On the Expressive Power of Non-Linear Merge-and-Shrink Representations

Malte Helmert and **Gabriele Röger** and **Silvan Sievers**
University of Basel
Basel, Switzerland
{malte.helmert,gabriele.roeger,silvan.sievers}@unibas.ch

Abstract

We prove that general merge-and-shrink representations are strictly more powerful than linear ones by showing that there exist problem families that can be represented compactly with general merge-and-shrink representations but not with linear ones. We also give a precise bound that quantifies the necessary blowup incurred by conversions from general merge-and-shrink representations to linear representations or BDDs/ADDs. Our theoretical results suggest an untapped potential for non-linear merging strategies and for the use of non-linear merge-and-shrink-like representations within symbolic search.

Introduction

Merge-and-shrink (*M&S*) abstraction is a framework for deriving abstractions in factored state-space search problems such as those occurring in classical planning. It was originally introduced for model-checking (Dräger, Finkbeiner, and Podelski 2006; 2009) and later adapted to planning (Helmert, Haslum, and Hoffmann 2007; 2008), where the original concepts were generalized further (e.g., Nissim, Hoffmann, and Helmert 2011a; Helmert et al. 2014; Sievers, Wehrle, and Helmert 2014).

The merge-and-shrink framework has been intensely studied recently for many reasons. Classical planning systems based on merge-and-shrink abstraction heuristics have shown excellent performance (e.g., Helmert, Haslum, and Hoffmann 2007; Nissim, Hoffmann, and Helmert 2011b). Merge-and-shrink abstractions strictly generalize *pattern databases* (e.g., Culberson and Schaeffer 1998), a highly influential approach for deriving search heuristics. They offer a clean "algebraic" approach for reasoning about and manipulating abstractions (e.g., Sievers, Wehrle, and Helmert 2014). They dominate many other classes of classical planning heuristics in terms of expressive power (Helmert and Domshlak 2009) and are among the few heuristic approaches that can derive perfect heuristics in polynomial time in nontrivial cases (Nissim, Hoffmann, and Helmert 2011a; Helmert et al. 2014), thus offering polynomial-time optimal planning algorithms for these cases.

Last, but certainly not least, the merge-and-shrink framework offers links between the two successful but largely disjoint research areas of *heuristic search planning* and *planning as symbolic search* due to a relationship between merge-and-shrink representations with so-called *linear merging strategies* and *binary* (Bryant 1985) and *algebraic* (Bahar et al. 1993) *decision diagrams* (*BDDs* and *ADDs*) that was first observed by Bonet (personal communications) and recently studied in more depth (Edelkamp, Kissmann, and Torralba 2012; Torralba, Linares López, and Borrajo 2013; Helmert et al. 2014; Torralba 2015).

The existing theoretical studies of the expressive power of merge-and-shrink representations are limited to the *linear* case, partly because the existing theory did not allow for efficient implementations of general non-linear abstractions until recently. However, with the development of *generalized label reduction* (Sievers, Wehrle, and Helmert 2014) this has changed, and non-linear merge-and-shrink abstractions have subsequently been shown to outperform previous linear ones (Sievers, Wehrle, and Helmert 2014; Fan, Müller, and Holte 2014; Sievers et al. 2015).

Developing nontrivial theoretical results for the general (non-linear) case has turned out to be much more challenging than in the linear case. The most detailed theoretical work on merge-and-shrink representations (Helmert et al. 2014) does not offer theoretical insights comparing the general to the linear case, but makes a number of conjectures entailing that general merge-and-shrink representations are strictly more powerful than linear ones. In this paper, we prove these conjectures correct.

Merge-and-Shrink Representations

The merge-and-shrink framework has mainly been used to derive heuristic functions for state-space search. For the purposes of this paper, however, it is useful to consider *merge-and-shrink representations* as a general mechanism for representing functions mapping variable assignments into some set of values, independently of their concrete use in state-space search (where the variable assignments are states, and they are mapped to numerical heuristic values).

Definition 1. *Let V be a set of* variables*, each with a finite* domain $dom(v) \neq \emptyset$.

An assignment *of a variable set V is a function α defined on V with $\alpha(v) \in dom(v)$ for all $v \in V$. We write $vars(\alpha)$ for the variable set V (i.e., for the domain of definition of α).*

Taking a general perspective helps emphasize that our results, in particular those regarding limitations in the linear case, are inherent properties of the representations as such, and not just due to the particular way that specific abstractions and abstraction heuristics are constructed. The consequences in the context of planning heuristics are then an immediate corollary of the general representation result.

In terms of presentation, this means that we do not need to formally introduce semantics for planning tasks or transition systems, as the results are independent of these aspects. Instead, we directly define merge-and-shrink representations. Throughout the paper, we assume that some underlying finite set of variables V is given.

Definition 2. Merge-and-shrink representations *(*MSR*s) are defined inductively as follows. Each MSR $\mathcal{M}$ has an associated* variable set $vars(\mathcal{M}) \subseteq V$ *and an associated finite* value set $val(\mathcal{M}) \neq \emptyset$.

