1))$. It follows that $\mathcal{R}_3(f_3(s_1)) \subseteq f_3(\mathcal{R}_1(s_1))$ and, thus, that τ_3 is $\mathbf{P_{\downarrow}}$ since s_1 was chosen arbitrarily.

$\mathbf{P_{S\downarrow}}$: Analogous to the $\mathbf{P_{W\downarrow}}$ case. □

The properties $\mathbf{P}_{k\uparrow}$ and $\mathbf{P}_{k\downarrow}$, on the other hand, are not transitive for any fixed k, which is less important since they are primarily used as tools for defining other properties.

We define composition of metric transformations in the obvious way. Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_3$ be STGs, let $\tau_1 = \langle f_1, R_1, w_1, w_2 \rangle$ be a metric transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$ and let $\tau_2 = \langle f_2, R_2, w_2, w_3 \rangle$ be a metric transformation from $\mathbb{G}_2$ to $\mathbb{G}_3$. Then $\tau_1 \circ \tau_2 = \langle f_1 \circ f_2, R_1 \circ R_2, w_1, w_3 \rangle$, i.e. the weights in the graphs remain unchanged.

Theorem 65. *Properties* $\mathbf{A_{\downarrow}}$ *and* $\mathbf{A_{C\downarrow}}$ *are transitive.*

Proof. We prove only the $\mathbf{A_{\downarrow}}$ case, since the $\mathbf{A_{C\downarrow}}$ case is analogous. Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$, $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ and $\mathbb{G}_3 = \langle S_3, E_3 \rangle$ be arbitrary STGs. Let $\tau_1 = \langle f_1, R_1, w_1, w_2 \rangle$ from $\mathbb{G}_1$ to $\mathbb{G}_2$ and let $\tau_2 = \langle f_2, R_2, w_2, w_3 \rangle$ from $\mathbb{G}_2$ to $\mathbb{G}_3$ be two arbitrary $\mathbf{M_{\uparrow}}$ metric transformations. Suppose τ_1 and τ_2 are $\mathbf{A_{\downarrow}}$ and that $\tau_1 \circ \tau_2$ is a transformation, i.e. $f_1 \circ f_2$ is a transformation function. Define $f_3 = f_1 \circ f_2$, $R_3 = R_1 \circ R_2$ and $\tau_3 = \tau_1 \circ \tau_2 = \langle f_1 \circ f_2, R_1 \circ R_2, w_1, w_3 \rangle$. Let s_1, t_1 be arbitrary states in S_1. Then there are states $s_2, t_2 \in S_2$ such that $s_2 = f_1(s_1)$ and $t_2 = f_1(t_1)$ and states $s_3, t_3 \in S_3$ such that $s_3 = f_2(s_2)$ and $t_3 = f_2(t_2)$. It follows from Theorem 63 that f_3 is $\mathbf{M_{\uparrow}}$ since both f_1 and f_2 are $\mathbf{M_{\uparrow}}$, so $s_3 = f_3(s_1)$ and $t_3 = f_3(t_1)$. Furthermore, $c_2(f_1(s_1), f_1(t_1)) = c_2(s_2, t_2) \leq c_1(s_1, t_1)$ since τ_1 is $\mathbf{A_{\downarrow}}$ and $c_3(f_2(s_2), f_2(t_2)) = c_3(s_3, t_3) \leq c_2(s_2, t_2)$ since τ_2 is $\mathbf{A_{\downarrow}}$. It follows that $c_3(f_3(s_1), f_3(t_1)) = c_3(s_3, t_3) \leq c_1(s_1, t_1)$ and, hence, that τ_3 is $\mathbf{A_{\downarrow}}$ since s_1 and t_1 were chosen arbitrarily. □

9.3. Transitivity of planning abstractions

In this section, we will show that (almost) all of the abstraction methods in Section 6 are transitive. It becomes essential here to cast these in the relational form that we have used. Although some of the methods have been used in a compositional manner in the literature, to create abstraction hierarchies, there is no obvious or standard way to define the extra information for compositions. Consider the composition of two **ABS** transformations τ_1, with critical variables V_{C1} and function g_1, and τ_2, with critical variables V_{C2} and function g_2. In practice, the sets V_{C1} and V_{C2} will usually be chosen by some principle. However, if we construct the composite transformation $\tau_3 = \tau_1 \circ \tau_2$, then there is no single obvious or standard way to construct the set of critical variables for this composition. Hence, we will instead show that there exists a systematic way to choose a set V_{C3} and function g_3 such that τ_3 preserves the **ABS** property.

Theorem 66. *Methods* ***ABS***, ***VP***, ***VDA***, ***RRAa***, ***RRAb*** *and* ***GIDL*** *are transitive.*

Proof. Let $\mathbb{F}_1 = \langle V_1, D_1, A_1 \rangle$, $\mathbb{F}_2 = \langle V_2, D_2, A_2 \rangle$ and $\mathbb{F}_3 = \langle V_3, D_3, A_3 \rangle$ be arbitrary SAS$^+$ frames with corresponding STGs $\mathbb{G}_1 = \langle S_1, E_1 \rangle$, $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ and $\mathbb{G}_3 = \langle S_3, E_3 \rangle$. Let $\tau_1 = \langle f_1, R_1 \rangle$ from $\mathbb{F}_1$ to $\mathbb{F}_2$ and $\tau_2 = \langle f_2, R_2 \rangle$ from $\mathbb{F}_2$ to $\mathbb{F}_3$ be two arbitrary transformations. Recall that τ_1 is then implicitly a transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$ and τ_2 a transformation from

$\mathbb{G}_2$ to $\mathbb{G}_3$. Suppose that $\tau_1 \circ \tau_2$ is a transformation, i.e. $f_1 \circ f_2$ is a transformation function. Define $f_3 = f_1 \circ f_2$, $R_3 = R_1 \circ R_2$ and $\tau_3 = \tau_1 \circ \tau_2 = \langle f_3, R_3 \rangle$.

ABS: Suppose both τ_1 and τ_2 are **ABS**. Then there are two corresponding sets $V_{C1} \subseteq V_1$ and $V_{C2} \subseteq V_2$ of critical variables and two corresponding bijections $g_1 : A_1 \to A_2$ and $g_2 : A_2 \to A_3$, such that Definition 30 is satisfied for both τ_1 and τ_2. We must prove that there is a set $V_{C3} \subseteq V_1$ of critical variables and a bijection $g_3 : A_1 \to A_3$ such that τ_3 is an **ABS** transformation from $\mathbb{F}_1$ to $\mathbb{F}_3$. Note that τ_3 does not specify V_{C3} and g_3, there must only exist a choice of them that satisfies Definition 30 for τ_3. Choose[9] $V_{C3} = V_{C1} \cap V_{C2}$ and $g_3 = g_1 \circ g_2$, which is a bijection. We now prove that conditions (1)–(5) in Definition 30 are satisfied.

(1) We have $V_2 = V_1$ and $D_2 = D_1$, since τ_1 is **ABS**, and $V_3 = V_2$ and $D_3 = D_2$ since τ_2 is **ABS**, so it follows that $V_3 = V_1$ and $D_3 = D_1$.

(2) $\{g_3(a) \mid a \in A_1\} = \{g_2(g_1(a)) \mid a \in A_1\} = \{g_2(a) \mid a \in A_2\} = A_3$.

(3) For all $a \in A_1$, we get $\mathrm{pre}(g_3(a)) = \mathrm{pre}(a)[V_{C3}] = \mathrm{pre}(a)[V_{C1} \cap V_{C2}] = \mathrm{pre}(a)[V_{C1}][V_{C2}] = \mathrm{pre}(g_1(a))[V_{C2}] = \mathrm{pre}(g_2(g_1(a))$ and that $\mathrm{post}(g_3(a)) = \mathrm{post}(a) = \mathrm{post}(g_1(a)) = \mathrm{post}(g_2(g_1(a))$.

(4) We need to prove that $f_3(s) = \{u \in S_3 \mid s[V_{C3}] = u[V_{C3}]\}$ for all $s \in S_1$. We have that $u \in f_3(s) = f_2(f_1(s))$ if and only if there is some $t \in S_2$ such that $t \in f_1(s)$ and $u \in f_2(t)$. By definition, this is equivalent to $t[V_{C1}] = s[V_{C1}]$ and $u[V_{C2}] = t[V_{C_2}]$, which holds if and only if $s[V_{C1} \cap V_{C2}] = u[V_{C1} \cap V_{C2}]$. Since $V_{C3} = V_{C1} \cap V_{C2}$ it follows that $u \in f_3(s)$ if and only if $s[V_{C3}] = u[V_{C3}]$.

