Abstracting Abstraction in Search with Applications to Planning

Christer Bäckström and **Peter Jonsson**
Department of Computer Science, Linköping University
SE-581 83 Linköping, Sweden
christer.backstrom@liu.se peter.jonsson@liu.se

Abstract

Abstraction has been used in search and planning from the very beginning of AI. Many different methods and formalisms for 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 transformations on labelled graphs. Transformations can have certain method properties that are inherent in the abstraction methods and describe their fundamental modelling characteristics, and they can have certain instance properties that describe algorithmic and computational characteristics of problem instances. The usefulness of the framework is demonstrated by applying it to problems in both search and planning. First, we show that we can capture many search abstraction concepts (such as avoidance of backtracking between levels) and that we can put them into a broader context. We further model five different abstraction concepts from the planning literature. Analysing what method properties they have highlights their fundamental differences and similarities. Finally, we prove that method properties sometimes imply instance properties. Taking also those instance properties into account reveals important information about computational aspects of the five methods.

1 Introduction

1.1 Background and our Approach

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 in automated reasoning. It has, for example, been intensively used in AI, verification, electronic design and reasoning about physical systems. The literature is consequently vast and we refer the reader to the surveys by Giunchiglia et al. (1997) or Holte and Choueiry (2003) as suitable introductions.

Throughout this paper, we concentrate on abstraction in search and planning. Abstraction has a long history even if we restrict ourselves in this way; its use dates back to ABSTRIPS (Sacerdoti 1974) and even to the first version of GPS (Newell, Shaw, and Simon 1959). In order for abstraction to be useful, the abstract instance should 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 achieve in practice. It has been demonstrated in many ways that abstraction can be very effective at decreasing overall solution time but few, if any, methods give any guarantees. For instance, Knoblock (1994) proposed a way to automatically create abstractions and demonstrated that it could give exponential speed-up in certain cases while Bäckström and Jonsson (1995) showed that the method can also backfire by creating solutions that are exponentially longer than the optimal solutions. Abstraction is thus a method that can strike both ways and it requires a careful analysis of the application domain to know if abstraction is useful or not.

A large number of different abstraction and abstraction-like methods appear in the literature. Unfortunately, many of these methods are tied to particular formalisms which make them difficult to analyse and compare in a meaningful way. We present a framework for comparing and analysing abstraction and abstraction-like methods based on *transformations* between labelled graphs. The idea of using functions (typically *homomorphisms*) on graphs (or other structures) for describing abstractions is very natural and has appeared in the literature earlier, cf. Holte et al. (1996) or Helmert, Haslum, and Hoffmann (2007). We extend this idea by viewing transformations as tuples $\langle f, R \rangle$ where, loosely speaking, the function f describes the "structure" of the abstracted graph and R gives an "interpretation" of the abstracted labels. This gives us a plethora of possibilities to model and study different kinds of abstraction-like methods. We stress that we do *not* set out to create a grand theory of abstraction. There are attempts in the literature to define and study abstraction on a very general level which allow for an in-depth treatment of ontological aspects, cf. Giunchiglia and Walsh (1992) or Pandurang Nayak and Levy (1995). Our approach is much more pragmatic, and it is first and foremost intended for studying computational aspects of abstraction in search. This does not exclude that it may be useful in other contexts but we view this as an added bonus and not a primary goal. We also want to point out that our purpose is not to invent new abstraction methods but to enable formal analyses of previously proposed methods. However, we hope that an increased understanding of abstraction will inspire the invention of new and better methods.

1.2 Results

Within a framework based on transformations, it is natural to identify and study abstract properties of transformations. Almost every result in this paper is, in one way or another, based on this idea. We have found that it is convenient to divide such properties into two classes:

Method properties characterise different abstraction methods on an abstract level. For instance, we show that a particular choice of such properties exactly captures the abstraction concept used by Zilles and Holte (2010). Method properties are used for comparing and analysing general properties of abstraction methods but they do not provide any information about specific problem instances.

Instance properties capture properties of problem instances, typically computational ones. For example, we show that our instance properties can be used to characterise different notions of backtrack-free search. Instance properties are used for studying computational aspects of problem instances but they are not dependent on the particular choice of abstraction method.

Both types of properties are discussed, in Sections 3 and 4, respectively. While this is a useful conceptual distinction, we allow ourselves to be flexible. We may, for example, say that a method has a certain instance property X: meaning that every instance that can be modelled by the method has property X. Once again, we emphasise that properties are defined on *transformations* and not on methods or instances. The idea of studying transformations abstractly gives us a powerful tool for analysing the relationship between modelling aspects and computational aspects. For example, it enables us to provide results of the type "if a problem instance I is modelled in a formalism with method property X, then I has instance property Y". Note that we do not restrict ourselves to a particular formalism here — we are only restricted to the class of methods having property X.

We now briefly describe what kind of concrete results we obtain. In Section 5, we take a closer look at the problem of avoiding backtracking between levels. As we have already pointed out, abstraction can, under certain conditions, slow down the search process substantially. One typical reason behind this adverse behaviour is backtracking between levels, i.e. when there are abstract solutions that cannot be refined to concrete solutions (and thus force the search algorithm to look for another abstract plan). This phenomenon has historically been a very active research area within planning and it still attracts a substantial amount of research. Partial solutions that have been presented include the ordered monotonicity criterion by Knoblock, Tenenberg, and Yang (1991), the downward refinement property (DRP) by Bacchus and Yang (1994), and the simulation-based approach by Bundy et al. (1996). Until now, no general conditions that fully capture this concept have been identified in the literature. We discuss three different notions of backtracking avoidance and show how these can be characterized within our framework.

In the final part of the paper (Sections 6–8), we study and compare different approaches to abstraction in automated planning. Abstraction has always attracted great interest in planning and there is a rich flora of different abstraction methods. Planning can be viewed as a special case of search where the state space is induced by a number of variables and the state transitions are induced by a set of actions. We demonstrate how to use transformations to model five different abstraction(-like) methods in planning, which highlights some of their differences and similarities. These five methods are a representative selection but there certainly are other methods worth studying, cf. Christensen (1990) and Fink and Yang (1997). We show how to derive instance properties from the method properties which shows that the five methods are all quite different. One interesting result is that the two different variants of ABSTRIPS, which seem to differ only marginally judged by their formal definitions, exhibit fundamentally different instance properties and, thus, computational properties. Another interesting result is that certain methods completely avoid *spurious states*, a property that is important, for example, when using abstraction for heuristic search (Haslum et al. 2007).

2 STGs and STG Transformations

We first introduce our framework for studying abstractions. Although the definitions may appear somewhat complex and difficult to understand at first sight, there is a reason: we want to *prove* results, not merely devote ourselves to discussions. We begin by defining some general notation and concepts, then we introduce state transition graphs and our transformation concept.

If X is a set, then $|X|$ denotes the cardinality of X. A *partition* of a set X is a set P of non-empty subsets of X such that (1) $\cup_{p \in P} p = X$ and (2) for all $p, q \in P$, if $p \neq q$, then $p \cap q = \varnothing$. Let $f : X \rightarrow Y$ be a function, then $Rng(f) = \{f(x) \mid x \in X\}$ is the *range* of f and the extension of f to subsets of X is defined as $f(Z) = \cup_{x \in Z} f(x)$ for all $Z \subseteq X$.

Before proceeding we remind the reader that when f is a function from X to 2^Y (for some sets X and Y), then $Rng(f) \subseteq 2^Y$, that is, the value of f is a subset of Y, not an element in Y.

Definition 1. A *state transition graph (STG)* over a set L of labels is a tuple $\mathbb{G} = \langle S, E \rangle$ where S is a set of vertices called *states* and $E \subseteq S \times S \times L$ is a set of labelled arcs. The set of labels in $\mathbb{G}$ is implicitly defined as $L(\mathbb{G}) = L(E) = \{\ell \mid \langle s, t, \ell \rangle \in E\}$. A sequence $s_0, s_1, \ldots, s_k$ of states in S is a *(state) path* in $\mathbb{G}$ if either (1) $k = 0$ or (2) there is some ℓ s.t. $\langle s_0, s_1, \ell \rangle \in E$ and $s_1, \ldots, s_k$ is a path in $\mathbb{G}$.

More than one arc in the same direction between two states is allowed, as long as the arcs have different labels. The intention of the labels is to provide a means to identify a subset of arcs by assigning a particular label to these arcs. This is useful, for instance, in planning where a single action may induce many arcs in an STG. We note that it is allowed to use the same label for every arc, in which case the STG concept collapses to an ordinary directed graph.

Definition 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 total function $f : S_1 \to 2^{S_2}$ is a *transformation function* from $\mathbb{G}_1$ to $\mathbb{G}_2$ if $Rng(f)$ is a partition of S_2. A *label relation* from $\mathbb{G}_1$ to $\mathbb{G}_2$ is a binary relation $R \subseteq L(\mathbb{G}_1) \times L(\mathbb{G}_2)$. An *(STG) 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 one STG to the other while the label relation R provides additional information about how sets of arcs are related between the two STGs. Note that f is formally a function from S_1 to 2^{S_2}, that is, it has a subset of S_2 as value. We use a function rather than a relation since it makes the theory clearer and simpler and is more in line with previous work in the area.

Example 3. Consider two STGs: $\mathbb{G}_1$: $00 \xrightarrow{a} 01 \xrightarrow{b} 10 \xrightarrow{a} 11$ and $\mathbb{G}_2 : 0 \xrightarrow{c} 1$. Also define $f_1 : \mathbb{G}_1 \to \mathbb{G}_2$ such that $f_1(xy) = \{x\}$. We see immediately that f_1 is a transformation function from $\mathbb{G}_1$ to $\mathbb{G}_2$. Define $f_2 : \mathbb{G}_1 \to \mathbb{G}_2$ 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 $Rng(f_2)$ does not partition $\mathbb{G}_2$. Finally, the function $f_3(xy) = \{2x + y, 7 - 2x - y\}$ is a transformation function from $\mathbb{G}_1$ to $\mathbb{G}_3$=$\langle \{0, \ldots, 7\}, \{\langle x, y, d \rangle \mid x \neq y\} \rangle$ since $Rng(f_3)$ partitions $\{0, \ldots, 7\}$ into $\{\{0, 7\}, \{1, 6\}, \{2, 5\}, \{3, 4\}\}$. The functions f_1 and f_3 are illustrated in Figure 1.