An MSR $\mathcal{M}$ is either

- atomic*, in which case $|vars(\mathcal{M})| = 1$ and the variable in $vars(\mathcal{M})$ is called the* variable *of $\mathcal{M}$, or*
- *a* merge*, in which case it has a* left component $\mathcal{M}_{\mathrm{L}}$ *and* right component $\mathcal{M}_{\mathrm{R}}$*, which are MSRs with disjoint variable sets. In this case, $vars(\mathcal{M}) = vars(\mathcal{M}_{\mathrm{L}}) \cup vars(\mathcal{M}_{\mathrm{R}})$.*

Each MSR $\mathcal{M}$ has an associated table*, written $\mathcal{M}^{\mathrm{tab}}$.*

- *If $\mathcal{M}$ is atomic with variable v, its table is a function $\mathcal{M}^{\mathrm{tab}} : dom(v) \to val(\mathcal{M})$.*
- *If $\mathcal{M}$ is a merge, its table is a function $\mathcal{M}^{\mathrm{tab}} : val(\mathcal{M}_{\mathrm{L}}) \times val(\mathcal{M}_{\mathrm{R}}) \to val(\mathcal{M})$.*

Merge-and-shrink representations are sometimes referred to as *cascading tables* in the literature (Helmert et al. 2014; Torralba 2015).[1]

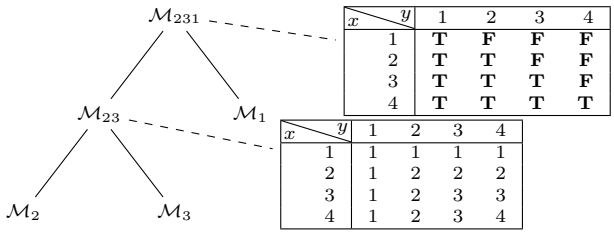
We illustrate the definition with an example, depicted in Figure 1. The variables are $V = \{v_1, v_2, v_3\}$ with domains $dom(v_1) = dom(v_2) = dom(v_3) = \{1, 2, 3, 4\}$. The overall MSR, $\mathcal{M}_{231}$, is a merge with left component $\mathcal{M}_{23}$ and right component $\mathcal{M}_1$. $\mathcal{M}_{23}$ is a merge with left component $\mathcal{M}_2$ and right component $\mathcal{M}_3$. $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ are atomic.

For the atomic MSRs $\mathcal{M}_i$, the variable is v_i, the value set is $\{1, 2, 3, 4\}$, and the table is simply the identity function. The value set of $\mathcal{M}_{23}$ is also $\{1, 2, 3, 4\}$, and its table is the minimum function: $\mathcal{M}_{23}^{\mathrm{tab}}(x, y) = \min(x, y)$. Finally, the value set of $\mathcal{M}_{231}$ consists of the truth values $\{\mathbf{T}, \mathbf{F}\}$, and its table is defined by $\mathcal{M}_{231}^{\mathrm{tab}}(x, y) = \mathbf{T}$ iff $x \geq y$.

The purpose of MSRs is to compactly represent functions defined on assignments as formalized in the next definition.

Definition 3. *An MSR $\mathcal{M}$* represents *a function that maps assignments of $vars(\mathcal{M})$ to values in $val(\mathcal{M})$. The symbol $\mathcal{M}$ is used both for the function and its representation.*

[1]For readers wondering where the *shrinking* aspect of merge-and-shrink factors into the definition: on the representational level, there is no need for any specific mechanisms related to shrinking, as this is integrated into the tables. We refer to the literature for a more detailed discussion (Helmert et al. 2014; Torralba 2015).



x \ y	1	2	3	4
1	**T**	**F**	**F**	**F**
2	**T**	**T**	**F**	**F**
3	**T**	**T**	**T**	**F**
4	**T**	**T**	**T**	**T**

x \ y	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	2	3	3
4	1	2	3	4

Figure 1: Example MSR (value sets and function tables for identity functions at the leaves omitted).

The function represented by $\mathcal{M}$ is defined inductively as follows, where α is an assignment to $vars(\mathcal{M})$ and $\alpha|_{V'}$ denotes the restriction of α to the variable subset V':

- *If $\mathcal{M}$ is atomic with variable v, then $\mathcal{M}(\alpha) := \mathcal{M}^{\mathrm{tab}}(\alpha(v))$.*
- *If $\mathcal{M}$ is a merge, then $\mathcal{M}(\alpha) := \mathcal{M}^{\mathrm{tab}}(\mathcal{M}_{\mathrm{L}}(\alpha|_{vars(\mathcal{M}_{\mathrm{L}})}), \mathcal{M}_{\mathrm{R}}(\alpha|_{vars(\mathcal{M}_{\mathrm{R}})}))$.*

In words, an atomic MSR defines an arbitrary function on the values of the associated variable, and a merge recursively computes a function value for each of its two components (where each component looks at a different subset of the variables) and combines the subresults into the final result.