(5) By definition, $R_3 = R_1 \circ R_2$. We have $R_1(a_1, a_2)$ if only if $a_2 = g_1(a_1)$ and $R_2(a_2, a_3)$ if and only if $a_3 = g_2(a_2)$. Hence, we have $R_3(a_1, a_3)$ if and only if $a_3 = g_2(g_1(a)) = g_3(a)$.

VP: Suppose both τ_1 and τ_2 are **VP**. Then there are two corresponding sets $V_{C1} \subseteq V_1$ and $V_{C2} \subseteq V_2$ of critical variables and two corresponding bijections $g_1 : A_1 \to A_2$ and $g_2 : A_2 \to A_3$, such that Definition 32 is satisfied for both τ_1 and τ_2. By definition, $V_2 = V_{C1}$ and $V_3 = V_{C2}$, so $V_3 \subseteq V_2 \subseteq V_1$ and $V_{C2} \subseteq V_{C1}$. We must prove that there is a set $V_{C3} \subseteq V_1$ of critical variables and a bijection $g_3 : A_1 \to A_3$ such that τ_3 is an **VP** transformation from $\mathbb{F}_1$ to $\mathbb{F}_3$. Choose $V_{C3} = V_{C2}$ and $g_3 = g_1 \circ g_2$, which is a bijection. We now prove that the conditions (1)–(5) in the definition are satisfied. We will tacitly use that $s[V_{C1}][V_{C2}] = s[V_{C2}]$ for all $s \in S_1$ since $V_{C2} \subseteq V_{C1}$. The proofs for (2) and (5) are identical to **ABS**.

(1) We know that $V_3 = V_{C2}$, since τ_2 is **VP**. Hence, it holds that $V_3 = V_{C3}$ since $V_{C3} = V_{C2}$.

(3) For all actions $a \in A_1$, we have $\mathrm{pre}(g_3(a)) = \mathrm{pre}(a)[V_{C3}] = \mathrm{pre}(a)[V_{C2}] = \mathrm{pre}(a)[V_{C1} \cap V_{C2}] = \mathrm{pre}(a)[V_{C1}][V_{C2}] = \mathrm{pre}(g_1(a))[V_{C2}] = \mathrm{pre}(g_2(g_1(a))$ and analogously for $\mathrm{post}(g_3(a))$.

(4) $f_3(s) = f_2(f_1(s)) = f_1(s)[V_{C2}] = s[V_{C1}][V_{C2}] = s[V_{C2}] = s[V_{C3}]$.

VDA: Suppose τ_1 and τ_2 are **VDA**. Then there are two families $H_1 = \{h_{1,v} : D_1(v) \to D_2(v) \mid v \in V_1\}$ and $H_2 = \{h_{2,v} : D_2(v) \to D_3(v) \mid v \in V_2\}$ of domain mappings and two corresponding bijections $g_1 : A_1 \to A_2$ and $g_2 : A_2 \to A_3$ such that Definition 34 is satisfied for both τ_1 and τ_2. We must prove that there is a family $H_3 = \{h_{3,v} : D_1(v) \to D_3(v) \mid v \in V_1\}$ of domain mappings and a bijection $g_3 : A_1 \to A_3$ such that τ_3 is a **VDA** transformation from $\mathbb{F}_1$ to $\mathbb{F}_3$. Choose H_3 such that $h_{3,v} = h_{1,v} \circ h_{2,v}$ for all $v \in V$, and $g_3(a) = g_2(g_1(a))$ for all $a \in A_1$, i.e. g_3 is a bijection. We must prove that conditions (1–5) in Definition 34 hold. The proofs for (2) and (5) are identical to **ABS**.

(1) We have $V_2 = V_1$ and $V_3 = V_2$, so $V_3 = V_1$. Furthermore, $h_{3,v}(D_1(v)) = h_{2,v}(h_{1,v}(D_1(v))) = h_{2,v}(D_2(v)) = D_3(v)$.

(3) We get $\mathrm{pre}(g_3(a)) = \mathrm{pre}(g_2(g_1(a)) = h_2(\mathrm{pre}(g_1(a))) = h_2(h_1(\mathrm{pre}(a))) = h_3(\mathrm{pre}(a))$ and analogously for $\mathrm{post}(g_3(a))$.

(4) $f_3(s) = f_2(f_1(s)) = h_2(h_1(s)) = h_3(s)$.

RRAa: Suppose both τ_1 and τ_2 are **RRAa**. We must prove that τ_3 is an **RRAa** transformation from $\mathbb{F}_1$ to $\mathbb{F}_3$, that is, we must prove that conditions (1–2), (3a) and (4–5) in Definition 36 hold. The proof for (1) is identical to **ABS**.

(2) Both τ_1 and τ_2 are **RRAa** so $A_2 \subseteq A_1$ and $A_3 \subseteq A_2$. It follows that $A_3 \subseteq A_1$.

(3a) We know that $\{\langle s, t\rangle \mid \langle s, t, \ell\rangle \in E_1\} = \{\langle s, t\rangle \mid \langle s, t, \ell\rangle \in E_2\}$ since τ_1 is **RRAa**, and that $\{\langle s, t\rangle \mid \langle s, t, \ell\rangle \in E_2\} = \{\langle s, t\rangle \mid \langle s, t, \ell\rangle \in E_3\}$, since τ_2 is **RRAa**. It follows that $\{\langle s, t\rangle \mid \langle s, t, \ell\rangle \in E_1\} = \{\langle s, t\rangle \mid \langle s, t, \ell\rangle \in E_3\}$.

(4) Both f_1 and f_2 are the identity function since both τ_1 and τ_2 are **RRAa**. It follows that f_3 is the identity function since $f_3 = f_1 \circ f_2$.

(5) Both R_1 and R_2 are the identity relation since both τ_1 and τ_2 are **RRAa**. It follows that R_3 is the identity relation since $R_3 = R_1 \circ R_2$.

RRAb: Identical to **RRAa**, except that we prove condition (3b) instead of (3a).

(3b) We know that $\{\langle s, t\rangle \mid t \in \mathcal{R}_1(s)\} = \{\langle s, t\rangle \mid t \in \mathcal{R}_2(s)\}$, since τ_1 is **RRAb**, and that $\{\langle s, t\rangle \mid t \in \mathcal{R}_2(s)\} = \{\langle s, t\rangle \mid t \in \mathcal{R}_3(s)\}$, since τ_2 is **RRAb**. It follows that $\{\langle s, t\rangle \mid t \in \mathcal{R}_1(s)\} = \{\langle s, t\rangle \mid t \in \mathcal{R}_3(s)\}$.

GIDL: Suppose both τ_1 and τ_2 are **GIDL** transformations. Then there are two corresponding bijections $g_1 : A_1 \to A_2$ and $g_2 : A_2 \to A_3$ such that Definition 38 is satisfied for both τ_1 and τ_2. We must prove that there is a bijection $g_3 : A_1 \to A_3$ such that τ_3 is a **GIDL** transformation from $\mathbb{F}_1$ to $\mathbb{F}_3$. Choose $g_3 = g_1 \circ g_2$, which is a bijection. We must prove that conditions (1)–(5) in Definition 38 hold. The proofs for (1), (2) and (5) are identical to **ABS** and the proof for (4) is identical to **RRA**.

(3) For all $a \in A_1$, it holds that $\mathrm{post}(g_3(a)) = \mathrm{post}(a)^{=1} = (\mathrm{post}(a)^{=1})^{=1} = \mathrm{post}(g_1(a))^{=1} = \mathrm{post}(g_2(g_1(a)))$ and analogously for $\mathrm{pre}(g_3(a))$. □

[9] In the case of action-dependent critical sets, we do as previously described. Each $a \in A_1$ has a set $V_{C1}(a)$ and each $a \in A_2$ has a set $V_{C2}(a)$, so we define $V_{C3}(a) = V_{C1}(a) \cap V_{C2}(g_1(a))$ for each $a \in A_1$. We can then define $V_{C1} = \bigcap_{a \in A_1} V_{C1}(a)$ and $V_{C2} = \bigcap_{a \in A_2} V_{C2}(a)$, as previously described. Defining $V_{C3} = V_{C1} \cap V_{C2}$ still works then.