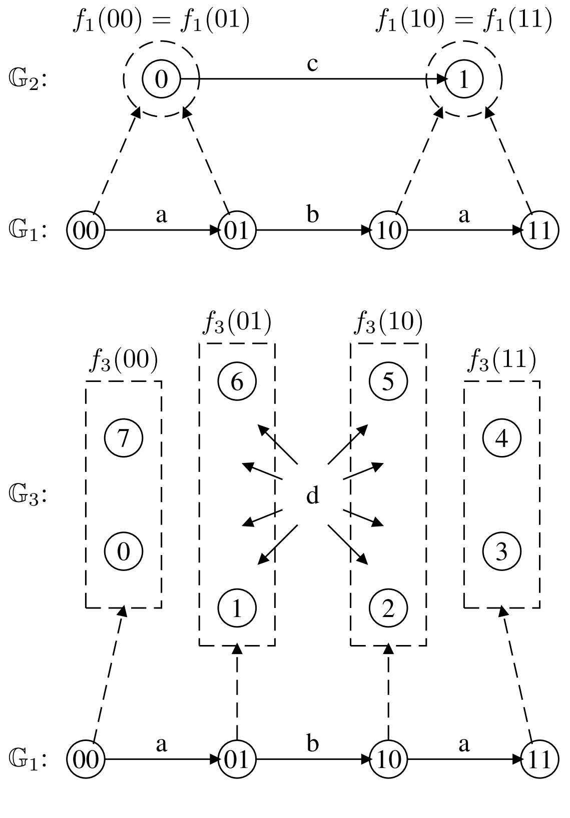

Figure 1: The functions f_1 and f_3 in Example 3.

A high degree of symmetry is inherent in our transformation concept. It is, in fact, 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 definitions of method and instance properties together with some of the forthcoming 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 $\mathbb{G}_1$ to $\mathbb{G}_2$ 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 as $\overline{f}(t) = \{s \in S_1 \mid t \in f(s)\}$ and the *reverse label relation* $\overline{R} \subseteq L(\mathbb{G}_2) \times L(\mathbb{G}_1)$ is defined as $\overline{R}(\ell_2, \ell_1)$ iff $R(\ell_1, \ell_2)$.

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 $\overline{f_3}(3) = \overline{f_3}(4) = \{11\}$.

Lemma 5. *Let f be a transformation function from an STG $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ to an STG $\mathbb{G}_2 = \langle S_2, E_2 \rangle$. Then:*

1) for all $s_1 \in S_1$, $s_2 \in S_2$, $s_1 \in \overline{f}(s_2)$ iff $s_2 \in f(s_1)$.

2) $\overline{f}$ is a transformation function from $\mathbb{G}_2$ to $\mathbb{G}_1$.

3) If $\langle f, R \rangle$ is a transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$, then $\langle \overline{f}, \overline{R} \rangle$ is a transformation from $\mathbb{G}_2$ to $\mathbb{G}_1$.

Proof. 1) Immediate from the definitions.

2) $S_2 = Rng(f)$ which readily implies that $\overline{f}$ is total. Suppose $s \in S_1$ and $s \notin Rng(\overline{f})$. Then there is some $t \in S_2$ s.t. $t \in f(s)$ but $s \notin \overline{f}(t)$, which contradicts (1). Hence, $Rng(\overline{f}) = S_1$. Let $t_1, t_2 \in S_2$ s.t. $\overline{f}(t_1) \neq \overline{f}(t_2)$. Suppose $s_1 \in \overline{f}(t_1) - \overline{f}(t_2)$ and $s_2 \in \overline{f}(t_1) \cap \overline{f}(t_2)$. Then $t_1 \in f(s_1)$, $t_1 \in f(s_2)$ and $t_2 \in f(s_2)$ but $t_2 \notin f(s_1)$. Hence, $f(s_1) \neq f(s_2)$ but $f(s_1) \cap f(s_2) \neq \varnothing$, which contradicts that $Rng(f)$ is a partition of S_2. Thus, $Rng(\overline{f})$ is a partition of S_1.

3) Immediate from (1), (2), and the definitions. □

3 Method Properties

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 *method properties*.

Definition 6. 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 method properties:

M$_\uparrow$: $|f(s)| = 1$ for all $s \in S_1$.

M$_\downarrow$: $|\overline{f}(s)| = 1$ for all $s \in S_2$.

R$_\uparrow$: If $\langle s_1, t_1, \ell_1 \rangle \in E_1$, then there is some $\langle s_2, t_2, \ell_2 \rangle \in E_2$ such that $R(\ell_1, \ell_2)$.

R$_\downarrow$: If $\langle s_2, t_2, \ell_2 \rangle \in E_2$, then there is some $\langle s_1, t_1, \ell_1 \rangle \in E_1$ such that $R(\ell_1, \ell_2)$.

C$_\uparrow$: If $R(\ell_1, \ell_2)$ and $\langle s_1, t_1, \ell_1 \rangle \in E_1$, then there is some $\langle s_2, t_2, \ell_2 \rangle \in E_2$ such that $s_2 \in f(s_1)$ and $t_2 \in f(t_1)$.

C$_\downarrow$: If $R(\ell_1, \ell_2)$ and $\langle s_2, t_2, \ell_2 \rangle \in E_2$, 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 an $\mathbf{C}_\uparrow$ transformation from $\mathbb{G}_2$ to $\mathbb{G}_1$. It should be noted that these properties are only examples that we have choosen to use in this paper since they are simple, yet powerful. It is naturally possible to define other method properties within our framework.

Example 7. We reconsider Example 3. The function $f_1 : \mathbb{G}_1 \to \mathbb{G}_2$ is $\mathbf{M}_\uparrow$ but is not $\mathbf{M}_\downarrow$ while function $f_3 : \mathbb{G}_1 \to \mathbb{G}_3$ is $\mathbf{M}_\downarrow$ but not $\mathbf{M}_\uparrow$. Define $R = \{a, b\} \times \{c\}$ and note that the transformation $\langle f_1, R\rangle : \mathbb{G}_1 \to \mathbb{G}_2$ has both property $\mathbf{R}_\uparrow$ and $\mathbf{R}_\downarrow$. Furthermore, $\langle f_1, R\rangle$ is $\mathbf{C}_\downarrow$ but not $\mathbf{C}_\uparrow$ (consider the edge from 00 to 01). One may also note that if $R' = \{a, b\} \times \{d\}$ then $\langle f_3, R'\rangle$ is $\mathbf{C}_\uparrow$ but not $\mathbf{C}_\downarrow$

We proceed to show that these properties can describe abstraction as defined by Zilles and Holte (2010) and others (Haslum et al. 2007; Helmert, Haslum, and Hoffmann 2007; Holte et al. 1996). We refer to such abstraction as *strong homomorphism abstraction (SHA)* since the abstraction function f is a strong homomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$.

Definition 8. 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 f be an $\mathbf{M}_\uparrow$ transformation function from $\mathbb{G}_1$ to $\mathbb{G}_2$. Then, f is **SHA** if it satisfies the following two conditions: (1) for every $\langle s_1, t_1, \ell_1\rangle \in E_1$ there is some $\langle s_2, t_2, \ell_2\rangle \in E_2$ such that $s_2 = f(s_1)$ and $t_2 = f(t_1)$, and (2) for every $\langle s_2, t_2, \ell_2\rangle \in E_2$ there is some $\langle s_1, t_1, \ell_1\rangle \in E_1$ such that $s_2 = f(s_1)$ and $t_2 = f(t_1)$.

It does not matter that we use labelled graphs since the labels are not relevant for the definition.

Theorem 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 f be an $\mathbf{M}_\uparrow$ transformation function from $\mathbb{G}_1$ to $\mathbb{G}_2$. Then f is a* **SHA** *if and only if there is a label relation R s.t. $\tau = \langle f, R\rangle$ is an $\mathbf{R}_\updownarrow\mathbf{C}_\updownarrow$ transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$.*

Proof. if: Suppose τ is $\mathbf{R}_\updownarrow\mathbf{C}_\updownarrow$. We first prove condition 1 of **SHA**. Let $e_1 = \langle s_1, t_1, \ell_1\rangle$ be an arbitrary arc in E_1. Since $\mathbf{R}_\uparrow$ holds there is some label ℓ_2 s.t. $R(\ell_1, \ell_2)$. Applying $\mathbf{C}_\uparrow$ gives that there is some arc $\langle s_2, t_2, \ell_2\rangle \in E_2$ s.t. $s_2 \in f(s_1)$ and $t_2 \in f(t_1)$, but τ is $\mathbf{M}_\uparrow$ so $f(s_1) = s_2$ and $f(t_1) = t_2$. It follows that condition 1 holds since e_1 was chosen arbitrarily. We then prove condition 2. Let $e_2 = \langle s_2, t_2, \ell_2\rangle$ be an arbitrary arc in E_2. Since $\mathbf{R}_\downarrow$ holds there is some label ℓ_1 s.t. $R(\ell_1, \ell_2)$. Applying $\mathbf{C}_\downarrow$ gives that there is some arc $\langle s_1, t_1, \ell_1\rangle \in E_1$ s.t. $s_1 \in \overline{f}(s_2)$ and $t_1 \in \overline{f}(t_2)$. Hence, $s_2 \in f(s_1)$ and $t_2 \in f(t_1)$ holds, that is, $s_2 = f(s_1)$ and $t_2 = f(t_1)$ since f is $\mathbf{M}_\uparrow$. It follows that also condition 2 holds (since e_2 was chosen arbitrarily) so f is **SHA** if the transformation τ is $\mathbf{R}_\updownarrow\mathbf{C}_\updownarrow$.

only if: Suppose f is **SHA**. Then it is $\mathbf{M}_\uparrow$ by definition. Define R s.t. $R(\ell_1, \ell_2)$ iff there are arcs $\langle s_1, t_1, \ell_1\rangle \in E_1$ and $\langle s_2, t_2, \ell_2\rangle \in E_2$ s.t. $s_2 \in f(s_1)$ and $t_2 \in f(t_1)$. We first prove that τ is $\mathbf{R}_\uparrow$. Let $e_1 = \langle s_1, t_1, \ell_1\rangle$ be an arbitrary arc in E_1. Condition 1 of **SHA** guarantees that there is some arc $e_2 = \langle s_2, t_2, \ell_2\rangle \in E_2$ s.t. $s_2 = f(s_1)$ and $t_2 = f(t_1)$. The construction of R guarantees that $R(\ell_1, \ell_2)$ so $\mathbf{R}_\uparrow$ holds since e_1 was choosen arbitrarily. Proving that τ is $\mathbf{R}_\downarrow$ is analogous. We now prove that τ is $\mathbf{C}_\uparrow$. Suppose $e_1 = \langle s_1, t_1, \ell_1\rangle \in E_1$ and $R(\ell_1, \ell_2)$. Condition 1 of **SHA** guarantees that there is an arc $\langle s_2, t_2, \ell_2\rangle \in E_2$ s.t. $s_2 = f(s_1)$ and $t_2 = f(t_1)$. Hence, τ is $\mathbf{C}_\uparrow$. Proving that τ is $\mathbf{C}_\downarrow$ is analogous. Since we can always construct R as above, it follows that if f is **SHA**, then there is some R s.t. $\langle f, R\rangle$ is $\mathbf{R}_\updownarrow\mathbf{C}_\updownarrow$. □

4 Instance Properties

We now turn our attention to transformation properties that are related to finding paths in STGs. We refer to these as *instance properties* since they are not necessarily related to the particular method (class of transformations) used but may hold or not on a per instance basis.