To illustrate, consider an example assignment for the MSR $\mathcal{M}_{231}$ defined previously:

$$
\begin{aligned}
&\mathcal{M}_{231}(\{v_1 \mapsto 2, v_2 \mapsto 4, v_3 \mapsto 3\}) \\
&= \mathcal{M}_{231}^{\mathrm{tab}}(\mathcal{M}_{23}(\{v_2 \mapsto 4, v_3 \mapsto 3\}), \mathcal{M}_1(\{v_1 \mapsto 2\})) \\
&= \mathcal{M}_{231}^{\mathrm{tab}}(\mathcal{M}_{23}^{\mathrm{tab}}(\mathcal{M}_2(\{v_2 \mapsto 4\}), \mathcal{M}_3(\{v_3 \mapsto 3\})), \\
&\qquad\qquad \mathcal{M}_1(\{v_1 \mapsto 2\})) \\
&= \mathcal{M}_{231}^{\mathrm{tab}}(\mathcal{M}_{23}^{\mathrm{tab}}(\mathcal{M}_2^{\mathrm{tab}}(4), \mathcal{M}_3^{\mathrm{tab}}(3)), \mathcal{M}_1^{\mathrm{tab}}(2)) \\
&= \mathcal{M}_{231}^{\mathrm{tab}}(\mathcal{M}_{23}^{\mathrm{tab}}(4, 3), 2) \\
&= \mathbf{T} \text{ iff } \min(4, 3) \geq 2 \\
&= \mathbf{T}
\end{aligned}
$$

More generally, $\mathcal{M}_{231}$ represents a predicate that is true for assignment α iff $\min(\alpha(v_2), \alpha(v_3)) \geq \alpha(v_1)$.

The structure of an MSR can be viewed as a tree, with the overall MSR as the root and each merge having the MSRs of its components as its children. Such trees are called *merging strategies*, and an important special case is the one where this merging strategy degenerates into a linear sequence.

Definition 4. *An MSR is called* linear *if it is atomic or if it is a merge whose right component is atomic and whose left component is linear.*

The structure of a linear MSR is fully defined by the sequence of associated variables for the leaves of its tree representation, read from left to right. We call this sequence the variable order *of the MSR.*

The important defining property of a linear MSR is that it never contains merges of two merges. The requirement that it is the *right* component that must be atomic is somewhat arbitrary, but serves to make the merging strategy for a given variable order canonical.

The example MSR in Figure 1 is linear, and its variable order is $\langle v_2, v_3, v_1 \rangle$. A *non-linear* MSR is one that is not linear. Sometimes we speak of *general* MSRs to emphasize that we want to include both linear and non-linear MSRs.

We will study the question which functions can be represented compactly with linear and with general merge-and-shrink representations. For this purpose, we need to define a measure for the representation size of an MSR. We assume that the tables are represented explicitly by their entries, and hence the total number of table entries dominates the overall representation size and defines a suitable size measure.

Definition 5. *The* size *of an MSR $\mathcal{M}$, written as $\|\mathcal{M}\|$, and the* size *of its table, written as $\|\mathcal{M}^{\text{tab}}\|$, are defined inductively as follows. If $\mathcal{M}$ is atomic with variable v, then $\|\mathcal{M}\| = \|\mathcal{M}^{\text{tab}}\| = |dom(v)|$. If $\mathcal{M}$ is a merge, then $\|\mathcal{M}^{\text{tab}}\| = |val(\mathcal{M}_{\text{L}})| \cdot |val(\mathcal{M}_{\text{R}})|$ and $\|\mathcal{M}\| = \|\mathcal{M}_{\text{L}}\| + \|\mathcal{M}_{\text{R}}\| + \|\mathcal{M}^{\text{tab}}\|$.*

In the example, we obtain $\|\mathcal{M}_{231}\| = 2 \cdot 4 \cdot 4 + 3 \cdot 4 = 44$, which counts the two 4×4 tables for the merges and the three size-4 tables for the atomic MSRs. If we generalize the example to a larger domain with D values, we obtain a representation size of $\Theta(D^2)$ to define a function on D^3 possible variable assignments.

In order to analyze the scalability of MSRs, we need to consider representation sizes for *families* of functions.

Definition 6. *Let $\mathcal{F}$ be a family of functions on variable assignments. We say that $\mathcal{F}$* has compact MSRs *if there exists a polynomial p such that for each function $f \in \mathcal{F}$ defined over variables V_f, there exists an MSR $\mathcal{M}_f$ for f with $\|\mathcal{M}_f\| \leq p(\sum_{v \in V_f} |dom(v)|)$.*

We say that $\mathcal{F}$ has compact linear MSRs *if additionally all MSRs $\mathcal{M}_f$ are linear.*

In words, the size of compact representations is at most polynomial in the sum of the variable domain sizes (i.e., in the number of variable/value pairs).

Expressive Power of M&S in Previous Work

It is easy to see that merge-and-shrink representations can represent arbitrary functions on assignments, so the study of the expressive power of the merge-and-shrink framework has focused on the (more practically important) question which functions can be represented *compactly*.