In the proof for **ABS**, it may happen that the set V_{C3} is empty, in the case where V_{C1} and V_{C2} are disjoint. This is fine, but results in a very coarse abstraction where $f_3(s) = S_3$ for all $s \in S_1$ and all preconditions will be empty, which happens also when doing the transformations τ_1 and τ_2 in sequence. For **GIDL** transformations, we note that it is redundant to apply the transformation twice. Applying it once results in a frame where all action postconditions are positive, i.e. of the form $(v = 1)$. Applying it once more has no effect since all postconditions already are positive.

Property **DLBS** is the only one of the methods in Section 6 that is not transitive. The following example illustrates why.

Example 67. Let M_1 be the landmarks for τ_1 and M_2 the landmarks for τ_2. Let v be a variable in V_1 and let $\varphi_1 = \{(v = 0)\} \in M_1$ be a landmark on V_1. Then V_2 contains both v and a variable v_{φ_1} for the landmark (the actual name of the variable is not important) and every action in A_2 that contains $(v = 0)$ in its postcondition will also contain $(V_{\varphi_1} = 1)$. Since M_2 is a set of landmarks on V_2, it is allowed to contain the landmark $\varphi_2 = \{(v_{\varphi_1} = 1)\}$, which is a landmark for the landmark φ_1. Then V_3 contains v, v_{φ_1} and v_{φ_2} and every action in A_3 with $(v = 0)$ in its postcondition will also contain $(v_{\varphi_1} = 1)$ and $(v_{\varphi_2} = 1)$ in its postcondition. Now consider the composite transformation $\tau_3 = \tau_1 \circ \tau_2$, which is a direct transformation from $\mathbb{F}_1$ to $\mathbb{F}_3$. Let M_3 be the landmark set for τ_3. By definition, M_3 can only contain landmarks on V_1 so it can contain the landmark $\varphi_1 = \{(v = 0)\}$ but not the landmark $\varphi_2 = \{(v_{\varphi_1} = 1)\}$. Hence, τ_3 and $\tau_1 \circ \tau_2$ do not result in the same frame and, thus, cannot both be transformations from $\mathbb{G}_1$ to $\mathbb{G}_3$.

In practice there will usually be some reason for composing two transformations, for instance, to let $\mathbb{F}_2$ and $\mathbb{F}_3$ represent different abstraction levels for $\mathbb{F}_1$. It is then reasonable that M_2 does not contain any landmarks on the variables in M_1, since this is redundant; we note in the example above that both the landmarks $\varphi_1 \in M_1$ and $\varphi_2 \in M_2$ must always be set simultaneously, so the first one is sufficient. We will now show that also **DLBS** is transitive under the additional assumption that M_2 contains no landmarks on the explicit landmarks in M_1.

Theorem 68. *Let $\mathbb{F}_1 = \langle V_1, D_1, A_1 \rangle$, $\mathbb{F}_2 = \langle V_2, D_2, A_2 \rangle$ and $\mathbb{F}_3 = \langle V_3, D_3, A_3 \rangle$ be arbitrary SAS$^+$ frames with the corresponding STGs $\mathbb{G}_1 = \langle S_1, E_1 \rangle$, $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ and $\mathbb{G}_3 = \langle S_3, E_3 \rangle$. Let $\tau_1 = \langle f_1, R_1 \rangle$ from $\mathbb{G}_1$ to $\mathbb{G}_2$ and $\tau_2 = \langle f_2, R_2 \rangle$ from $\mathbb{G}_2$ to $\mathbb{G}_3$ be two arbitrary transformations such that $\tau_1 \circ \tau_2$ is a transformation. Let $f_3 = f_1 \circ f_2$, $R_3 = R_1 \circ R_2$ and $\tau_3 = \tau_1 \circ \tau_2 = \langle f_3, R_3 \rangle$. If there are two sets $M_1, M_2 \subseteq 2^{V_1 \cdot D_1}$ of disjunctive landmarks and two bijections $g_1 : A_1 \to A_2$ and $g_2 : A_2 \to A_3$ such that both τ_1 and τ_2 are **DLBS** transformations, then there is a set $M_3 \subseteq 2^{V_1 \cdot D_1}$ of disjunctive landmarks and a bijection $g_3 : A_1 \to A_3$ such that τ_3 is **DLBS**.*

Proof. We must prove that there is a set M_3 of landmarks and a bijection $g_3 : A_1 \to A_3$ such that τ_3 is an **DLBS** transformation from $\mathbb{F}_1$ to $\mathbb{F}_3$. Choose $M_3 = M_1 \cup M_2$ and $g_3 = g_1 \circ g_2$, which is a bijection. We must prove that conditions (1–5) in Definition 40 hold. The proofs for (2) and (5) are like those for **ABS**.

(1) We know that $V_2 = V_1 \cup V_{M1}$, $D_2 = D_1 \cup D_M$, since τ_1 is **DLBS**, and that $V_3 = V_2 \cup V_{M2}$, $D_3 = D_2 \cup D_M$, since also τ_2 is **DLBS**. It follows that $V_1 \cup V_{M3} = V_1 \cup V_{M1} \cup V_{M2} = V_2 \cup V_{M2} = V_3$.

(3) We have $\text{post}_{M1}(a) = \{(v_\varphi = 1) \mid \text{post}(a) \cap \varphi \neq \varnothing \text{ and } \varphi \in M_1\}$ and $\text{post}(g_1(a)) = \text{post}(a) \cup \text{post}_{M1}(a)$ for all $a \in A_1$ since τ_1 is **DLBS**. Since also τ_2 is **DLBS** we further get that $\text{post}_{M2}(g_1(a)) = \{(v_\varphi = 1) \mid \text{post}(g_1(a)) \cap \varphi \neq \varnothing \text{ and } \varphi \in M_2\} = \{(v_\varphi = 1) \mid (\text{post}(a) \cup \text{post}_{M1}(a)) \cap \varphi \neq \varnothing \text{ and } \varphi \in M_2\}$. However, $\text{post}_{M1}(a) \cap \varphi = \varnothing$ for all $\varphi \in M_2$ since $\text{post}_{M1}(a) \subseteq V_{M1} \cdot D_M$ and $\varphi \subseteq V_1 \cdot D_1$ for all $\varphi \in M_2$. Hence $\text{post}_{M2}(g_1(a)) = \{(v_\varphi = 1) \mid \text{post}(a) \cap \varphi \neq \varnothing \text{ and } \varphi \in M_2\}$. We get $\text{post}(g_2(g_1(a))) = \text{post}(g_1(a)) \cup \text{post}_{M2}(g_1(a)) = \text{post}(a) \cup \text{post}_{M1}(a) \cup \text{post}_{M2}(g_1(a))$. However, we have $\text{post}_{M1}(a) = \{(v_\varphi = 1) \mid \text{post}(a) \cap \varphi \neq \varnothing \text{ and } \varphi \in M_1\}$ and $\text{post}_{M2}(g_1(a)) = \{(v_\varphi = 1) \mid \text{post}(a) \cap \varphi \neq \varnothing \text{ and } \varphi \in M_2\}$, so we get that $\text{post}_{M1}(a) \cup \text{post}_{M2}(g_1(a)) = \{(v_\varphi = 1) \mid \text{post}(a) \cap \varphi \neq \varnothing \text{ and } \varphi \in M_1 \cup M_2\}$. Since $M_1 \cup M_2 = M_3$, we get $\text{post}(g_2(g_1(a))) = \{(v_\varphi = 1) \mid \text{post}(a) \cap \varphi \neq \varnothing \text{ and } \varphi \in M_3\}$.