In order for an abstraction to be useful, a path in the abstract graph must be useful for finding a path in the original graph. In loose terms, we say that an abstraction is *sound* if every abstract path somehow corresponds to a ground path and that it is *complete* if every ground path has some corresponding abstract path. We will not define these concepts formally, but only note that completeness means that we do not miss any solutions by using abstraction and soundness means that we do not waste time trying to refine something that does not correspond to a solution. We will, however, define and analyse some more specific concepts, but we first need the concept of reachability.

Definition 10. 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 $\mathcal{R}(s) = \{t \in S \mid$ there is a path from s to t in $\mathbb{G}\}$. We extend this s.t. for all $T \subseteq S$, $\mathcal{R}(T) = \cup_{s \in T}\mathcal{R}(s)$.

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.

Definition 11. Let $\mathbb{G}_1 = \langle S_1, E_1\rangle$ and $\mathbb{G}_2 = \langle S_1, E_1\rangle$ be two STGs and let f be a transformation function from $\mathbb{G}_1$ to $\mathbb{G}_2$. Then f can have the following instance properties:

$\mathbf{P}_{\mathbf{k}\downarrow}$**:** If there are $t_0, \ldots, t_k \in S_2$ s.t. $t_i \in \mathcal{R}_2(t_{i-1})$ for all i $(1 \leq i \leq k)$, then there are $s_0, \ldots, s_k \in S_1$ s.t. $s_i \in \overline{f}(t_i)$ for all i $(0 \leq i \leq k)$ and $s_i \in \mathcal{R}_1(s_{i-1})$ for all i $(1 \leq i \leq k)$.

$\mathbf{P_{k\uparrow}}$: If there are $s_0, \ldots, s_k \in S_1$ s.t. $s_i \in \mathcal{R}_1(s_{i-1})$ for all i ($1 \le i \le k$), then there are $t_0, \ldots, t_k \in S_2$ s.t. $t_i \in f(s_i)$ for all i ($0 \le i \le k$) and $t_i \in \mathcal{R}_2(t_{i-1})$ for all i ($1 \le i \le k$).

$\mathbf{P_{T\downarrow}}$: $\mathbf{P}_{1\downarrow}$ holds.

$\mathbf{P_{T\uparrow}}$: $\mathbf{P}_{1\uparrow}$ holds.

$\mathbf{P_{W\downarrow}}$: $\mathbf{P}_{k\downarrow}$ holds for all $k > 0$.

$\mathbf{P_{W\uparrow}}$: $\mathbf{P}_{k\uparrow}$ holds for all $k > 0$.

$\mathbf{P_{\downarrow}}$: If $t \in \mathcal{R}_2(f(s))$, then $\overline{f}(t) \cap \mathcal{R}_1(s) \neq \varnothing$.

$\mathbf{P_{\uparrow}}$: If $t \in \mathcal{R}_1(s)$, then $f(t) \cap \mathcal{R}_2(f(s)) \neq \varnothing$.

$\mathbf{P_{S\downarrow}}$: If $t \in \mathcal{R}_2(f(s))$, then $\overline{f}(t) \subseteq \mathcal{R}_1(s)$.

$\mathbf{P_{S\uparrow}}$: If $t \in \mathcal{R}_1(s)$, then $f(t) \subseteq \mathcal{R}_2(f(s))$.

The properties marked with a downward arrow can be viewed as different degrees of soundness and properties marked with an upward arrow as different degrees of completeness. We continue by briefly describing the soundness properties and we note that the completeness properties can be described analogously by using the symmetries inherent in transformations. Consider a path $t_0, t_1, \ldots, t_k$ in the abstract graph, for some $k > 0$. If property $\mathbf{P_{k\downarrow}}$ holds, then there are states $s_0, s_1, \ldots, s_k$ in the original graph such that there is a path from s_0 to s_k passing through all of $s_1, \ldots, s_{k-1}$ in order. Consider the example in Figure 2. This transformation function f satisfies $\mathbf{P_{1\downarrow}}$ since both single-arc paths t_0, t_1 and t_1, t_2 in $\mathbb{G}_2$ have corresponding paths in $\mathbb{G}_1$. However, f does not satisfy $\mathbf{P_{2\downarrow}}$ since the path t_0, t_1, t_2 does not have a corresponding path in $\mathbb{G}_1$; we can go from $\overline{f}(t_0)$ to $\overline{f}(t_1)$ and from $\overline{f}(t_1)$ to $\overline{f}(t_2)$ but we cannot go all the way from $\overline{f}(t_0)$ to $\overline{f}(t_2)$.

Immediately by definition, $\mathbf{P_{W\downarrow}}$ (where **W** stands for 'weak') holds if $\mathbf{P_{k\downarrow}}$ holds for all k (i.e. there is no upper bound on the length of the sequence), and property $\mathbf{P_{T\downarrow}}$ (where **T** stands for 'trivial') implies that if there is a path between two states in the abstract graph, then there is a path between two corresponding states in the original graph. Property $\mathbf{P}_{\downarrow}$ implies that for any state s in the original graph and any state t in the abstract graph, if there is a path from $f(s)$ to t in the abstract graph, then there is a path from s to *some* $u \in \overline{f}(t)$ in the original graph. Property $\mathbf{P_{S\downarrow}}$ (where **S** stands for 'strong') is defined similarly: if there is a path from $f(s)$ to t in the abstract graph, then there is a path from s to *all* $u \in \overline{f}(t)$ in the original graph.

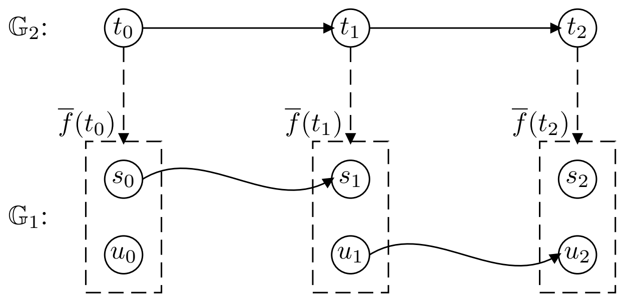

Figure 2: A transformation that is $\mathbf{P_{1\downarrow}}$ but not $\mathbf{P_{2\downarrow}}$.

We will link many of these properties to different computational phenomena in this and the next section. This link will typically be very natural; for instance, the property $\mathbf{P_{W\downarrow}}$ is associated with *weak* downward state refinements. We note that the following implications hold:

$$\mathbf{P_{S\downarrow}} \Rightarrow \mathbf{P_{\downarrow}} \Rightarrow \mathbf{P_{W\downarrow}} \Rightarrow \mathbf{P_{T\downarrow}}$$

That $\mathbf{P_{S\downarrow}} \Rightarrow \mathbf{P_{\downarrow}}$ and $\mathbf{P_{W\downarrow}} \Rightarrow \mathbf{P_{T\downarrow}}$ follows directly from the definitions above, while the implication $\mathbf{P_{\downarrow}} \Rightarrow \mathbf{P_{W\downarrow}}$ follows from Theorem 16 (which appears in the next section). We also see that

$$\mathbf{P_{T\downarrow}} \not\Rightarrow \mathbf{P_{W\downarrow}} \not\Rightarrow \mathbf{P_{\downarrow}} \not\Rightarrow \mathbf{P_{S\downarrow}}$$

The fact that $\mathbf{P_{W\downarrow}} \not\Rightarrow \mathbf{P_{\downarrow}}$ follows from Theorem 16 while the other two non-implications can be demonstrated by straightforward counterexamples.

We exemplify by returning to Zilles and Holte's approach. They defined a property that they refer to as the *downward path preserving (**DPP**)* property. Given a state s in the original graph, we define its corresponding set of spurious states $\mathcal{S}(s)$ to be the set $\mathcal{R}_2(f(s)) \setminus f(\mathcal{R}_1(s))$. The intention behind the **DPP** property is to avoid spurious states, i.e. guarantee that $\mathcal{S}(s) = \varnothing$ for all s. They introduce two criteria on SHA abstraction that together define **DPP**, and we generalise this idea as follows.

Definition 12. Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_1, E_1 \rangle$ be two STGs and let f be a transformation function from $\mathbb{G}_1$ to $\mathbb{G}_2$. Then f can be classified as:

SH1: $\mathcal{R}_2(f(s)) \subseteq f(\mathcal{R}_1(s))$ for all $s \in S_1$.

SH2: $f(\mathcal{R}_1(s)) \subseteq \mathcal{R}_2(f(s))$ for all $s \in S_1$.

DPP: Both **SH1** and **SH2** hold.

Although this definition looks more or less identical to theirs, it is a generalisation since it does not require f to be SHA. In our terminology, **SH1** is a soundness condition and **SH2** a completeness condition. We see that condition **SH1** holds if there are no spurious states, that is, **SH1** guarantees that if we can reach an abstract state, then we can also reach some corresponding ground state. Additionally, Zilles and Holte noted that **SH2** is inherent in every SHA abstraction but this is not necessarily true in general abstractions. We note that both these conditions are captured by the instance properties $\mathbf{P}_{\downarrow}$ and $\mathbf{P}_{\uparrow}$.