It is well-known that a family of functions can be represented compactly with *linear* merge-and-shrink representations iff it can be represented compactly with ADDs,[2] which in turn is equivalent to having compact representations by families of BDDs (e.g., Torralba 2015).

In more detail, Helmert et al. (2014) and Torralba (2015) both prove results that relate linear MSRs to ADDs representing the same function. Translated to our notation, Helmert et al. show that every linear MSR $\mathcal{M}$ can be converted into an equivalent ADD of size $O(|V| T^{\max} D^{\max})$, where $V = vars(\mathcal{M})$, $T^{\max}$ is the maximum over the table sizes of $\mathcal{M}$ and its (direct and indirect) subcomponents, and $D^{\max} = \max_{v \in V} |dom(v)|$. A finer-grained analysis, which also follows from Torralba's results, gives a slightly better bound of $O(\|\mathcal{M}\| D^{\max})$. (Note that $\|\mathcal{M}\| = O(|V| T^{\max})$ because $\|\mathcal{M}\|$ is the sum of $O(|V|)$ table sizes, each of which is individually bounded by $T^{\max}$. Hence the second bound is at least as tight as the first one.)

In the opposite direction, it is easy to convert an arbitrary ADD with N nodes into a linear MSR $\mathcal{M}$ with $\|\mathcal{M}\| = O(N \cdot |V|)$, where V is the set of ADD variables. The conversion first performs the opposite of *Shannon reductions* (which, in the worst case, replaces each ADD edge by a structure of size $O(|V|)$) and then converts the resulting structure into a linear MSR, which incurs no further blowup.

For *non-linear* MSRs, no such bounds are known. However, Helmert et al. (2014) made the following conjecture:

> We conjecture that ... there exists a family $\mathcal{T}_n$ of (non-linear) merge-and-shrink trees over n variables such that the smallest ADD representation of $\mathcal{T}_n$ is of size $\Omega((|\mathcal{T}_n|^{\max})^{c \log n})$ for some constant c. Furthermore, we conjecture that a bound of this form is tight and that the $\log n$ factor in the exponent can be more accurately expressed with the *Horton-Strahler number* of the merge-and-shrink tree.

In the following we will prove this conjecture correct, including the relation to the Horton-Strahler number. Rather than working with ADDs, we prove the results directly on the level of general vs. linear MSRs; the conjectured relationship to ADDs then follows from the above discussion. Specifically, we will prove:

1. Every MSR $\mathcal{M}$ can be converted into a linear MSR of size at most $\|\mathcal{M}\|^{\text{HS}(\mathcal{M})}$, where $\text{HS}(\mathcal{M})$ is the Horton-Strahler number (see next section) of the merging strategy of $\mathcal{M}$. $\text{HS}(\mathcal{M})$ is bounded from above by $\log_2 n + 1$, where n is the number of variables.
2. There exist families of functions which can be compactly represented by general MSRs, but every linear MSR has size $\Omega(\|\mathcal{M}\|^{c \log_2 n})$, where $c > 0$ is a constant, n is the number of variables, and $\mathcal{M}$ is a general MSR representing the same function.

General to Linear M&S Representations

The Horton-Strahler number (Horton 1945) of a rooted binary tree is a measure of its "bushiness". The measure is high for complete trees and low for thin trees.[3] For this paper, it is easiest to define the measure directly for MSRs.

Definition 7. *The* Horton-Strahler *number of an MSR $\mathcal{M}$, written* $\text{HS}(\mathcal{M})$*, is inductively defined as* $\text{HS}(\mathcal{M}) =$

$$\begin{cases} 1 & \text{if } \mathcal{M} \text{ is atomic} \\ \max(\text{HS}(\mathcal{M}_{\text{L}}), \text{HS}(\mathcal{M}_{\text{R}})) & \text{if } \text{HS}(\mathcal{M}_{\text{L}}) \neq \text{HS}(\mathcal{M}_{\text{R}}) \\ \text{HS}(\mathcal{M}_{\text{L}}) + 1 & \text{if } \text{HS}(\mathcal{M}_{\text{L}}) = \text{HS}(\mathcal{M}_{\text{R}}) \end{cases}$$

[2] ADDs only directly support functions over assignments to binary variables. To bridge this gap, Helmert et al. and Torralba encode non-binary variables as contiguous sequences of binary ADD variables, as commonly done in symbolic planning algorithms.

[3] Horton-Strahler numbers of rooted trees are closely related to the *pathwidth* (e.g., Cattell, Dinneen, and Fellows 1996) of the tree seen as an undirected graph. It can be shown that the Horton-Strahler number is always in the range $\{k, \ldots, 2k\}$, where k is the pathwidth (Mamadou M. Kanté, personal communications).

It is easy to prove inductively that the smallest tree (in terms of number of nodes) with Horton-Strahler number k is the complete binary tree with k layers, from which it follows that Horton-Strahler numbers grow at most logarithmically in the number of tree nodes.