(4) We have that $f_3(s) = f_2(f_1(s)) = \bigcup_{t \in f_1(s)} f_2(t)$ and we also have that $f_1(s) = \{s \cup m \mid m \in \mathcal{T}(V_{M1} \cdot D_M)\}$ and $f_2(t) = \{t \cup m \mid m \in \mathcal{T}(V_{M2} \cdot D_M)\}$. It follows that

$$\begin{aligned} f_3(s) &= \{s \cup m_1 \cup m_2 \mid m_1 \in \mathcal{T}(V_{M1} \cdot D_M) \text{ and } m_2 \in \mathcal{T}(V_{M2} \cdot D_M)\} \\ &= \{s \cup m \mid m \in \mathcal{T}((V_{M1} \cup V_{M2}) \cdot D_M)\} \\ &= \{s \cup m \mid m \in \mathcal{T}(V_{M3} \cdot D_M)\}. \quad \square \end{aligned}$$

9.4. An example: merge and shrink abstraction

As an example of composing mixed types of transformations, we model the *Merge-and-shrink* (**M&S**) abstraction method [48,49]. This method computes a smaller abstract version of the STG for a SAS$^+$ frame, with the purpose of using this abstraction to compute a heuristic function. We will briefly define this method within our transformation framework and sketch an analysis of it. We first need the concept of synchronous products, which is identical to the usual one in the literature although we use the set-based definition of SAS$^+$ instead of the usual vector-based one. Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_2, E_2 \rangle$ be two STGs such that S_1 and S_2 are disjoint. Then the *synchronous product* of $\mathbb{G}_1$ and $\mathbb{G}_2$ is $\mathbb{G}_1 \otimes \mathbb{G}_2 = \langle S_3, E_3 \rangle$, where $S_3 = \{s \cup t \mid s \in S_1 \text{ and } t \in S_2\}$ and

$$E_3 = \{\langle s \cup s', t \cup t', \ell\rangle \mid \langle s, t, \ell\rangle \in E_1 \text{ and } \langle s', t', \ell\rangle \in E_2\}.$$

The **M&S** method can now be described as follows within our framework. Let $\mathbb{F} = \langle V, D, A\rangle$ be a SAS$^+$ frame with STG $\mathbb{G}(\mathbb{F})$. Assume $V = \{v_1, \ldots, v_n\}$. For each $v_i \in V$, compute the SAS$^+$ frame $\mathbb{F}_i$ as the **VP** abstraction of $\mathbb{F}$ with $V_C = \{v_i\}$ and let $\mathbb{G}_{v_i} = \mathbb{G}(\mathbb{F}_i)$. Each $\mathbb{G}_{v_i}$ is of polynomial size in the size of $\mathbb{F}$ and can be computed in polynomial time. By definition of **VP**, all actions in $\mathbb{F}$ appear as labels in each $\mathbb{G}_{v_i}$, i.e. $L(\mathbb{G}_{v_i}) = L(\mathbb{G}(\mathbb{F}))$. Hence, we get that $\mathbb{G}(\mathbb{F}) = \mathbb{G}_{v_1} \otimes, \ldots, \otimes \mathbb{G}_{v_n}$, so we could, in principle, construct $\mathbb{G}(\mathbb{F})$ in this way. However, this is no gain since the size of $\mathbb{G}(\mathbb{F})$ is typically exponential in the size of $\mathbb{F}$ and it could as well be computed directly from $\mathbb{F}$. The **M&S** method instead constructs a smaller abstraction of $\mathbb{G}(\mathbb{F})$ as follows. Let $\Gamma = \{\mathbb{G}_{v_1}, \ldots, \mathbb{G}_{v_n}\}$. The abstract STG for $\mathbb{F}$ is then constructed by repeatedly applying the following three operations.[10]

Merge Choose two $\mathbb{G}, \mathbb{G}' \in \Gamma$. Remove $\mathbb{G}$ and $\mathbb{G}'$ from Γ and add $\mathbb{G} \otimes \mathbb{G}'$ to Γ.

Shrink Choose some $\mathbb{G} = \langle S, E\rangle \in \Gamma$, $S' \subset S$ and surjective function $h : S \to S'$. Define $\mathbb{G}' = \langle S', E'\rangle$ where $E' = \{\langle h(s), h(t), \ell\rangle \mid \langle s, t, \ell\rangle \in E\}$. Replace $\mathbb{G}$ with $\mathbb{G}'$ in Γ.

Reduce labels Let L be the set of labels in Γ. Choose a new label set L' and define a function $\lambda : L \to L'$. Replace each $\mathbb{G} = \langle S, E\rangle$ in Γ with $\mathbb{G}' = \langle S, E'\rangle$, where $E' = \{\langle s, t, \lambda(\ell)\rangle \mid \langle s, t, \ell\rangle \in E\}$.

These operations can be arbitrarily interleaved with each other. The process typically stops when $|\Gamma| = 1$ and no more shrinking is required to achieve the desired size of the abstract state space. Let $\mathbb{G} = \langle S, E\rangle$ be an STG in Γ at some point during this process. Then $S \subseteq \mathcal{T}(V' \cdot D)$ for some $V' \subseteq V$ and $S \neq \mathcal{T}(V' \cdot D)$ if and only if $\mathbb{G}$ is the result of applying at least one shrink operation. Let $\mathbb{G}_A = \langle S_A, E_A\rangle$ be the final single STG in Γ. Then $S_A \subseteq \mathcal{T}(V \cdot D)$ and $|S_A| \leq N$ for some predefined value N.

Once again, let $\mathbb{G} = \langle S, E\rangle$ be an STG in Γ at some point during the process. Although S will often contain states defined by more than one variable, we may view S itself as the domain of a new single compound variable (which is also how **M&S** is usually described in the literature). Hence, the shrink operation is nothing else but method **VDA** (Definition 34) where all h_v functions but one are the identity function, i.e. it only abstracts the domain of one compound variable.

Let $op_1, \ldots, op_m$ be a sequence of merge/shrink/reduce labels operations. Let $\Gamma_0 = \{\mathbb{G}_{v_1}, \ldots, \mathbb{G}_{v_n}\}$ and let Γ_i be the result of applying operation op_i on Γ_{i-1} for all i $(1 \leq i \leq m)$. For each Γ_i, define the STG $\mathbb{G}_i^A = \langle S_i^A, E_i^A\rangle$ as $\otimes_{\mathbb{G} \in \Gamma_i} \mathbb{G}$. We can then treat the sequence $\mathbb{G}_1^A, \ldots, \mathbb{G}_m^A$ as a hierarchy of abstractions of $\mathbb{G}_0^A = \mathbb{G}(\mathbb{F})$.

For arbitrary i $(1 \leq i \leq m)$, let $\mathbb{G}_1, \ldots, \mathbb{G}_p$ be the STGs in Γ_{i-1}. Then $\mathbb{G}_i^A = \mathbb{G}_1 \otimes \mathbb{G}_2 \otimes \ldots \otimes \mathbb{G}_p$. There are three cases to consider:

1. Suppose op_i is a merge operation of $\mathbb{G}_j$ and $\mathbb{G}_k$ in Γ_{i-1}, where $j \neq k$. Without losing generality, assume $j = 1$ and $k = 2$. Then we construct $\Gamma_i = \{\mathbb{G}_1 \otimes \mathbb{G}_2, \mathbb{G}_3, \ldots, \mathbb{G}_p\}$. We get $\mathbb{G}_i^A = (\mathbb{G}_1 \otimes \mathbb{G}_2) \otimes \ldots \otimes \mathbb{G}_p = \mathbb{G}_{i-1}^A$, since synchronous product is associative. We can thus define a transformation function f_i from $\mathbb{G}_{i-1}^A$ to $\mathbb{G}_i^A$ as the identity function, i.e. $f(s) = s$. We further note that $L(\mathbb{G}_{v_j} \otimes \mathbb{G}_{v_k}) = L(\mathbb{G}_{v_j}) = L(\mathbb{G}_{v_k})$ and that $L(\mathbb{G}_{v_j}) = L(\mathbb{G}_{v_k}) = L(\mathbb{G}(\mathbb{F}))$, as previously noted, so $L(\mathbb{G}_{v_j} \otimes \mathbb{G}_{v_k}) = L(\mathbb{G}(\mathbb{F}))$. We can thus define a label relation $R_i \subseteq L(\mathbb{G}_{i-1}^A) \times L(\mathbb{G}_i^A)$ as the identity relation, i.e. $R_i(\ell, \ell)$ for all $\ell \in L(\mathbb{G}_{i-1}^A) = L(\mathbb{G}_i^A)$. Then define $\tau_i = \langle f_i, R_i\rangle$, which is an $\mathbf{M}_\uparrow\mathbf{R}_\updownarrow\mathbf{C}_\updownarrow$ transformation since f_i is the identity function, R_i the identity relation and $\mathbb{G}_i^A = \mathbb{G}_{i-1}^A$.