Lemma 13. *SH1 is equivalent to $\mathbf{P}_{\downarrow}$ and SH2 is equivalent to $\mathbf{P}_{\uparrow}$.*

Proof. We prove that **SH1** is equivalent to $\mathbf{P}_{\downarrow}$.
$\mathcal{R}_2(f(s)) \subseteq f(\mathcal{R}_1(s))$ iff $t \in \mathcal{R}_2(f(s)) \Rightarrow t \in f(\mathcal{R}_1(s))$
iff $t \in \mathcal{R}_2(f(s)) \Rightarrow \exists u.(u \in \mathcal{R}_1(s)$ and $t \in f(u))$
iff $t \in \mathcal{R}_2(f(s)) \Rightarrow \exists u.(u \in \mathcal{R}_1(s)$ and $u \in \overline{f}(t))$
iff $t \in \mathcal{R}_2(f(s)) \Rightarrow \overline{f}(t) \cap \mathcal{R}_1(s) \neq \varnothing$.
Proving that **SH2** is equivalent to $\mathbf{P}_{\uparrow}$ is analogous. □

5 Path Refinement

If we want to find a path in an STG by abstraction, then we must transform this STG into an abstract STG and find a path in the latter. We must then somehow refine this abstract path into a path in the original STG. Preferably, we want to do this without backtracking to the abstract level. We define three different kinds of path refinements that achieves this, with varying degrees of practical usefulness.

Definition 14. 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 f be a transformation function from $\mathbb{G}_1$ to $\mathbb{G}_2$. Let $\sigma = t_0, t_1, \ldots, t_k$ be an arbitrary path in $\mathbb{G}_2$. Then:

1) σ is *trivially downward state refinable* if there are two states $s_0 \in \overline{f}(t_0)$ and $s_\ell \in \overline{f}(t_k)$ s.t. there is a path in $\mathbb{G}_1$ from s_0 to s_ℓ.

2) σ is *weakly downward state refinable* if there is a sequence $s_0, s_1, \ldots, s_k$ of states in S_1 such that $s_i \in \overline{f}(t_i)$ for all i s.t. $0 \leq i \leq k$ and there is a path from s_{i-1} to s_i in $\mathbb{G}_1$ for all i ($1 \leq i \leq k$).

3) σ is *strongly downward state refinable* if for every i s.t. $1 \leq i \leq k$, there is a path from s_{i-1} to s_i in $\mathbb{G}_1$ for all $s_{i-1} \in \overline{f}(t_{i-1})$ and all $s_i \in \overline{f}(t_i)$.

Trivial path refinement only requires that if there is a path between two states in the abstract graph, then there is a path between two corresponding states in the original graph. The two paths need not have any other connection at all. The other two refinements tie the two paths to each other in such a way that the states along the abstract path are useful for finding the ground path.

In Figure 3 we provide two algorithms that, under certain conditions, both avoid backtracking between levels. The **choose** statements are non-deterministic, that is, an actual implementation would use search with the **choose** statements as backtrack points. Algorithm TPath implements trivial path refinement. It first finds an abstract path. If this succeeds, then it calls Refine to find a ground path between the first and last states. Under the assumption that all paths are trivially refinable, there is no need for Refine to backtrack up to TPath again. Algorithm WSPath implements weak and strong path refinement. It first finds an abstract path. If this succeeds, then it passes the whole path to Refine so the states along the path can be used as subgoals. If all paths are weakly refinable, then there is no need for Refine to backtrack up to WSPath again. If all paths are strongly refinable, then there is not even any need to backtrack to the **choose** points within Refine. The different degrees of refinements are captured by instance properties as follows.

Theorem 15. *Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_1, E_1 \rangle$ be two STGs and let $\tau = \langle f, R \rangle$ be a transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$. Then:*

1) Every path in $\mathbb{G}_2$ is trivially downward state refinable iff τ is $\mathbf{P}_{\mathbf{T}\downarrow}$.

2) Every path in $\mathbb{G}_2$ is weakly downward state refinable iff τ is $\mathbf{P}_{\mathbf{W}\downarrow}$.

3) Every path in $\mathbb{G}_2$ is strongly downward state refinable iff τ is $\mathbf{P}_{\mathbf{S}\downarrow}$.

Proof. 1) and 2) are straightforward from the definitions.

3) *if:* Suppose τ is $\mathbf{P}_{\mathbf{S}\downarrow}$. Induction over the path length.

Base case: For every path t of length one in $\mathbb{G}_2$ every $s \in \overline{f}(t)$ is a path in $\mathbb{G}_1$.

Induction: Suppose every path of length k in $\mathbb{G}_2$ is strongly downwards refinable, for some $k > 0$. Let $\sigma = t_0, t_1, \ldots, t_k$ be an arbitrary path in $\mathbb{G}_2$. It follows from the induction hypothesis that $t_1, \ldots, t_k$ is strongly downwards refinable, so

```
function TPath($s$,$t$)
    choose $\sigma = t_0, t_1, \ldots, t_k$ s.t. $t_0 \in f(s)$, $t_k \in f(t)$
        and $\sigma$ is a path from $t_0$ to $t_k$
    if no such $\sigma$ then fail
    else return Refine($t_0$,$t_k$)

function WSPath($s$,$t$)
    choose $\sigma = t_0, t_1, \ldots, t_k$ s.t. $t_0 \in f(s)$, $t_k \in f(t)$
        and $\sigma$ is a path from $t_0$ to $t_k$
    if no such $\sigma$ then fail
    else return Refine($\sigma$)

function Refine($t_0, t_1, \ldots, t_k$)
    choose $s_0 \in \overline{f}(t_0)$
    if $k = 0$ then return $s_0$
    else
        $\sigma_2 = \text{Refine}(t_1, \ldots, t_k)$
        $s_1 = \text{first}(\sigma_2)$
        choose path $\sigma_1$ from $s_0$ to $s_1$
        if no such $\sigma_1$ then fail
        else return $\sigma_1; \sigma_2$
```

Figure 3: Algorithms for path refinement.

it remains to prove that there is a path in $\mathbb{G}_1$ from every $s_0 \in \overline{f}(t_0)$ to every $s_1 \in \overline{f}(t_1)$. Let s_0 arbitrary in $\overline{f}(t_0)$. Hence, $t_0 \in f(s_0)$ and $t_1 \in \mathcal{R}_2(t_0)$ so $t_1 \in \mathcal{R}_2(f(s_0))$. It follows from $\mathbf{P}_{\mathbf{S}\downarrow}$ that $\overline{f}(t_1) \subseteq \mathcal{R}_1(s_0)$ which means there must be a path in $\mathbb{G}_1$ from s_0 to every $s_1 \in \overline{f}(t_1)$. The result follows since both σ and s_0 were chosen arbitrarily.

only if: Suppose all paths in $\mathbb{G}_2$ are strongly downwards refinable. Suppose $s_0 \in S_1$ and $t_1 \in S_2$ are two states s.t. $t_1 \in \mathcal{R}_2(f(s_0))$. Then, there is some state $t_0 \in S_2$ s.t. $t_0 \in f(s_0)$ and $t_1 \in \mathcal{R}_2(t_0)$. Hence, there is a path in $\mathbb{G}_2$ from t_0 to t_1. By assumption this path is strongly downward refinable so there must be a path from every $u_0 \in \overline{f}(t_0)$ to every $u_1 \in \overline{f}(t_1)$. It follows that $\overline{f}(t_1) \subseteq \mathcal{R}_1(s_0)$ and, thus, that $\mathbf{P}_{\mathbf{S}\downarrow}$ holds. $\square$

We can now clarify the relation between $\mathbf{P}_\downarrow$ and $\mathbf{P}_{\mathbf{W}\downarrow}$.

Theorem 16. *Let $\mathbb{G}_1 = \langle S_1, E_1 \rangle$ and $\mathbb{G}_2 = \langle S_1, E_1 \rangle$ be two STGs and let $\tau = \langle f, R \rangle$ be a transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$. Then:*

1) Every path in $\mathbb{G}_2$ is weakly downward state refinable if τ is $\mathbf{P}_\downarrow$.

2) That every path in $\mathbb{G}_2$ is weakly downward state refinable does not imply that τ is $\mathbf{P}_\downarrow$.

3) $\mathbf{P}_\downarrow \Rightarrow \mathbf{P}_{\mathbf{W}\downarrow}$ but $\mathbf{P}_{\mathbf{W}\downarrow} \not\Rightarrow \mathbf{P}_\downarrow$.

Proof. 1) Assume $\mathbf{P}_\downarrow$ holds and assume there is a path $t_0, ..., t_k$ in $\mathbb{G}_2$. For every i, $0 \leq i < k$, it holds that $t_{i+1} \in \mathcal{R}_2(t_i)$. Property $\mathbf{P}_\downarrow$ implies that for arbitrariy $s \in \overline{f}(t_i)$, there is some s' such that $s' \in \overline{f}(t_{i+1})$ and $s' \in \mathcal{R}_1(s_i)$. Now, arbitrarily choose $s_0 \in \overline{f}(t_0)$. If we for each i let $s = s_i$ and choose s_{i+1} to be the corresponding s' above, then $s_0, \ldots, s_k$ satisfies the conditions for $\mathbf{P}_{\mathbf{W}\downarrow}$.

2) Suppose $S_1 = \{s_0^a, s_0^b, s_1\}$, $S_2 = \{t_0, t_1\}$, $E_1 = \{\langle s_0^a, s_1, \ell_1\rangle\}$ and $E_2 = \{\langle t_0, t_1, \ell_2\rangle\}$. Also define a transformation function f s.t. $f(s_0^a) = f(s_0^b) = t_0$ and $f(s_1) = t_1$. Then, there are three paths in $\mathbb{G}_2$: two atomary paths t_0 and t_1 and the path t_0, t_1. These are all weakly refinable to paths in $\mathbb{G}_1$. That is, all paths in $\mathbb{G}_2$ are weakly refinable. Obviously $t_0 \in f(s_0^b)$ and $t_1 \in \mathcal{R}_2(t_0)$, so $t_1 \in \mathcal{R}_2(f(s_0^b))$. However, $\overline{f}(t_1) = \{s_1\}$ but $\mathcal{R}_1(s_0^b) = \{s_0^b\}$ so $\mathbf{P}_\downarrow$ does not hold for this example.

3) Combine 1) and 2) with Theorem 15. $\square$

One consequence of this result is that if an abstraction avoids spurious states (for instance, by satisfying the **DPP** or the $\mathbf{P}_\downarrow$ condition), then the WSPath algorithm can solve the problem without doing any backtracking to the abstract level. Avoiding spurious states is, however, not a necessary condition for avoiding backtracking between levels.