For MSRs, this implies $\mathrm{HS}(\mathcal{M}) \leq \log_2 |vars(\mathcal{M})| + 1$. If $\mathcal{M}$ is linear, then $\mathrm{HS}(\mathcal{M}) = 2$, unless $\mathcal{M}$ is atomic (in which case $\mathrm{HS}(\mathcal{M}) = 1$). More generally, MSRs with small Horton-Strahler number are "close" to being linear. In this section we show that general MSRs $\mathcal{M}$ can be converted to linear ones with an increase in representation size that is exponential only in $\mathrm{HS}(\mathcal{M})$. In the following section, we then show that this blow-up is unavoidable.

Before we can prove the main result of this section, we show how to convert an MSR with a *single* violation of the linearity condition into a linear one. The main result uses this construction as a major ingredient.

Throughout the section, we will refer to the *parts* of an MSR $\mathcal{M}$, by which we mean its direct and indirect subcomponents, including $\mathcal{M}$ itself.

Lemma 1. *Let $\mathcal{M}$ be a merge MSR where $\mathcal{M}_\mathrm{L}$ and $\mathcal{M}_\mathrm{R}$ are linear. Then there exists a linear MSR $\mathcal{M}^\mathrm{lin}$ representing the same function as $\mathcal{M}$ with $\|\mathcal{M}^\mathrm{lin}\| \leq \|\mathcal{M}_\mathrm{L}\| + (|val(\mathcal{M}_\mathrm{L})| + 1)\|\mathcal{M}_\mathrm{R}\|$.*

Proof: We demonstrate the construction of $\mathcal{M}^\mathrm{lin}$ given $\mathcal{M}$. To reduce notational clutter, we set $\mathcal{L} = \mathcal{M}_\mathrm{L}$ and $\mathcal{R} = \mathcal{M}_\mathrm{R}$ for the two (linear) components of $\mathcal{M}$. Further, we set $r := |vars(\mathcal{R})|$ and write $\langle v_1, \ldots, v_r \rangle$ for the variable order underlying $\mathcal{R}$. For $1 \leq i \leq r$, we write $\mathcal{R}_i^\mathrm{var}$ for the atomic part of $\mathcal{R}$ with variable v_i and $\mathcal{R}_i$ for the part of $\mathcal{R}$ with $vars(\mathcal{R}_i) = \{v_1, \ldots, v_i\}$. (This implies that $\mathcal{R}_1 = \mathcal{R}_1^\mathrm{var}$, and hence $\mathcal{R}_1$, unlike the other $\mathcal{R}_i$, is atomic.)

If $\mathcal{R}$ is atomic, then $\mathcal{M}$ is already linear and we can set $\mathcal{M}^\mathrm{lin} = \mathcal{M}$. In the following we can thus assume $r \geq 2$.

We construct a sequence of linear MSRs $\mathcal{M}_0, \ldots, \mathcal{M}_r$ such that $\mathcal{M}^\mathrm{lin} = \mathcal{M}_r$ is the desired linear representation for $\mathcal{M}$. To initialize the construction, we set $\mathcal{M}_0 = \mathcal{L}$. All other MSRs $\mathcal{M}_i$ are merges, so to define them we must specify their left and right components, value sets, and tables.

For $1 \leq i \leq r$, $\mathcal{M}_i$ is a merge with left component $\mathcal{M}_{i-1}$ and right component $\mathcal{R}_i^\mathrm{var}$. The value sets are $val(\mathcal{M}_i) = val(\mathcal{L}) \times val(\mathcal{R}_i)$ for all $i \neq r$ and $val(\mathcal{M}_r) = val(\mathcal{M})$.

Finally, the tables are defined as follows:

- $i = 1$: $\mathcal{M}_1^\mathrm{tab}(l, x) = \langle l, \mathcal{R}_1^\mathrm{tab}(x) \rangle$
- $1 < i < r$: $\mathcal{M}_i^\mathrm{tab}(\langle l, x \rangle, y) = \langle l, \mathcal{R}_i^\mathrm{tab}(x, y) \rangle$
- $i = r$: $\mathcal{M}_r^\mathrm{tab}(\langle l, x \rangle, y) = \mathcal{M}^\mathrm{tab}(l, \mathcal{R}_r^\mathrm{tab}(x, y))$

To see that this MSR represents the desired function, it is perhaps easiest to view the evaluation of an MSR for a given assignment as a bottom-up process. Computing $\mathcal{M}(\alpha)$ with the original MSR entails first computing $\mathcal{L}(\alpha|_{vars(\mathcal{L})})$, then $\mathcal{R}(\alpha|_{vars(\mathcal{R})})$ and finally feeding the two component results into $\mathcal{M}^\mathrm{tab}$. During the evaluation of $\mathcal{R}$, the result for $\mathcal{L}$ must be "remembered" because it will be needed at the end when plugging the two component results into $\mathcal{M}^\mathrm{tab}$.

The linearized MSR emulates this process by first computing $\mathcal{L}(\alpha|_{vars(\mathcal{L})})$ and then propagating the resulting value l, unchanged, through all the computation steps of $\mathcal{R}$ as part of the value sets, until the final step, when $\mathcal{R}(\alpha|_{vars(\mathcal{R})})$ becomes available and hence the final result can be computed.