2. Instead suppose op_i is a shrink operation on $\mathbb{G}_j = \langle S_j, E_j\rangle \in \Gamma_{i-1}$ with result $\mathbb{G}'_j = \langle S'_j, E'_j\rangle$ and shrinking function $h_i : S_j \to S'_j$. Without losing generality, assume $j = 1$. Then $\mathbb{G}_i^A = h_i(\mathbb{G}_1) \otimes \mathbb{G}_2 \otimes \ldots \otimes \mathbb{G}_p$. Let V_j be the variables defining the state space S_j. Define a function f_i such that $f_i(s) = s[V \setminus V_j] \cup h_i(s[V_j])$ for all $s \in \mathbb{G}_{i-1}^A$. Then f_i is a transformation function from $\mathbb{G}_{i-1}^A$ to $\mathbb{G}_i^A$. We also have that $L(\mathbb{G}'_j) = L(\mathbb{G}_j) = \mathbb{G}(\mathbb{F})$, so also in this case we can define the label relation $R_i \subseteq L(\mathbb{G}_{i-1}^A) \times L(\mathbb{G}_i^A)$ as the identity relation. Finally, define $\tau_i = \langle f_i, R_i\rangle$, which is a transformation. We note that f_i can also be viewed as a shrinking function from S_{i-1}^A to S_i^A, so τ_i is a **VDA** abstraction, and it is thus also $\mathbf{M}_\uparrow\mathbf{R}_\updownarrow\mathbf{C}_\updownarrow$.

3. Finally, suppose op_i is a reduce labels operation. Let λ_i be the label reduction function for this step. For each $\mathbb{G}_j = \langle S_j, E_j\rangle$ in Γ_{i-1}, let $\mathbb{G}'_j = \langle S_j, E'_j\rangle$ where $E'_j = \{\langle s, t, \lambda_i(\ell)\rangle \mid \langle s, t, \ell\rangle \in E_j\}$. Then $\Gamma_i = \{\mathbb{G}'_1, \ldots, \mathbb{G}'_p\}$ and $\mathbb{G}_i^A = \mathbb{G}'_1 \otimes \mathbb{G}'_2 \otimes \ldots \otimes \mathbb{G}'_p$. Clearly, $\mathbb{G}_{i-1}^A$ and $\mathbb{G}_i^A$ have the same state sets, so we can define the transformation function f_i for this step as the identity function. Also define the label relation R_i such that $R_i(\ell, \lambda_i(\ell))$ for all $\ell \in L(\mathbb{G}_{i-1}^A)$. Define the transformation $\tau_i = \langle f_i, R_i\rangle$, which is obviously $\mathbf{M}_\uparrow$ since f_i is the identity function. Furthermore, if $\langle s, t, \ell\rangle$ is an arc in $\mathbb{G}_{i-1}^A$, then it is straightforward that $\langle s, t, \lambda_i(\ell)\rangle$ is an arc in $\mathbb{G}_i^A$, and it follows that τ_i is $\mathbf{R}_\uparrow$ and $\mathbf{C}_\uparrow$. Finally, op_i is a reduce labels operation so τ_i is $\mathbf{R}_\downarrow$ and $\mathbf{C}_\downarrow$ due to the definition of reduce labels operations.

It follows that each op_i implements an $\mathbf{M}_\uparrow\mathbf{R}_\updownarrow\mathbf{C}_\updownarrow$ transformation, so it follows by repeated application of Theorem 63 that the composite transformation $\tau = \tau_1 \circ \tau_2 \circ \ldots \circ \tau_m$ is $\mathbf{M}_\uparrow\mathbf{R}_\updownarrow\mathbf{C}_\updownarrow$. That is, **M&S** abstraction is $\mathbf{M}_\uparrow\mathbf{R}_\updownarrow\mathbf{C}_\updownarrow$ and, thus, also inherits properties $\mathbf{P}_{1\downarrow}$ and $\mathbf{P}_{\mathbf{S}\uparrow}$ by Theorem 22 and Corollary 23. We can finally use this result in combination with Theorem 13 as an alternative way to deduce that **M&S** abstractions are strong homomorphisms (which is already known in the literature).

[10] Note that there are various definitions of label reduction in the literature. We follow the definition in [85], which is simpler and more general than previous approaches.

We finally note that **M&S** abstraction is very similar to a method used in model checking, where clusters of variables are merged to single variables and domain abstraction is then performed on these variables [19].

10. Discussion

We have presented a general and flexible framework for modelling different methods for abstraction and similar concepts in search and planning. We have shown that this framework enables us to study many different aspects of both general methods and individual problem instances. In particular, it allowed for comparing a number of different abstraction methods on an abstract level, which high-lighted similarities and differences between the methods. It also allowed for a similar high-level comparison of abstraction refinement and abstraction heuristics, which is important for understanding their connections. However, the properties and classifications we have defined should only be viewed as examples of what our framework can do. The strength of a unifying framework is that it opens up for a multitude of different comparisons and analyses. We acknowledge that the framework may need adjustments and/or generalisations in order to be applicable in different situations—such variations appear to be simple to implement in many cases, though. At least one such example already exists. Sievers [84] defined a more restricted transformation concept for transition systems with the purpose of analysing transformation steps for **M&S** abstraction. Similar to us, he identified a number of properties that such transformations may have, with a focus on properties, and combinations of properties, that are beneficial when using the transformed instances to compute heuristics. Sieves acknowledges that the earlier conference publications of our work was one of the major inspirations for defining a transformation concept and study properties of transformations. We will briefly discuss a number of other ways to use or extend our framework below.

10.1. Labelled transition systems

We consider labelled state-transition graphs in order to focus on the general properties of various abstraction methods. It is also common to consider specific cases where also a set of initial states (or a single such state) is specified. This is common not only in search and planning, but also in other areas such as model checking. We can then specify a structure $\langle S, E, I\rangle$ where $I \subseteq S$ is the set of initial states. Such a structure is often referred to as a *labelled transition system* [49]). In this case, we may not want to require that certain properties hold for the whole graph $\langle S, E\rangle$, but only for the part that is reachable from I. Suppose we have a transformation $\tau = \langle f, R\rangle$ from $\mathbb{G}_1 = \langle S_1, E_1\rangle$ to $\mathbb{G}_2 = \langle S_2, E_2\rangle$ and a set $I \subseteq S_1$ of initial states. Then $S'_1 = \mathcal{R}_1(I)$ and $S'_2 = \mathcal{R}_2(f(I))$ are the ground and abstract state spaces reachable from I and $f(I)$ respectively. We can then construct the restricted STGs $\mathbb{G}'_1 = \mathbb{G}_{1|S'_1}$ and $\mathbb{G}'_2 = \mathbb{G}_{2|S'_2}$ and restrict τ accordingly. Our framework can then be directly applied to this restriction. It is also common to additionally specify a set G of goal states. We may then in some cases wish to define $S'_1 = \{s \in S_1 \mid s \in \mathcal{R}_1(I) \text{ and } \mathcal{R}_1(s) \cap G \neq \varnothing\}$ and similarly for S'_2, i.e. we consider only the space of states that are on a path from I to G. We conclude that our choice of defining the framework using STGs rather than specific structures is more flexible. However, an interesting case here is **GIDL** abstraction, which is known to be admissible, but does not even satisfy the $\mathbf{P_1}_{\uparrow}$ property. Consider a SAS$^+$ instance $\mathbb{P}_1 = \langle V_1, D_1, A_1, I_1, G_1\rangle$ and its corresponding **GIDL** transformation $\mathbb{P}_2 = \langle V_2, D_2, A_2, I_2, G_2\rangle$. Suppose there is a path $s_0, s_1, \ldots, s_n$ in $\mathbb{G}(\mathbb{P}_1)$ such that $s_0 = I_1$ and $G_1 \subseteq s_n$. Then there is a path $t_0, t_1, \ldots, t_n$ in $\mathbb{G}(\mathbb{P}_2)$ such that $t_0 = I_2$ and $s_i{}^{=1} \subseteq t_i{}^{=1}$ for all i $(0 \leq i \leq n)$, and it follows that $G_2 \subseteq t_n$. This means that a solution path for a specific instance can always be weakly upwards refined into a corresponding abstract solution. It could, thus, make sense to also consider variants of our properties that apply only to specific instances. Such properties should not replace the ones defined in this article, but rather complement them. A similar case arises for admissibility of heuristics, which is defined to hold for all paths in the instance, disregarding specific initial states and goals. The idea of globally admissible heuristics [62] restricts the concept such that admissibility only needs to hold for specific paths, which is similar to the suggestion above of adding a restricted case of refinability.