6 Planning

We will now consider state spaces that are induced by variables. A state is then defined as a vector of values for these variables. We will, however, do this a bit differently and use a state concept based on sets of variable-value pairs. While this make the basic definitions slightly more complicated, it will simplify the forthcoming definitions and proofs.

Definition 17. A *variable set* V is a set of objects called *variables*. A *domain function* D for V is a function that maps every variable $v \in V$ to a corresponding *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 $V \cdot D = \cup_{v \in V}(\{v\} \times D_v)$ 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 at most once in s,

2) *total* if each $v \in V$ occurs exactly once 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) $\mathcal{T}(S) = \{s \subseteq S \mid s \text{ is total }\}$.

For arbitrary states $s, t \in \mathcal{C}(V \cdot D)$, variable set $U \subseteq V$ and variable $v \in V$: $V(s) = \{v \mid (v = x) \in s\}$, $s[U] = s \cap (U \cdot D)$, $s[v] = s[\{v\}]$, $s \ltimes t = s[V - V(t)] \cup t$, $s^{=c} = \{(v = c) \in s\}$ and $U^{:=c} = \{(v = c) \mid v \in U\}$.

The operator $\ltimes$ is typically used for updating a state s with the effects of an action a; this will be made formally clear in Definition 19. Note that $\mathcal{T}(V \cdot D)$ is the set of all total states over V and D and $\mathcal{C}(V \cdot D)$ is the set of all consistent states. Unless otherwise specified, states will be assumed total and we will usually write state rather than total state. The following observations will be tacitly used henceforth.

Proposition 18. *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 disjunct $V_1, V_2 \subseteq V$.

5) $s[U] \star t[U] = (s \star t)[U]$, where $\star$ is $\cup$ or $\ltimes$.

6) If s is total, then $s \ltimes t$ is total.

We define MSTRIPS as a variant of STRIPS that uses multivalued variables instead of propositional atoms, as follows.

Definition 19. An MSTRIPS *instance* is a tuple $\boldsymbol{p} = \langle V, D, A, I, G\rangle$ where V is a variable set, D is a domain function for V, A is a set of *actions*, $I \in \mathcal{T}(V \cdot D)$ is the *initial state* and $G \in \mathcal{C}(V \cdot D)$ is the *goal*. 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 $\mathbb{G}(\boldsymbol{p}) = \langle S, E\rangle$ for $\boldsymbol{p}$ is defined s.t. 1) $S = \mathcal{T}(V \cdot D)$ and 2) $E = \{\langle s, t, a\rangle \mid a \in A, \mathrm{pre}(a) \subseteq s \text{ and } t = s \ltimes \mathrm{post}(a)\}$. Let $\omega = a_1, \ldots, a_k$ be a sequence of actions in A. Then, ω is a *plan* for $\boldsymbol{p}$ if there is a path $s_0, s_1, \ldots, s_k$ in $\mathbb{G}(\boldsymbol{p})$ s.t. $I = s_0$ and $G \subseteq s_k$.

It is clear that many propositional planning formalisms over finite domains, such as STRIPS and SAS$^+$, can be modelled within MSTRIPS.

7 Abstraction in Planning

The goal of this section is to model five different abstraction-like methods 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. One may, for instance, note that labels will exclusively be used for keeping track of action names in all five examples. This coherent way of defining the methods makes it possible to systematically study their intrinsic method properties; something that will be carried out in the next section. In order to simplify the notation, we extend the transformation concept to planning instances such that τ is a transformation from $\boldsymbol{p}_1$ to $\boldsymbol{p}_2$ if it is a transformation from $\mathbb{G}(\boldsymbol{p}_1)$ to $\mathbb{G}(\boldsymbol{p}_2)$.

ABSTRIPS Style Abstraction. ABSTRIPS (Sacerdoti 1974) is a version of the STRIPS planner using state abstraction. The idea 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 idea has been commonly used, for instance, in the ABTWEAK planner (Bacchus and Yang 1994). We refer to Sacerdotis original idea as ABSTRIPS I (**ABI**), which is a transformation as follows.

Definition 20. (**ABI**) Let $\boldsymbol{p}_1 = \langle V_1, D_1, A_1, I_1, G_1\rangle$ and $\boldsymbol{p}_2 = \langle V_2, D_2, A_2, I_2, G_2\rangle$ be two MSTRIPS instances and let $\tau = \langle f, R\rangle$ be a transformation from $\mathbb{G}(\boldsymbol{p}_1) = \langle S_1, E_1\rangle$ to $\mathbb{G}(\boldsymbol{p}_2) = \langle S_2, E_2\rangle$. Then, τ is an **ABI** transformation if there is a set of critical variables $V_C \subseteq V_1$ and a bijection $g : A_1 \rightarrow A_2$ s.t. the following holds:

1) $V_1 = V_2$, $D_1 = D_2$, $I_1 = I_2$, $G_1 = G_2$.

2) $g(a) = \langle \mathrm{pre}(a)[V_C], \mathrm{post}(a)\rangle$ for all $a \in A_1$.

3) $A_2 = \{g(a) \mid a \in A_1\}$.

4) $f(s) = \{t \in S_2 \mid s[V_C] = t[V_C]\}$ for all $s \in S_1$.

5) $R = \{\langle a, g(a)\rangle \mid a \in A_1\}$.

Variations on this idea occur in the literature. For instance, Knoblock (1994) removes the non-critical atoms everywhere, not only in preconditions. We refer to his variant as ABSTRIPS II (**ABII**).

Definition 21. (**ABII**) Let $\boldsymbol{p}_1 = \langle V_1, D_1, A_1, I_1, G_1\rangle$ and $\boldsymbol{p}_2 = \langle V_2, D_2, A_2, I_2, G_2\rangle$ be two MSTRIPS instances and let $\tau = \langle f, R\rangle$ be a transformation from $\mathbb{G}(\boldsymbol{p}_1) = \langle S_1, E_1\rangle$ to $\mathbb{G}(\boldsymbol{p}_2) = \langle S_2, E_2\rangle$. Then, τ is an **ABII** transformation if there is a set of critical variables $V_C \subseteq V_1$ and a bijection $g : A_1 \to A_2$ s.t. the following holds:

1) $V_2 = V_C$, $I_2 = I_1[V_C]$, $G_2 = G_1[V_C]$.
2) $g(a) = \langle \mathrm{pre}(a)[V_C], \mathrm{post}(a)[V_C]\rangle$ for all $a \in A_1$.
3) $A_2 = \{g(a) \mid a \in A_1\}$.
4) $f(s) = s[V_C]$ for all $s \in S_1$.
5) $R = \{\langle a, g(a)\rangle \mid a \in A\}$.

Removing Redundant Actions. As a response to the belief that it is good for a planner to have many choices, Haslum and Jonsson (2000) showed that it may be more efficient to have as few choices as possible. They proposed removing some, or all, redundant actions. While the authors did not think of this as an abstraction, it is quite reasonable to do so: we abstract away redundant information by removing redundant actions. 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, **RRA**a and **RRA**b, differing in condition 3 below.

Definition 22. (**RRA**) Let $\boldsymbol{p}_1 = \langle V_1, D_1, A_1, I_1, G_1\rangle$ and $\boldsymbol{p}_2 = \langle V_2, D_2, A_2, I_2, G_2\rangle$ be two MSTRIPS instances and let $\tau = \langle f, R\rangle$ be a transformation from $\mathbb{G}(\boldsymbol{p}_1) = \langle S_1, E_1\rangle$ to $\mathbb{G}(\boldsymbol{p}_2) = \langle S_2, E_2\rangle$. Then, τ is an **RRA** transformation if the following holds:

1) $V_1 = V_2$, $D_1 = D_2$, $I_1 = I_2$, $G_1 = G_2$.
2) $A_2 \subseteq A_1$.
3a) $\{\langle s,t\rangle \mid \langle s,t,\ell\rangle \in E_1\} = \{\langle s,t\rangle \mid \langle s,t,\ell\rangle \in E_2\}$.
3b) $\{\langle s,t\rangle \mid \langle s,t,\ell\rangle \in E_1\} = \{\langle s,t\rangle \mid \langle s,t,\ell\rangle \in E_2\}^+$.
4) f is the identity function.
5) $R = \{\langle a, a\rangle \mid a \in A_2\}$.

Variant 3a (**RRA**a) says that if an action a 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 (**RRA**b), on the other hand, only requires that there is still a path from s to t (the '+' denotes transitive closure). While 3a preserves the length of solutions, 3b does not.

Ignoring Delete Lists. The idea of removing the negative postconditions from all actions in STRIPS (Bonet, Loerincs, and Geffner 1997; McDermott 1996) is known as *ignoring delete lists*. This means that false atoms can be set to true, but not vice versa. The method is commonly used as an abstraction for computing the h^+ heuristic in planning (Hoffmann 2005). We refer to this method as **IDL**.

Definition 23. (**IDL**) Let $\boldsymbol{p}_1 = \langle V_1, D_1, A_1, I_1, G_1\rangle$ and $\boldsymbol{p}_2 = \langle V_2, D_2, A_2, I_2, G_2\rangle$ be two MSTRIPS instances with binary variables and let $\tau = \langle f, R\rangle$ be a transformation from $\mathbb{G}(\boldsymbol{p}_1)$ to $\mathbb{G}(\boldsymbol{p}_2)$. Then, τ is an **IDL** transformation if there is a bijection $g : A_1 \to A_2$ s.t. the following holds:

1) $V_1 = V_2$, $D_1 = D_2$, $I_1 = I_2$ and $G_1 = G_2$.
2) $g(a) = \langle \mathrm{pre}(a), \mathrm{post}(a)^{=1}\rangle$ for all $a \in A_1$.
3) $A_2 = \{g(a) \mid a \in A_1\}$.
4) f is the identity function.
5) $R = \{\langle a, g(a)\rangle \mid a \in A_1\}$.

Explicit Landmark Abstraction. A landmark is a necessary subgoal for a plan. Sets of landmarks are usually added as separate information to planners as extra guidance for how to solve a particular instance (Hoffmann, Porteous, and Sebastia 2004). However, Domshlak, Katz, and Lefler (2010) suggested to combine abstraction with landmarks by encoding the landmark set explicitly in the instance. We refer to this method as *Explicit landmark abstraction (**ELA**)*.