To prove the claimed size bound, observe that all tables of $\mathcal{L}$ occur identically in $\mathcal{M}^\mathrm{lin}$, so the total size of these tables in $\mathcal{M}^\mathrm{lin}$ can be bounded by $\|\mathcal{L}\|$. Similarly, the tables of the atomic parts of $\mathcal{R}$ occur identically in $\mathcal{M}^\mathrm{lin}$ and their size can be bounded by $\|\mathcal{R}\|$. The remaining parts of $\mathcal{M}^\mathrm{lin}$ are the merges $\mathcal{M}_i$ with $1 \leq i \leq r$. The size of $\mathcal{M}_i^\mathrm{tab}$ is $|val(\mathcal{L})|$ times the size of $\mathcal{R}_i^\mathrm{tab}$ (in $\mathcal{R}$), and hence the total size of these tables can be bounded by $|val(\mathcal{L})|\|\mathcal{R}\|$. Summing these three terms gives the desired overall bound. □

Armed with this construction, we can now proceed to prove the main result.

Theorem 1. *Let $D^\mathrm{sum}(\mathcal{M}) = \sum_{v \in vars(\mathcal{M})} |dom(v)|$. For every MSR $\mathcal{M}$ there is a linear MSR $\mathcal{M}^\mathrm{lin}$ representing the same function with $\|\mathcal{M}^\mathrm{lin}\| \leq D^\mathrm{sum}(\mathcal{M})\|\mathcal{M}\|^{\mathrm{HS}(\mathcal{M})-1} \leq \|\mathcal{M}\|^{\mathrm{HS}(\mathcal{M})}$.*

Proof: Clearly $D^\mathrm{sum}(\mathcal{M}) \leq \|\mathcal{M}\|$, so it is sufficient to show $\|\mathcal{M}^\mathrm{lin}\| \leq D^\mathrm{sum}(\mathcal{M})\|\mathcal{M}\|^{\mathrm{HS}(\mathcal{M})-1}$. This holds for linear MSRs with $\mathcal{M}^\mathrm{lin} = \mathcal{M}$, so in the following let $\mathcal{M}$ be nonlinear. In particular, we will use that $\mathcal{M}$ is not atomic.

Let B be the maximum of $|val(\mathcal{P})|$ over all parts $\mathcal{P}$ of $\mathcal{M}$ with $\mathcal{P} \neq \mathcal{M}$. We have $B + 1 \leq \|\mathcal{M}\|$ because each value set size $|val(P)|$ contributes to $\|\mathcal{M}\|$ (in the table of a merge), and it is not the only term that contributes to $\|\mathcal{M}\|$ (because $\mathcal{M}$ is not atomic).

We now prove that every part $\mathcal{P}$ of $\mathcal{M}$ (including $\mathcal{P} = \mathcal{M}$) can be converted to a linear representation $\mathcal{P}^\mathrm{lin}$ of size at most $D^\mathrm{sum}(\mathcal{P})\|\mathcal{M}\|^{\mathrm{HS}(\mathcal{P})-1}$. The claim then follows by setting $\mathcal{P} = \mathcal{M}$.

The proof is by induction over the number of variables in $\mathcal{P}$. If $|vars(\mathcal{P})| = 1$, $\mathcal{P}$ is atomic and the result holds because $\mathcal{P}$ is already linear and satisfies $\|\mathcal{P}\| = D^\mathrm{sum}(\mathcal{P})$.

For the inductive step, let $\mathcal{P}$ be a merge. Let $k = \mathrm{HS}(\mathcal{P})$, $k_\mathrm{L} = \mathrm{HS}(\mathcal{P}_\mathrm{L})$ and $k_\mathrm{R} = \mathrm{HS}(\mathcal{P}_\mathrm{R})$. We can apply the inductive hypothesis to linearize $\mathcal{P}_\mathrm{L}$ and $\mathcal{P}_\mathrm{R}$, obtaining linear representations $\mathcal{P}_\mathrm{L}^\mathrm{lin}$ and $\mathcal{P}_\mathrm{R}^\mathrm{lin}$ with $\|\mathcal{P}_\mathrm{L}^\mathrm{lin}\| \leq D^\mathrm{sum}(\mathcal{P}_\mathrm{L})\|\mathcal{M}\|^{k_\mathrm{L}-1}$ and $\|\mathcal{P}_\mathrm{R}^\mathrm{lin}\| \leq D^\mathrm{sum}(\mathcal{P}_\mathrm{R})\|\mathcal{M}\|^{k_\mathrm{R}-1}$.