10.2. Metric refinement

The concept of *simulation* is often used in model checking [19]) and sometimes also in planning [75]. An abstraction $\mathbb{G}_2$ of $\mathbb{G}_1$ is a simulation of $\mathbb{G}_1$ if every path $t_0, t_1, \ldots, t_n$ in $\mathbb{G}_2$ has a corresponding path $s_0, s_1, \ldots, s_n$ in $\mathbb{G}_1$ such that $s_i \in \underline{f}(t_i)$ for all i. Fan and Holte [33] extend the concept of spurious states in search to *spurious paths*, based on the simulation concept. A path in $\mathbb{G}_2$ is spurious if it has no corresponding simulation in $\mathbb{G}_1$, which is analogous to the concept of spurious counterexamples in model checking [19]. Simulation is a stronger concept than $\mathbf{P}_{\mathbf{W}\downarrow}$ since each abstract arc must be refined into a single ground arc, not a path. Hence, the simulation concept is related to the results in Section 8, which demonstrate that the lack of metrics in usual refinement concepts makes it hard to prove further positive results on the connections between refinement and heuristics. The obvious way forward would be to somehow add metric aspects also to refinements. The properties $\mathbf{P}_{k\uparrow}$ and $\mathbf{P}_{k\downarrow}$ do so, but not in a sufficient way. For instance, one might consider a property $\mathbf{P}^m_{k\downarrow}$ that is like $\mathbf{P}_{k\downarrow}$ but additionally requires that each arc along the path can be refined into a path of length m at most. Obviously, the simulation concept could then be covered by the property $\mathbf{P}^1_{k\downarrow}$. This might also allow for finding tighter relationships between refinement and heuristics, perhaps relating $\mathbf{P}^m_{k\downarrow}$ to approximate heuristic search, e.g. using weighted A* [29]. However, it could also provide a deeper insight into refinement itself; it is well known that purely qualitative

criteria can cause anomalous behaviour in refinement such as exponential slow-down of the search process [5]. Another possibility is a property expressing that every path σ in $\mathbb{G}_2$ can be loosely refined into a path that is at most k times longer than σ, which would open up for connections with approximation algorithms.

10.3. Other abstraction methods

We will now briefly give some further examples of abstractions that can be modelled in our framework. Sturtevant and Buro [88] build abstractions for path planning in connected graphs. Somewhat simplified, they let each clique in the graph become an abstract state, and then repeat this process hierarchically. The graphs are not directed in this case, but almost all of our framework is applicable also to undirected graphs, so we can see that each abstraction step in their framework seems to result in a transformation that is $\mathbf{P_{S\downarrow}}$, thus allowing for easy refinement. Some abstraction methods are even more procedural in nature. For instance, the STAR method [59] builds an abstraction by arbitrarily choosing one hub state at a time, and aggregates it with all non-abstracted states within a predefined radius to form an equivalence class to be abstracted to one abstract state. Hierarchies can also be built in this way. As yet another example, Heusner et al. [50] have recently suggested a method where one first tries to find a plan with only a subset of all actions available. As long as this fails, the restricted action set is gradually increased. We can view this as an abstraction hierarchy where the top level corresponds to the initial restricted action set and the bottom level to the full original action set. This results in a hierarchy where all levels have the same state set, but where the arc set at one level is always a subset of the arc set on the level below.

In the case of abstraction heuristics, one might consider also non-$\mathbf{M}_\uparrow$ abstractions, but defining such heuristics is much less straightforward. For instance, we can no longer exploit ordinary homomorphisms, and the literature on this topic is very scarce. One interesting exception is *multimapping abstractions* [76] that is a method for aggregating multiple heuristics. In order to make the heuristic admissible, it must satisfy that

$$h(s,t) = \max_{s' \in f(s)} \min_{t' \in f(t)} d(s',t').$$

However, in their case, the function f does not necessarily induce a partition on the states in the abstract graph, so modelling multi-mapping abstractions would also require relaxing the definition of transformation functions, which may have a fundamental impact on our framework. There is also recent work by Steinmetz and Torralba [87] who consider abstract states, called concepts, and abstractions that map each ground state to a set of abstract states. Contrary to our theory, it is possible that two ground states map to distinct but overlapping sets of abstract states, so the abstraction does not define a partition on the abstract states. Another difference is that the abstract graph is a hypergraph, induced by the actions. This is used to compute a heuristic such that $h(s,t)$ is the minimal distance from $f(s)$ to $f(t)$ in this hypergraph.

We have also previously noted that the **ABS** method may be viewed as a kind of generalised embedding. This can be formalised to consider generalised variants of embeddings, retractions and homomorphisms. For instance, it is straightforward and natural to define a generalised homomorphism such that if $\langle s,t,\ell\rangle \in E_1$, then there is some $\langle s',t',\ell'\rangle \in E_2$ such that $s' \in f(s)$ and $t' \in f(t)$.

10.4. Other applications: hierarchical graphs

We will finally sketch an example of how our framework could be applied in other contexts than search and planning. It is common to represent graphs in hierarchical representations, where a top-level abstraction of the graph can be refined into increasing levels of detail. Such representations are important in many application areas, e.g. to speed up graph operations in CAD systems [70]. It is usually desirable that such representations retain certain interesting properties at the abstract levels. Ancona et al. [2] considered *structured graphs*, where the abstraction consists of reducing subgraphs with certain properties into a single vertex or arc. In particular, they studied properties of paths and whether a path in the original graph can be adequately represented by a structured path, i.e. a path that is defined via the abstraction hierarchy. We will not go into details of this topic, but briefly sketch how our framework might be applied in this context. Let us consider vertex-structured graphs, where a subgraph can be reduced into a single vertex. Let $G = \langle V, E\rangle$ be the original graph and $G_1 = \langle V_1, E_1\rangle, \ldots, G_n = \langle V_n, E_n\rangle$ be the disjoint subgraphs to reduce. Let $v_1, \ldots, v_n$ be the new vertices corresponding to these subgraphs, i.e. the abstract graph will have the vertex set $V' = (V \setminus \bigcup_{i=1}^{n} V_i) \cup \{v_1, \ldots, v_n\}$. It is then natural to define a transformation function f such that $f(v) = v_i$ if $v \in V_i$ for some subgraph G_i and $f(v) = v$ for all vertices not in any of the subgraphs. Given the requirements stated by the authors for when a subgraph may be reduced, it follows that this abstraction is $\mathbf{P_{W\updownarrow}}$. Since f is obviously $\mathbf{M}_\uparrow$, it also follows that it is $\mathbf{P_{S\uparrow}}$. This only applies to ordinary paths in the graphs, though, so new properties would have to be defined if we want to also analyse structured paths.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. List of key concepts

Table A.2
Key concepts in this article.

Concept	Abbreviation	Definition
Upwards many-one	$\mathbf{M}_{\uparrow}$	Definition 9
Downwards many-one	$\mathbf{M}_{\downarrow}$	Definition 9
Upwards related	$\mathbf{R}_{\uparrow}$	Definition 9
Downwards related	$\mathbf{R}_{\downarrow}$	Definition 9
Upwards coupled	$\mathbf{C}_{\uparrow}$	Definition 9
Downwards coupled	$\mathbf{C}_{\downarrow}$	Definition 9
Loosely upwards state refinable	$\mathbf{P}_{\mathbf{L}\uparrow}$	Definition 16
Loosely downwards state refinable	$\mathbf{P}_{\mathbf{L}\downarrow}$	Definition 16
Weakly upwards state refinable	$\mathbf{P}_{\mathbf{W}\uparrow}$	Definition 16
Weakly downwards state refinable	$\mathbf{P}_{\mathbf{W}\downarrow}$	Definition 16
Strongly upwards state refinable	$\mathbf{P}_{\mathbf{S}\uparrow}$	Definition 16
Strongly downwards state refinable	$\mathbf{P}_{\mathbf{S}\downarrow}$	Definition 16
Downward path preserving	DPP	Sec. 4.1
Completeness of DPP	$\mathbf{P}_{\uparrow}$	Definition 16
Soundness of DPP	$\mathbf{P}_{\downarrow}$	Definition 16
Downward refinement property	DRP	Sec. 4.1
Abstrips-style abstraction	**ABS**	Definition 30
Variable projection	**VP**	Definition 32
Variable domain abstraction	**VDA**	Definition 34
Removing redundant actions	**RRA**	Definition 36
Generalised ignoring delete lists	**GIDL**	Definition 38
Direct landmark-based surrogates	**DLBS**	Definition 40
Merge-and-shrink	**M&S**	Sec. 9.4
Admissibility	$\mathbf{A}_{\downarrow}$	Definition 46
Conditional admissibility	$\mathbf{A}_{\mathbf{C}\downarrow}$	Definition 46