Definition 24. (**ELA**) Let $\boldsymbol{p}_1 = \langle V_1, D_1, A_1, I_1, G_1\rangle$ and $\boldsymbol{p}_2 = \langle V_2, D_2, A_2, I_2, G_2\rangle$ be two MSTRIPS instances and let $\tau = \langle f, R\rangle$ be a transformation from $\mathbb{G}(\boldsymbol{p}_1) = \langle S_1, E_1\rangle$ to $\mathbb{G}(\boldsymbol{p}_2) = \langle S_2, E_2\rangle$. Furthermore, let $M \subseteq V_1 \cdot D_1$ be a set of *landmarks*. Define the variable set $V_M = \{v_{u,x} \mid (u = x) \in M\}$ with domain function $D_M : V_M \to \{0, 1\}$. For each $a \in A_1$, define $\mathrm{post}_M(a) = \{(v_{u,x} = 1) \mid (u = x) \in \mathrm{post}(a) \cap M\}$. Then, τ is an **ELA** transformation if there is a bijection $g : A_1 \to A_2$ s.t. the following holds:

1) $V_2 = V_1 \cup V_M$, $D_2 = D_1 \cup D_M$, $I_2 = I_1 \cup V_M{}^{:=0}$ and $G_2 = G_1 \cup V_M{}^{:=1}$.
2) $g(a) = \langle \mathrm{pre}(a), \mathrm{post}(a) \cup \mathrm{post}_M(a)\rangle$ for all $a \in A_1$.
3) $A_2 = \{g(a) \mid a \in A_1\}$.
4) $f(s) = \{t \in S_2 \mid s \subseteq t\}$ for all $s \in S_1$.
5) $R = \{\langle a, g(a)\rangle \mid a \in A_1\}$.

8 Analysis of Planning Abstraction

We can now analyse the methods presented in the previous section with respect to their method properties. From this, we will also get a number of results concerning their computational properties; we will see that certain combinations of method properties imply certain instance properties. The reader should keep the following in mind.

Proposition 25. *Let $\boldsymbol{p}_1 = \langle V_1, D_1, A_1, I_1, G_1\rangle$, $\boldsymbol{p}_2 = \langle V_2, D_2, A_2, I_2, G_2\rangle$ be two* MSTRIPS *instances and let $V_C \subseteq V$ and $M \subseteq V_1 \cdot D_1$. Define $\overline{V_C} = V_1 - V_C$ and assume $\tau = \langle f, R\rangle$ to be a transformation from $\mathbb{G}(\boldsymbol{p}_1) = \langle S_1, E_1\rangle$ to $\mathbb{G}(\boldsymbol{p}_2) = \langle S_2, E_2\rangle$. Then:*

*1) If τ is **ABI**, then for all $s \in S_2$:*
a) $\overline{f}(s) = \{s[V_C] \cup t \mid t \in \mathcal{T}(\overline{V_C} \cdot D)\}$.
b) $\overline{f}(s) = \{t \in S_1 \mid s[V_C] = t[V_C]\}$.
*2) If τ is **ABII**, then for all $s \in S_2$:*
a) $\overline{f}(s) = \{s \cup t \mid t \in \mathcal{T}(\overline{V_C} \cdot D)\}$.
b) $\overline{f}(s) = \{t \in S_1 \mid s[V_C] = t[V_C]\}$.
*3) If τ is **ELA**, then for all $s \in S_1$):*
a) $f(s) = \{s \cup t \mid t \in \mathcal{T}(V_M \cdot D_M)\}$.
b) $\overline{f}(s) = s[V_1]$.

The following theorem presents the method properties inherent in the methods in the previous section. The results are summarised in column 2 of Table 1.

Theorem 26. *Let $\boldsymbol{p}_1 = \langle V_1, D_1, A_1, I_1, G_1\rangle$ and $\boldsymbol{p}_2 = \langle V_2, D_2, A_2, I_2, G_2\rangle$ be two* MSTRIPS *instances and let $\tau = \langle f, R\rangle$ be a transformation from $\mathbb{G}(\boldsymbol{p}_1) = \langle S_1, E_1\rangle$ to $\mathbb{G}(\boldsymbol{p}_2) = \langle S_2, E_2\rangle$. Then if τ is:*

*1) **ABI**, then it is $\boldsymbol{R}_{\updownarrow}\boldsymbol{C}_{\updownarrow}$ but not necessarily $\boldsymbol{M}_{\uparrow}$ or $\boldsymbol{M}_{\downarrow}$.*
*2) **ABII**, then it is $\boldsymbol{M}_{\uparrow}\boldsymbol{R}_{\updownarrow}\boldsymbol{C}_{\updownarrow}$ but not necessarily $\boldsymbol{M}_{\downarrow}$.*

*3) **RRA**a or **RRA**b, then it is $\mathbf{M}_{\updownarrow}\mathbf{R}_{\downarrow}\mathbf{C}_{\updownarrow}$ but not necessarily $\mathbf{R}_{\uparrow}$.*
*4) **IDL**, then it is $\mathbf{M}_{\updownarrow}\mathbf{R}_{\updownarrow}$ but not necessarily $\mathbf{C}_{\uparrow}$ or $\mathbf{C}_{\downarrow}$.*
*5) **ELA**, then it is $\mathbf{M}_{\downarrow}\mathbf{R}_{\updownarrow}\mathbf{C}_{\updownarrow}$ but not necessarily $\mathbf{M}_{\uparrow}$.*

Proof. We will tacitly make frequent use of Propositions 18 and 25 in the following proof.

ABI: Let $V_C \subseteq V_1$ denote the set of critical variables and $\overline{V_C} = V_1 - V_C$. Suppose $\langle s_1, t_1, a_1\rangle \in E_1$ and note that $\text{pre}(a_1) \subseteq s_1$. Let $a_2 = g(a_1)$. Then, $R(a_1, a_2)$ by definition and $\text{pre}(a_2) = \text{pre}(a_1)[V_C] \subseteq \text{pre}(a_1) \subseteq s_1$ so $\langle s_1, s_1 \ltimes \text{post}(a_2), a_2\rangle \in E_2$ and it follows that τ is $\mathbf{R}_{\uparrow}$.

Suppose instead that $\langle s_2, t_2, a_2\rangle \in E_2$ and note that $\text{pre}(a_2) \subseteq s_2$. Let $a_1 = g^{-1}(a_2)$. Then, $R(a_1, a_2)$ by definition, $\text{pre}(a_1)[V_C] = \text{pre}(a_2)$ and $\text{pre}(a_1)[\overline{V_C}] \in \mathcal{C}(\overline{V_C}\cdot D)$. There must thus be some state $s_1 \in \{s_2[V_C] \cup t \mid t \in \mathcal{T}(\overline{V_C}\cdot D)\} = \overline{f}(s_2)$ s.t. $\text{pre}(a_1) \subseteq s_1$. Hence, $\langle s_1, s_1 \ltimes \text{post}(a_1), a_1\rangle \in E_1$ τ is $\mathbf{R}_{\downarrow}$.

We now turn our attention to properties $\mathbf{C}_{\uparrow}$ and $\mathbf{C}_{\downarrow}$. Suppose $\langle s_1, t_1, a_1\rangle \in E_1$ and $R(a_1, a_2)$. Then, $\text{pre}(a_1) \subseteq s_1$ and $t_1 = s_1 \ltimes \text{post}(a_1)$. Furthermore $a_2 = g(a_1)$ so $\text{pre}(a_2) = \text{pre}(a_1)[V_C]$ and $\text{post}(a_2) = \text{post}(a_1)$. Thus, $\text{pre}(a_2) \subseteq s_1$ and $s_1 \ltimes \text{post}(a_2) = s_1 \ltimes \text{post}(a_1) = t_1$. Consequently, $\langle s_1, t_1, a_2\rangle \in E_2$ and it follows that τ is $\mathbf{C}_{\uparrow}$ since $s_1 \in f(s_1)$ and $t_1 \in f(t_1)$.

Suppose instead that $\langle s_2, t_2, a_2\rangle \in E_2$ and $R(a_1, a_2)$. Then, $\text{pre}(a_2) \subseteq s_2$, $t_2 = s_2 \ltimes \text{post}(a_2)$ and $a_1 = g^{-1}(a_2)$. With the same argument as in the $\mathbf{R}_{\downarrow}$ case, there must be some $s_1 \in \overline{f}(s_2)$ s.t. $\text{pre}(a_1) \subseteq s_1$. Let $t_1 = s_1 \ltimes \text{post}(a_1)$ and observe that $t_1[V_C] = s_1[V_C] \ltimes \text{post}(a_1)[V_C] = s_2[V_C] \ltimes \text{post}(a_1)[V_C] = s_2[V_C] \ltimes \text{post}(a_2)[V_C] = t_2[V_C]$ so $t_1 \in \overline{f}(t_2)$. Hence, $\langle s_1, t_1, a_1\rangle \in E_1$ and it follows that τ is also $\mathbf{C}_{\downarrow}$.

Finally, we show that **ABI** is not necessarily $\mathbf{M}_{\uparrow}$ or $\mathbf{M}_{\downarrow}$ by a counterexample. Let $V_1 = \{u, v\}$, $D_u = D_v = \{0, 1\}$ and $V_C = \{u\}$. Let $s_0, s_1 \in S_1$ s.t. $s_0 = \{(u = 0), (v = 0)\}$ and $s_1 = \{(u = 0), (v = 1)\}$. Let τ' be an arbitrary **ABI** transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$. Since $S_2 = S_1$ we also have $s_0, s_1 \in S_2$. Then, $s_0[V_C] = s_1[V_C]$ so $f(s_0) = \{s_0, s_1\}$ and it follows that τ' is not $\mathbf{M}_{\uparrow}$. Since $f(s_0) = f(s_1)$ we also get $\overline{f}(s_0) = \{s_0, s_1\}$ so τ' is not $\mathbf{M}_{\downarrow}$ either.

ABII: Let $V_C \subseteq V_1$ denote the set of critical variables and $\overline{V_C} = V_1 - V_C$. Using the same example as in the **ABI** case shows that **ABII** is not necessarily $\mathbf{M}_{\downarrow}$. However, $f(s) = s[V_C]$ for every state s so τ is $\mathbf{M}_{\uparrow}$.