If $k_\mathrm{L} > k_\mathrm{R}$, let $\mathcal{Q}$ be the MSR with left component $\mathcal{P}_\mathrm{L}^\mathrm{lin}$ and right component $\mathcal{P}_\mathrm{R}^\mathrm{lin}$ and the same table and value set as $\mathcal{P}$. Clearly $\mathcal{Q}$ represents the same function as $\mathcal{P}$ and satisfies the criteria of Lemma 1, so we can apply the lemma to receive a linear MSR $\mathcal{Q}^\mathrm{lin}$ with size at most:

$$
\begin{aligned}
\|\mathcal{Q}^\mathrm{lin}\| &\leq \|\mathcal{P}_\mathrm{L}^\mathrm{lin}\| + (|val(\mathcal{P}_\mathrm{L}^\mathrm{lin})| + 1)\|\mathcal{P}_\mathrm{R}^\mathrm{lin}\| \\
(*) \quad &\leq \|\mathcal{P}_\mathrm{L}^\mathrm{lin}\| + (B+1)\|\mathcal{P}_\mathrm{R}^\mathrm{lin}\| \\
&\leq \|\mathcal{P}_\mathrm{L}^\mathrm{lin}\| + \|\mathcal{M}\|\|\mathcal{P}_\mathrm{R}^\mathrm{lin}\| \\
&\leq D^\mathrm{sum}(\mathcal{P}_\mathrm{L})\|\mathcal{M}\|^{k_\mathrm{L}-1} + \|\mathcal{M}\|D^\mathrm{sum}(\mathcal{P}_\mathrm{R})\|\mathcal{M}\|^{k_\mathrm{R}-1} \\
&= D^\mathrm{sum}(\mathcal{P}_\mathrm{L})\|\mathcal{M}\|^{k_\mathrm{L}-1} + D^\mathrm{sum}(\mathcal{P}_\mathrm{R})\|\mathcal{M}\|^{k_\mathrm{R}} \\
(**) \quad &\leq D^\mathrm{sum}(\mathcal{P}_\mathrm{L})\|\mathcal{M}\|^{k-1} + D^\mathrm{sum}(\mathcal{P}_\mathrm{R})\|\mathcal{M}\|^{k-1} \\
&= (D^\mathrm{sum}(\mathcal{P}_\mathrm{L}) + D^\mathrm{sum}(\mathcal{P}_\mathrm{R}))\|\mathcal{M}\|^{k-1} \\
&= D^\mathrm{sum}(\mathcal{P})\|\mathcal{M}\|^{k-1}
\end{aligned}
$$

where (*) uses that $\mathcal{P}_\mathrm{L}^\mathrm{lin}$ has the same value set as $\mathcal{P}_\mathrm{L}$, which

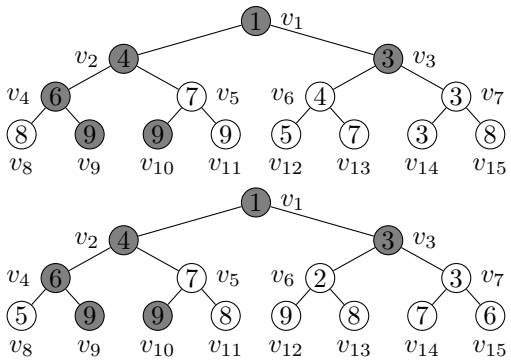


Figure 2: Two assignments for heap predicate $\varphi_{15,9}$. Assignment α (top) satisfies the heap condition; assignment α' (bottom) does not because $\alpha'(v_3) > \alpha'(v_6)$ and $\alpha'(v_4) > \alpha'(v_8)$.

is a part of $\mathcal{M}$, and hence the value set bound B applies to it; (**) uses that $k = k_{\mathrm{L}} > k_{\mathrm{R}}$ in the case considered here.

If $k_{\mathrm{L}} < k_{\mathrm{R}}$, we use essentially the same construction, but make $\mathcal{P}^{\mathrm{lin}}_{\mathrm{R}}$ the *left* and $\mathcal{P}^{\mathrm{lin}}_{\mathrm{L}}$ the *right* component of $\mathcal{Q}$, which allows us to swap the roles of k_{L} and k_{R} in the computation. To represent the same function despite the swapped subcomponents, we set $\mathcal{Q}^{\mathrm{tab}}(x, y) = \mathcal{P}^{\mathrm{tab}}(y, x)$.

Finally, if $k_{\mathrm{L}} = k_{\mathrm{R}}$, we use the same construction and computation as with $k_{\mathrm{L}} > k_{\mathrm{R}}$. Step (**) remains correct because in this case $k = k_{\mathrm{L}} + 1 = k_{\mathrm{R}} + 1$.

In all cases, $\mathcal{Q}^{\mathrm{lin}}$ is the desired linear MSR: we showed that it meets the size bound, and it represents the same function as $\mathcal{Q}$, which represents the same function as $\mathcal{P}$. □

Separation of General and Linear M&S

In this section, we show that the size increase in the conversion from general to linear MSRs in Theorem 1 is unavoidable in general. To prove the result, we describe a family of functions which have compact general merge-and-shrink representations, but do not have compact linear representations. More precisely, we will show that if $\mathcal{M}$ is a compact general representation for such a function f, then every linear representation must have size $\Omega(\|\mathcal{M}\|^{\Theta(\log n)})$, where n is the number of variables of f. This lower bound matches the upper bound of $\|\mathcal{M}\|^{\mathrm{HS}(\mathcal{M})}$ from the previous section.