References

[1] M. Aghighi, C. Bäckström, Cost-optimal and net-benefit planning–a parameterised complexity view, in: Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, 2015, pp. 1487–1493.
[2] M. Ancona, L.D. Floriani, J.S. Deogun, Path problems in structured graphs, Comput. J. 29 (1986) 553–563.
[3] F. Bacchus, Q. Yang, Downward refinement and the efficiency of hierarchical problem solving, Artif. Intell. 71 (1994) 43–100.
[4] C. Bäckström, Expressive equivalence of planning formalisms, Artif. Intell. 76 (1995) 17–34.
[5] C. Bäckström, P. Jonsson, Planning with abstraction hierarchies can be exponentially less efficient, in: Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI 1995, Montréal QC, Canada, 1995, pp. 1599–1605.
[6] C. Bäckström, P. Jonsson, Abstracting abstraction in search with applications to planning, in: Principles of Knowledge Representation and Reasoning: Proceedings of the 13th International Conference, KR 2012, Rome, Italy, 2012, pp. 446–456.
[7] C. Bäckström, P. Jonsson, Bridging the gap between refinement and heuristics in abstraction, in: Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013, Beijing, China, 2013, pp. 2261–2267.
[8] C. Bäckström, P. Jonsson, S. Ordyniak, S. Szeider, A complete parameterized complexity analysis of bounded planning, J. Comput. Syst. Sci. 81 (2015) 1311–1332.
[9] C. Bäckström, P. Jonsson, S. Ståhlberg, Fast detection of unsolvable planning instances using local consistency, in: Proceedings of the 6th Annual Symposium on Combinatorial Search, SoCS 2013, Leavenworth, WA, USA, 2013, pp. 29–37.
[10] C. Bäckström, I. Klein, Planning in polynomial time: the SAS-PUBS class, Comput. Intell. 7 (1991) 181–197.
[11] C. Bäckström, B. Nebel, Complexity results for SAS^{+} planning, Comput. Intell. 11 (1995) 625–656.
[12] J. Balcázar, The complexity of searching implicit graphs, Artif. Intell. 86 (1996) 171–188.
[13] S. Bogomolov, D. Magazzeni, A. Podelski, M. Wehrle, Planning as model checking in hybrid domains, in: Proceedings of the 28th AAAI Conference on Artificial Intelligence, AAAI 2014, Québec City, QC, Canada, 2014, pp. 2228–2234.
[14] B. Bonet, G. Loerincs, H. Geffner, A robust and fast action selection mechanism for planning, in: Proceedings of the 14th National Conference on Artificial Intelligence, AAAI 1997, Providence, RI, USA, 1997, pp. 714–719.
[15] A. Botea, M. Enzenberger, M. Müller, J. Schaeffer, Macro-FF: improving AI planning with automatically learned macro-operators, J. Artif. Intell. Res. 24 (2005) 581–621.
[16] A. Bundy, F. Giunchiglia, R. Sebastiani, T. Walsh, Calculating criticalities, Artif. Intell. 88 (1996) 39–67.
[17] T. Bylander, The computational complexity of propositional Strips planning, Artif. Intell. 69 (1994) 165–204.
[18] B.Y. Choueiry, Y. Iwasaki, S.A. McIlraith, Towards a practical theory of reformulation for reasoning about physical systems, Artif. Intell. 162 (2005) 145–204.
[19] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided abstraction refinement for symbolic model checking, J. ACM 50 (2003) 752–794.
[20] E.M. Clarke, O. Grumberg, D.E. Long, Model checking and abstraction, ACM Trans. Program. Lang. Syst. 16 (1994) 1512–1542.
[21] P. Cousot, R. Cousot, Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, in: Conference Record of the 4th ACM Symposium on Principles of Programming Languages, POPL 1977, Los Angeles, CA, USA, 1977, pp. 238–252.
[22] P. Cousot, R. Cousot, Abstract interpretation: past, present and future, in: Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS 2014, Vienna, Austria, 2014, 2.

[23] J.C. Culberson, J. Schaeffer, Pattern databases, Comput. Intell. 14 (1998) 318–334.
[24] R. Dechter, J. Pearl, Generalized best-first search strategies and the optimality of A*, J. ACM 32 (1985) 505–536.
[25] C. Domshlak, J. Hoffmann, M. Katz, Red-black planning: a new systematic approach to partial delete relaxation, Artif. Intell. 221 (2015) 73–114.
[26] C. Domshlak, J. Hoffmann, A. Sabharwal, Friends or foes? On planning as satisfiability and abstract CNF encodings, J. Artif. Intell. Res. 36 (2009) 415–469.
[27] C. Domshlak, M. Katz, S. Lefler, When abstractions met landmarks, in: Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, ON, Canada, 2010, pp. 50–56.
[28] C. Domshlak, A. Nazarenko, The complexity of optimal monotonic planning: the bad, the good, and the causal graph, J. Artif. Intell. Res. 48 (2013) 783–812.
[29] R. Ebendt, R. Drechsler, Weighted A* search - unifying view and application, Artif. Intell. 173 (2009) 1310–1342.
[30] S. Edelkamp, S. Leue, W. Visser, Summary of Dagstuhl seminar 06172 on directed model checking, in: Directed Model Checking, 2007.
[31] S. Edelkamp, S. Schrödl, Heuristic Search - Theory and Applications, Academic Press, 2012.
[32] S. Eriksson, G. Röger, M. Helmert, A proof system for unsolvable planning tasks, in: Proceedings of the 28th International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, 2018, pp. 65–73.
[33] G. Fan, R.C. Holte, The spurious path problem in abstraction, in: Proceedings of the 8th Annual Symposium on Combinatorial Search, SoCS 2015, Ein Gedi, Israel, 2015, pp. 18–27.
[34] A. Felner, U. Zahavi, R. Holte, J. Schaeffer, N.R. Sturtevant, Z. Zhang, Inconsistent heuristics in theory and practice, Artif. Intell. 175 (2011) 1570–1603.
[35] H. Galperin, A. Wigderson, Succinct representations of graphs, Inf. Control 56 (1983) 183–198.
[36] J. Gaschnig, A problem similarity approach to devising heuristics: first results, in: Proceedings of the 6th International Joint Conference on Artificial Intelligence, IJCAI 1979, Tokyo, Japan, 1979, pp. 301–307.
[37] F. Geißer, T. Keller, R. Mattmüller, Abstractions for planning with state-dependent action costs, in: Proceedings of the 26th International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, 2016, pp. 140–148.
[38] F. Giunchiglia, A. Villafiorita, T. Walsh, Theories of abstraction, AI Commun. 10 (1997) 167–176.
[39] F. Giunchiglia, T. Walsh, A theory of abstraction, Artif. Intell. 57 (1992) 323–389.
[40] S.G. Govindaraju, D.L. Dill, Verification by approximate forward and backward reachability, in: Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1998, San Jose, CA, USA, 1998, pp. 366–370.
[41] P. Gregory, D. Long, M. Fox, J.C. Beck, Planning modulo theories: extending the planning paradigm, in: Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS-2012), 2012.
[42] P. Gregory, D. Long, C. McNulty, S.M. Murphy, Exploiting path refinement abstraction in domain transition graphs, in: Proceedings of the 25th AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, CA, USA, 2011, pp. 971–976.
[43] G. Guida, M. Somalvico, A method for computing heuristics in problem solving, Inf. Sci. 19 (1979) 251–259.
[44] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (1968) 100–107.
[45] P. Haslum, A. Botea, M. Helmert, B. Bonet, S. Koenig, Domain-independent construction of pattern database heuristics for cost-optimal planning, in: Proceedings of the 22nd AAAI Conference on Artificial Intelligence, AAAI 2007, Vancouver, BC, Canada, 2007, pp. 1007–1012.
[46] P. Haslum, P. Jonsson, Planning with reduced operator sets, in: Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems, AIPS 2000, Breckenridge, CO, US, 2000, pp. 150–158.
[47] M. Helmert, C. Domshlak, Landmarks, critical paths and abstractions: what's the difference anyway?, in: Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, 2009, pp. 162–169.
[48] M. Helmert, P. Haslum, J. Hoffmann, Flexible abstraction heuristics for optimal sequential planning, in: Proceedings of the 17th International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, RI, USA, 2007, pp. 176–183.
[49] M. Helmert, P. Haslum, J. Hoffmann, R. Nissim, Merge-and-shrink abstraction: a method for generating lower bounds in factored state spaces, J. ACM 61 (2014) 16.
[50] M. Heusner, M. Wehrle, F. Pommerening, M. Helmert, Under-approximation refinement for classical planning, in: Proceedings of the 24th International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, NH, USA, 2014, pp. 365–369.
[51] J. Hoffmann, Where 'ignoring delete lists' works: local search topology in planning benchmarks, J. Artif. Intell. Res. 24 (2005) 685–758.
[52] J. Hoffmann, P. Kissmann, Á. Torralba, "Distance"? Who cares? Tailoring merge-and-shrink heuristics to detect unsolvability, in: Proceedings of the 21st European Conference on Artificial Intelligence, ECAI 2014, Prague, Czech Republic, 2014, pp. 441–446.
[53] J. Hoffmann, J. Porteous, L. Sebastia, Ordered landmarks in planning, J. Artif. Intell. Res. 22 (2004) 215–278.
[54] J. Hoffmann, A. Sabharwal, C. Domshlak, Friends or foes? An AI planning perspective on abstraction and search, in: Proceedings of the 16th International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, 2006, pp. 294–303.
[55] R. Holte, B. Arneson, N. Burch, PSVN Manual, Technical Report TR14-03, Department of Computing Science, University of Alberta, Edmonton, AB, Canada, 2014.
[56] R.C. Holte, Common misconceptions concerning heuristic search, in: A. Felner, N.R. Sturtevant (Eds.), Proceedings of the 3rd Annual Symposium on Combinatorial Search, SOCS 2010, Stone Mountain, Atlanta, GA, USA, July 8-10, 2010, AAAI Press, 2010, pp. 46–51.
[57] R.C. Holte, B. Choueiry, Abstraction and reformulation in artificial intelligence, Philos. Trans. R. Soc. Lond. B 29 (358) (2003) 1197–1204.
[58] R.C. Holte, I.T. Hernádvölgyi, A space-time tradeoff for memory-based heuristics, in: Proceedings of the 16th National Conference on Artificial Intelligence, AAAI 1999, Orlando, FL, USA, 1999, pp. 704–709.
[59] R.C. Holte, T. Mkadmi, R.M. Zimmer, A.J. MacDonald, Speeding up problem solving by abstraction: a graph oriented approach, Artif. Intell. 85 (1996) 321–361.
[60] R.C. Holte, M.B. Perez, R.M. Zimmer, A.J. MacDonald, The Tradeoff Between Speed and Optimality in Hierarchical Search, Technical Report TR-95-19, Computer Science, University of Ottawa, 1995.
[61] R.C. Holte, M.B. Perez, R.M. Zimmer, A.J. MacDonald, Hierarchical A*: searching abstraction hierarchies efficiently, in: Proceedings of the 13th National Conference on Artificial Intelligence, AAAI 1996, Portland, OR, USA, 1996, pp. 530–535.
[62] E. Karpas, C. Domshlak, Optimal search with inadmissible heuristics, in: Proceedings of the 22nd International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, 2012, pp. 92–100.
[63] C.A. Knoblock, A theory of abstraction for hierarchical planning, in: D.P. Benjamin (Ed.), Change of Representation and Inductive Bias, Kluwer, Boston, MA, USA, 1990, pp. 81–104.
[64] C.A. Knoblock, Search reduction in hierarchical problem solving, in: Proceedings of the 9th National Conference on Artificial Intelligence, AAAI 1991, Anaheim, CA, USA, 1991, pp. 686–691.
[65] C.A. Knoblock, Automatically generating abstractions for planning, Artif. Intell. 68 (1994) 243–302.
[66] C.A. Knoblock, J.D. Tenenberg, Q. Yang, Characterizing abstraction hierarchies for planning, in: Proceedings of the 9th National Conference on Artificial Intelligence, AAAI 1991, Anaheim, CA, USA, 1991, pp. 692–697.
[67] R.E. Korf, Macro-operators: a weak method for learning, Artif. Intell. 26 (1985) 35–77.
[68] R.E. Korf, Planning as search: a quantitative approach, Artif. Intell. 33 (1987) 65–88.