Suppose $\langle s_1, t_1, a_1\rangle \in E_1$ and note that $\text{pre}(a_1) \subseteq s_1$. Let $a_2 = g(a_1)$ and $R(a_1, a_2)$ holds by definition. Let $s_2 = s_1[V_C]$. Then, $\text{pre}(a_2) = \text{pre}(a_1)[V_C] \subseteq s_1[V_C] = s_2$, so $\langle s_2, s_2 \ltimes \text{post}(a_2), a_2\rangle \in E_2$. It follows that $\mathbf{R}_{\uparrow}$ holds. Property $\mathbf{R}_{\downarrow}$ holds by the same argument as for **ABI**.

Suppose $\langle s_1, t_1, a_1\rangle \in E_1$ and $R(a_1, a_2)$. Then, $\text{pre}(a_1) \subseteq s_1$, $t_1 = s_1 \ltimes \text{post}(a_1)$, and $a_2 = g(a_1)$. By letting $s_2 = f(s_1) = s_1[V_C]$, it follows that $\text{pre}(a_2) = \text{pre}(a_1)[V_C] \subseteq s_1[V_C] = s_2$. Let $t_2 = s_2 \ltimes \text{post}(a_2) = s_1[V_C] \ltimes \text{post}(a_1)[V_C] = t_1[V_C] = f(t_1)$. Then, $\langle s_2, t_2, a_2\rangle = \langle f(s_1), f(t_1), a_2\rangle \in E_2$ so τ is $\mathbf{C}_{\uparrow}$.
Suppose $\langle s_2, t_2, a_2\rangle \in E_2$ and $R(a_1, a_2)$. Then, $\text{pre}(a_2) \subseteq s_2$, $t_2 = s_2 \ltimes \text{post}(a_2)$, and $a_1 = g^{-1}(a_2)$. Since $\text{pre}(a_1)[V_C] = \text{pre}(a_2) \subseteq s_2$ and $\text{pre}(a_1)[\overline{V_C}] \in \mathcal{C}(\overline{V_C}\cdot D_1)$, it follows that there is some $s_1 \in \{s_2 \cup t \mid t \in \mathcal{T}(\overline{V_C}\cdot D_1)\} = \overline{f}(s_2)$ s.t. $\text{pre}(a_1) \subseteq s_1$. Let $t_1 = s_1 \ltimes \text{post}(a_1) = (s_1[V_C] \ltimes \text{post}(a_1)[V_C]) \cup (s_1[\overline{V_C}] \ltimes \text{post}(a_1)[\overline{V_C}]) = (s_2 \ltimes \text{post}(a_2)) \cup (s_1[\overline{V_C}] \ltimes \text{post}(a_1)[\overline{V_C}]) = t_2 \cup (s_1[\overline{V_C}] \ltimes \text{post}(a_1)[\overline{V_C}])$. Hence, $t_1[V_C] = t_2[V_C]$ and, thus, $t_1 \in \overline{f}(t_2)$. Since $\langle s_1, t_1, a_1\rangle \in E_1$, it follows that τ is $\mathbf{C}_{\downarrow}$, too.

RRAa/**RRA**b: The proofs hold both for variant 3a and 3b so these cases need not be distinguished. Since f is the identity function, τ is $\mathbf{M}_{\updownarrow}$. Suppose $\langle s, t, a\rangle \in E_1$, $a \in A_1$ and $a \notin A_2$, which is possible since $A_2 \subseteq A_1$. Then, $R(a, a)$ does not hold so τ is not $\mathbf{R}_{\uparrow}$. Suppose instead that $a \in A_2$ and $\langle s, t, a\rangle \in E_2$. Then, $a \in A_1$ since $A_2 \subseteq A_1$. Furthermore, $\langle s, t, a\rangle \in E_1$ because $S_1 = S_2$. Since $R(a, a)$ must hold it follows that τ is $\mathbf{R}_{\downarrow}$.

Suppose $\langle s_1, t_1, a_1\rangle \in E_1$ and $R(a_1, a_2)$. Then, $a_1 = a_2$ and $a_2 \in A_2$ so $\langle s_1, t_1, a_2\rangle = \langle f(s_1), f(t_1), a_2\rangle \in E_2$. Hence, τ is $\mathbf{C}_{\uparrow}$. Proving that τ is $\mathbf{C}_{\downarrow}$ is analogous.

IDL: Since f is the identity function, τ is $\mathbf{M}_{\updownarrow}$. Suppose $\langle s_1, t_1, a_1\rangle \in E_1$ and note that $\text{pre}(a_1) \subseteq s_1$. Let $a_2 = g(a_1)$. Then, $R(a_1, a_2)$ by definition and $\text{pre}(a_2) = \text{pre}(a_1) \subseteq s_1$. Hence, $\langle s_1, s_1 \ltimes \text{post}(a_2), a_2\rangle \in E_2$ and τ is $\mathbf{R}_{\uparrow}$. Proving $\mathbf{R}_{\downarrow}$ is analogous.

Next, we show that **IDL** is not $\mathbf{C}_{\uparrow}$ nor $\mathbf{C}_{\downarrow}$ by a counterexample. Let $V = \{v\}$, $D_v = \{0, 1\}$, $s_0 = \{(v = 0)\}$, $s_1 = \{(v = 1)\}$, and $A_1 = \{a_1\}$ where $\text{pre}(a_1) = s_1$ and $\text{post}(a_1) = s_0$. Let τ' be an **IDL** transformation from $\mathbb{G}_1$ to $\mathbb{G}_2$. Then, $A_2 = \{a_2\}$ where $a_2 = g(a_1)$ so $\text{pre}(a_2) = s_1$ and $\text{post}(a_2) = \varnothing$. Obviously, $\langle s_1, s_0, a_1\rangle \in E_1$ and $R(a_1, a_2)$ holds by definition. However, $\langle f(s_1), f(s_0), a_2\rangle = \langle s_1, s_0, a_2\rangle \notin E_2$ since $s_1 \ltimes \text{post}(a_2) = s_1$. Consequently, τ' is not $\mathbf{C}_{\uparrow}$. Proving that τ' is not $\mathbf{C}_{\downarrow}$ is analogous, using the same example.

ELA: Arbitrarily choose $s \in S_1$ and observe that $f(s) = \{s \cup t \mid t \in \mathcal{T}(V_M\cdot D_M)\}$ so $|f(s)| > 1$ unless $V_M = \varnothing$. It follows that τ is not $\mathbf{M}_{\uparrow}$.

Let $t \in S_2$. Then, $\overline{f}(t) = t[V_1]$ and τ is $\mathbf{M}_{\downarrow}$.

Suppose $\langle s_2, t_2, a_2\rangle \in E_2$. Then, $\text{pre}(a_2) \subseteq s_2$. Let $a_1 = g^{-1}(a_2)$ and we have $R(a_1, a_2)$ by definition. Let $s_1 = s_2[V_1] = \overline{f}(s_2)$. Then, $\text{pre}(a_1) \subseteq s_1$ since $\text{pre}(a_2) = \text{pre}(a_1) \subseteq \mathcal{C}(V_1\cdot D_1)$. Hence, $\langle s_1, s_1 \ltimes \text{post}(a_1), a_1\rangle \in E_1$ and τ is $\mathbf{R}_{\downarrow}$. Property $\mathbf{R}_{\uparrow}$ can be shown similarly.

Suppose $\langle s_2, t_2, a_2\rangle \in E_2$ and $R(a_1, a_2)$. Then, $a_1 = g^{-1}(a_2)$ and $\text{pre}(a_2) \subseteq s_2$. Let $s_1 = \overline{f}(s_2) = s_2[V_1]$ and $t_1 = \overline{f}(t_2) = t_2[V_1]$. Now, $t_2 = s_2 \ltimes \text{post}(a_2) = (s_2[V_1] \ltimes \text{post}(a_2)[V_1]) \cup (s_2[V_M] \ltimes \text{post}(a_2)[V_M]) = (s_1 \ltimes \text{post}(a_1)) \cup (s_2[V_M] \ltimes \text{post}_M(a_1))$. Hence, $t_1 = t_2[V_1] = s_1 \ltimes \text{post}(a_1)$ so $\langle s_1, t_1, a_1\rangle = \langle \overline{f}(s_2), \overline{f}(t_2), a_1\rangle \in E_1$ and it follows that τ is $\mathbf{C}_{\downarrow}$. Showing $\mathbf{C}_{\uparrow}$ is similar. □

We are now ready to analyse the methods with respect to instance properties. This is done by combining Theorem 26 with the next result.

Theorem 27. *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, the following holds for properties of τ:
1a) $\mathbf{M}_\downarrow\mathbf{R}_\downarrow\mathbf{C}_\downarrow \Rightarrow \mathbf{P}_{\mathbf{S}\downarrow}$ 1b) $\mathbf{M}_\uparrow\mathbf{R}_\uparrow\mathbf{C}_\uparrow \Rightarrow \mathbf{P}_{\mathbf{S}\uparrow}$
2a) $\mathbf{M}_\downarrow \Rightarrow (\mathbf{P}_{\mathbf{W}\downarrow} \Leftrightarrow \mathbf{P}_{\mathbf{S}\downarrow})$ 2b) $\mathbf{M}_\uparrow \Rightarrow (\mathbf{P}_{\mathbf{W}\uparrow} \Leftrightarrow \mathbf{P}_{\mathbf{S}\uparrow})$

Proof. 1a) Suppose τ is $\mathbf{M}_\downarrow\mathbf{R}_\downarrow\mathbf{C}_\downarrow$. We first prove by induction over the path length that every path in $\mathbb{G}_2$ is strongly downward state refinable.

Base case: For every path t of length one in $\mathbb{G}_2$ every $s \in \overline{f}(t)$ is a path in $\mathbb{G}_1$.
Induction: Suppose every path of length k in $\mathbb{G}_2$ is strongly downwards refinable, for some $k > 0$. Let $\sigma = t_0, t_1, \ldots, t_k$ be a path in $\mathbb{G}_2$. There must be some ℓ_2 s.t. $\langle t_0, t_1, \ell_2\rangle \in E_2$. It follows from $\mathbf{R}_\downarrow$ that there is some ℓ_1 s.t. $R(\ell_1, \ell_2)$. Hence, it follows from $\mathbf{C}_\downarrow$ that there is some $\langle s_0, t_0, \ell_1\rangle \in E_1$ s.t. $s_0 \in \overline{f}(t_0)$ and $s_1 \in \overline{f}(t_1)$. The transformation τ is $\mathbf{M}_\downarrow$ so there is an arc $\langle s_0, t_0, \ell_1\rangle \in E_1$ for every $s_0 \in \overline{f}(t_0)$ and every $s_1 \in \overline{f}(t_1)$. It follows from the induction hypothesis that there is a path from every $s_1 \in \overline{f}(t_1)$ to every $s_k \in \overline{f}(t_k)$, implying that there must be a path from every $s_0 \in \overline{f}(t_0)$ to every $s_k \in \overline{f}(t_k)$.