[69] R.E. Korf, Linear-time disk-based implicit graph search, J. ACM 55 (2008) 26.
[70] T. Lengauer, K.W. Wagner, The correlation between the complexities of the nonhierarchical and hierarchical versions of graph problems, J. Comput. Syst. Sci. 44 (1992) 63–93.
[71] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL—the Planning Domain Definition Language, Technical Report CVC TR98003/DCS TR1165, Yale Center for Computational Vision and Control, New Haven, CT, USA, 1998.
[72] D.V. McDermott, A heuristic estimator for means-ends analysis in planning, in: Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems, AIPS 1996, Edinburgh, UK, 1996, pp. 142–149.
[73] P.P. Nayak, A.Y. Levy, A semantic theory of abstractions, in: Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI 1995, Montréal QC, Canada, 1995, pp. 196–203.
[74] A. Newell, J.C. Shaw, H.A. Simon, Report on a general problem-solving program, in: IFIP Congress, Paris, France, UNESCO, 1959, pp. 256–264.
[75] R. Nissim, J. Hoffmann, M. Helmert, Computing perfect heuristics in polynomial time: on bisimulation and merge-and-shrink abstraction in optimal planning, in: Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011, Barcelona, Catalonia, Spain, 2011, pp. 1983–1990.
[76] B. Pang, R.C. Holte, Multimapping abstractions and hierarchical heuristic search, in: Proceedings of the 5th Annual Symposium on Combinatorial Search, SoCS 2012, Niagara Falls, ON, Canada, 2012, pp. 72–79.
[77] F. Pommerening, M. Helmert, B. Bonet, Abstraction heuristics, cost partitioning and network flows, in: Proceedings of the 27th International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, PA, USA, 2017, pp. 228–232.
[78] E.D. Sacerdoti, Planning in a hierarchy of abstraction spaces, Artif. Intell. 5 (1974) 115–135.
[79] E.D. Sacerdoti, The nonlinear nature of plans, in: Advance Papers of the 4th International Joint Conference on Artificial Intelligence, IJCAI 1975, Tbilisi, Georgia, USSR, 1975, pp. 206–214.
[80] L. Saitta, J.D. Zucker, Abstraction in Artificial Intelligence and Complex Systems, Springer, New York, 2013.
[81] Z.G. Saribatur, J.P. Wallner, S. Woltran, Explaining non-acceptability in abstract argumentation, in: Proceedings of the 24th European Conference on Artificial Intelligence, ECAI 2020, Santiago de Compostela, Spain, 2020, pp. 881–888.
[82] J. Seipp, M. Helmert, Counterexample-guided Cartesian abstraction refinement, in: Proceedings of the 23rd International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, 2013, pp. 347–351.
[83] J. Seipp, M. Helmert, Counterexample-guided Cartesian abstraction refinement for classical planning, J. Artif. Intell. Res. 62 (2018) 535–577.
[84] S. Sievers, Merge-and-Shrink Abstractions for Classical Planning, Ph.D. thesis, Universität Basel, 2017.
[85] S. Sievers, M. Wehrle, M. Helmert, Generalized label reduction for merge-and-shrink heuristics, in: C.E. Brodley, P. Stone (Eds.), Proceedings of the 28th AAAI Conference on Artificial Intelligence, Québec City, Québec, Canada, July 27-31, 2014, AAAI Press, 2014, pp. 2358–2366.
[86] D. Smith, M. Peot, A critical look at Knoblock's hierarchy mechanism, in: Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems, AIPS 1992, College Park, MD, USA, 1992, pp. 307–308.
[87] M. Steinmetz, Á. Torralba, Bridging the gap between abstractions and critical-path heuristics via hypergraphs, in: Proceedings of the 29th International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, 2019, pp. 473–481.
[88] N.R. Sturtevant, M. Buro, Partial pathfinding using map abstraction and refinement, in: Proceedings of the 20th National Conference on Artificial Intelligence, AAAI 2005, Pittsburgh, PA, USA, 2005, pp. 1392–1397.
[89] A. Tate, Generating project networks, in: Proceedings of the 5th International Joint Conference on Artificial Intelligence, Cambridge, MA, USA, 1977, pp. 888–893.
[90] M. Valtorta, A result on the computational complexity of heuristic estimates for the A* algorithm, Inf. Sci. 34 (1984) 47–59.
[91] F. Yang, J.C. Culberson, R. Holte, U. Zahavi, A. Felner, A general theory of additive state space abstractions, J. Artif. Intell. Res. 32 (2008) 631–662.
[92] S. Zilles, R.C. Holte, The computational complexity of avoiding spurious states in state space abstraction, Artif. Intell. 174 (2010) 1072–1092.
[93] J.D. Zucker, A grounded theory of abstraction in artificial intelligence, Philos. Trans. R. Soc. Lond. B 29 (358) (2003) 1293–1309.