This ends the induction and we conclude that every path in $\mathbb{G}_2$ is strongly downwards state refinable. The result is now immediate from Theorem 15.

2a) Suppose τ is $\mathbf{M}_\downarrow\mathbf{P}_{\mathbf{W}\downarrow}$. For arbitrary $k > 0$, let $t_0, t_1, \ldots, t_k \in S_2$ be arbitrary states s.t. $t_i \in \mathcal{R}_2(t_{i-1})$ for all i $(1 \le i \le k)$. It follows from $\mathbf{P}_{\mathbf{W}\downarrow}$ that there are states $s_0, s_1, \ldots, s_k \in S_1$ s.t. $s_i \in \overline{f}(t_i)$ for all i $(0 \le i \le k)$ and $s_i \in \mathcal{R}_1(s_{i-1})$ for all i $(1 \le i \le k)$. However, $\mathbf{M}_\downarrow$ implies that $|\overline{f}(t_i)| = 1$ for all i $(0 \le i \le k)$. It thus follows that for all i $(1 \le i \le k)$, $s_i \in \mathcal{R}_1(s_{i-1})$ for all $s_{i-1} \in \overline{f}(t_{i-1})$ and all $s_i \in \overline{f}(t_i)$. Hence, $\mathbf{P}_{\mathbf{S}\downarrow}$ holds. The opposite direction is immediate since $\mathbf{P}_{\mathbf{S}\downarrow}$ always implies $\mathbf{P}_{\mathbf{W}\downarrow}$.

1b and 2b follow from symmetry. □

Some consequences of this theorem are summarised in column 3 of Table 1. It should be pointed out that the instance properties in the table are only those that we could derive directly from Theorem 27. Thus, we do not exclude the possibility that also other properties hold. Several conclusions can be drawn from these consequences; one is that spurious states are inherently avoided in **RRA** and **ELA** since both methods are $\mathbf{P}_{\mathbf{S}\downarrow}$ and, consequently, $\mathbf{P}_\downarrow$. We also note that there is a significant computational difference between **ABI** and **ABII**; this is a bit surprising since these two methods have almost identical definitions.

Theorem 28. *ABI is not $\mathbf{P}_{\mathbf{S}\uparrow}$.*

Proof. Let $V = \{u, v\}$, $D_u = D_v = \{0, 1\}$ and $V_C = \{u\}$. Let $s_{xy} = \{(u = x), (v = y)\}$ for $x, y \in \{0, 1\}$. Let $A_1 = \{a_1\}$ where $\mathrm{pre}(a_1) = s_{00}$ and $\mathrm{post}(a_1) = s_{10}$. That is, $E_1 = \{\langle s_{00}, s_{10}, a_1\rangle\}$. Then, $A_2 = \{a_2\}$ where $a_2 = g(a_1)$, $\mathrm{pre}(a_2) = \{(u = 0)\}$ and $\mathrm{post}(a_2) = s_{10}$ so $E_2 = \{\langle s_{00}, s_{10}, a_2\rangle, \langle s_{01}, s_{10}, a_2\rangle\}$. We have $s_{10} \in \mathcal{R}_1(s_{00})$ but $f(s_{10}) = \{s_{10}, s_{11}\}$ while $\mathcal{R}_2(f(s_{00})) = \{s_{00}, s_{10}\}$ so $f(s_{10}) \not\subseteq \mathcal{R}_2(f(s_{00}))$. Hence, τ is not $\mathbf{P}_{\mathbf{S}\uparrow}$. □

Type of method	Properties	
	Method	Instance
ABI	$\mathbf{R}_\updownarrow\mathbf{C}_\updownarrow$	—
ABII	$\mathbf{M}_\uparrow\mathbf{R}_\updownarrow\mathbf{C}_\updownarrow$	$\mathbf{P}_{\mathbf{S}\uparrow}$
RRAa/**RRA**b	$\mathbf{M}_\updownarrow\mathbf{R}_\downarrow\mathbf{C}_\updownarrow$	$\mathbf{P}_{\mathbf{S}\downarrow}$, $\mathbf{P}_{\mathbf{W}\uparrow} \Leftrightarrow \mathbf{P}_{\mathbf{S}\uparrow}$
IDL	$\mathbf{M}_\updownarrow\mathbf{R}_\updownarrow$	$\mathbf{P}_{\mathbf{W}\uparrow} \Leftrightarrow \mathbf{P}_{\mathbf{S}\uparrow}$, $\mathbf{P}_{\mathbf{W}\downarrow} \Leftrightarrow \mathbf{P}_{\mathbf{S}\downarrow}$,
ELA	$\mathbf{M}_\downarrow\mathbf{R}_\updownarrow\mathbf{C}_\updownarrow$	$\mathbf{P}_{\mathbf{S}\downarrow}$

Table 1: Transformation properties of different methods.

9 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. Such insights are important, for instance, when using abstractions for constructing heuristics (Culberson and Schaeffer 1998; Helmert, Haslum, and Hoffmann 2007). However, these properties and classifications 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. For instance, using transformations for quantitative analyses of abstraction (in the vein of Korf (1987)) is a natural continuation of this research.

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, though. One generalisation is to study hierarchies of abstraction levels instead of a single level (Knoblock 1994; Sacerdoti 1974). Generalising the framework in this way is straightforward, and we find it probable that the highly symmetric nature of transformations will simplify the study of complex abstraction hierarchies.

Labels are important in our framework and the examples have ranged from hardly using them at all (as in SHA) to using them massively (as in the planning examples). Labels can be used in many different ways. One example is path refinement via arc paths rather than state paths, which is common in planning (Bacchus and Yang 1994; Knoblock 1994). Then the actions along the abstract path are used as a skeleton for the ground path and then subplans are filled in between these actions. Using actions as labels give an immediate refinement specification from the label relation: if $R(a_1, a_2)$ holds then a_1 is a possible refinement of an abstract action a_2. Holte et al. (1996) investigated this method using labelled graphs similar to ours, but dismissed it as inferior to refining state sequences. However, they only investigated the restricted case where an action must be refined into itself (meaning, in our framework, that $R(a_1, a_2)$ implies $a_1 = a_2$) so their conclusion is not necessarily true in the general case. We want to emphasise that our framework does not prescribe how to use labels and that it is entirely up to the user to decide how to use them. We expect that the labels can be used in completely different (and surprising) ways for adding and handling auxiliary information in search problems.

References

Bacchus, F., and Yang, Q. 1994. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence* 71(1):43–100.

Bäckström, C., and Jonsson, P. 1995. Planning with abstraction hierarchies can be exponentially less efficient. In *Proc. 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, Montréal, QC, Canada, 1599–1605.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proc. 14th National Conference on Artificial Intelligence (AAAI'97)*, Providence, RI, USA, 714–719.

Bundy, A.; Giunchiglia, F.; Sebastiani, R.; and Walsh, T. 1996. Calculating criticalities. *Artificial Intelligence* 88(1-2):39–67.

Christensen, J. 1990. A hierarchical planner that generates its own hierarchies. In *Proc. 8th National Conference on Artificial Intelligence (AAAI'90)*, Boston, MA, USA, 1004–1009.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Domshlak, C.; Katz, M.; and Lefler, S. 2010. When abstractions met landmarks. In*Proc. 20th International Conference on Automated Planning and Scheduling (ICAPS'2010)*, Toronto, ON, Canada, 50–56.

Fink, E., and Yang, Q. 1997. Automatically selecting and using primary effects in planning: theory and experiments. *Artificial Intelligence* 89(1-2):323–389.

Giunchiglia, F., Villafiorita, A., and Walsh, T. 1997. Theories of abstraction. *AI Communications* 10(3-4):167–176.

Giunchiglia, F., and Walsh, T. 1992. A theory of abstraction. *Artificial Intelligence* 57(2-3):285–315.

Haslum, P., and Jonsson, P. 2000. Planning with reduced operator sets. In *Proc. 5th International Conference on Artificial Intelligence Planning Systems (AIPS'2000)*, Breckenridge, CO, USA, 150–158.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proc. 22nd National Conference on Artificial Intelligence (AAAI'2007)*, Vancouver, BC, USA, 1007–1012.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proc. 17th International Conference on Automated Planning and Scheduling (ICAPS'2007)*, Providence, RI, USA, 176–183.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Hoffmann, J. 2005. Where 'ignoring delete lists' works: local search topology in planning benchmarks. *Journal of Artificial Intelligence Research* 24:685–758.

Holte, R. C. and Choueiry, B. C. 2003. Abstraction and reformulation in artificial intelligence. *Philosophical Transactions of the Royal Society London B* 358:1197–1204.

Holte, R. C.; Mkdami, T.; Zimmer, R. M.; and MacDonald, A. J. 1996. Speeding up problem solving by abstraction: A graph oriented approach. *Artificial Intelligence* 85(1–2):321–361.

Knoblock, C. A.; Tenenberg, J. D.; and Yang, Q. 1991. Characterizing abstraction hierarchies for planning. In *Proc. 9th National Conference on Artificial Intelligence (AAAI'1991)*, Anaheim, CA, USA, 692–697.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68(2):243–302.

Korf, R. E. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33(1):65–88.

McDermott, D. V. 1996. A heuristic estimator for means-ends analysis in planning. In *Proc. 3rd International Conference on Artificial Intelligence Planning Systems (AIPS'1996)*, Edinburgh, Scotland, 142–149.

Newell, A.; Shaw, J. C.; and Simon, H. A. 1959. Report on a general problem-solving program. In *IFIP Congress, Paris, France*, 256–264. UNESCO.

Pandurang Nayak, P. and Levy, P. 1995. A semantic theory of abstractions. In *Proc. 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, Montréal, QC, Canada, 196–203.

Sacerdoti, E. D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5(2):115–135.

Zilles, S., and Holte, R. C. 2010. The computational complexity of avoiding spurious states in state space abstraction. *Artificial Intelligence* 174(14):1072–1092.