The functions we consider are *predicates*, i.e., they map to the set of truth values $\{\mathbf{T}, \mathbf{F}\}$. We say that an assignment α *satisfies* a predicate ψ if $\psi(\alpha) = \mathbf{T}$. We now introduce the family of predicates we will study.

Definition 8. *Let $k \in \mathbb{N}_1$, let $n = 2^k - 1$, and let $D \in \mathbb{N}_1$. The* heap predicate $\varphi_{n,D}$ *is a predicate defined over variables $V = \{v_1, \ldots, v_n\}$ with $dom(v) = \{1, \ldots, D\}$ for all $v \in V$. An assignment α of V satisfies $\varphi_{n,D}$ iff $\alpha(v_{\lfloor i/2 \rfloor}) \leq \alpha(v_i)$ for all $1 < i \leq n$. In this case, we say that α satisfies the* heap condition*.*

In the following, we omit k, n and D from the notation when this does not lead to confusion. Observe that $k = \Theta(\log n)$, or more precisely $k = \log_2(n + 1)$.

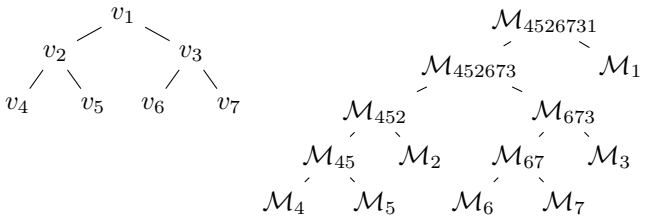


Figure 3: Heap and corresponding merging strategy.

The heap predicate has its name because it is the condition characterizing sequences of values that represent (*min-*) *heaps*, known from algorithm theory (e.g., Cormen, Leiserson, and Rivest 1990). Figure 2 shows two example assignments for heaps of height $k = 4$, i.e., with $n = 2^4 - 1 = 15$ variables. (Ignore the gray shading of some nodes for now.) We follow the common convention in algorithm theory of displaying the assignment as a complete binary tree (not to be confused with trees representing MSRs!), where the variables are numbered in breadth-first order. With this mode of display, the heap condition is satisfied iff the value assigned to each inner node is less than or equal to the values assigned to its children. The assignment shown at the top satisfies the heap condition, while the one at the bottom does not.

General M&S Representations for Heaps

It is easy to see that heap predicates can be represented compactly with general merge-and-shrink representations. In fact, the first MSR we showed in Figure 1 is an example of this, representing the heap predicate $\varphi_{3,4}$.

The general case is only slightly more complex and is illustrated for $n = 7$ variables ($k = 3$) in Figure 3. The left-hand side shows the tree order of the heap variables, and the right-hand side shows the corresponding merging strategy.

The idea is to build merges "along" the tree underlying the heap. For each variable v_i there is a *subtree MSR*, whose variable set includes all variables in the subtree of the heap rooted at v_i, and for each inner variable (in the example: v_1, v_2, v_3) there is a *descendant MSR*, whose variable set includes the same variables as the subtree MSR but not v_i itself. For example, $\mathcal{M}_{452}$ is the subtree MSR for v_2 and $\mathcal{M}_{45}$ is the descendant MSR for v_2 in Figure 3. For inner variables v_i, the descendant MSR is the merge of the subtree MSRs of the (heap) children of v_i, and the subtree MSR is the merge of the descendant MSR of v_i and an atomic MSR for v_i. Leaf nodes v_i of the heap do not have descendant MSRs, and their subtree MSR is an atomic MSR for v_i.

The value sets and tables for these MSRs are constructed in such a way that each MSR represents the *minimum* function over all variables it covers. This can easily be computed locally by setting $\mathcal{M}^{\mathrm{tab}}(x, y) = \min(x, y)$ everywhere. To complete the construction, we need a way of determining violations of the heap condition. This is accomplished by adding an additional value *violation* to the value sets of all merges, which is set at the subtree MSR of inner variable v_i whenever $\alpha(v_i)$ is larger than the value propagated up from the descendant MSR for v_i. We define $\min(x, y) =$ *violation* whenever $x =$ *violation* or

$y = \mathit{violation}$, which ensures that violations are propagated towards the root. To complete the construction, the table at the root converts *violation* to **F** and all other values to **T**.

Theorem 2. *The family of all heap predicates has compact MSRs.*

Proof: It is easy to verify that the MSRs $\mathcal{M}_{n,D}$ described in the text represent the heap functions $\varphi_{n,D}$. We can bound the representation size by $\|\mathcal{M}_{n,D}\| \leq n \cdot D + (n-1)(D+1)^2 = O(nD^2)$, where the first term counts the n tables of size D in the atomic MSRs, and the other term counts the $n-1$ tables of size $(D+1)^2$ (taking into account the additional *violation* value) at merges. This is polynomial in the sum over the domain sizes, which is $\sum_{i=1}^{n} D = nD$. □

Linear M&S Representations for Heaps

In our final result, we show that heap predicates cannot be represented compactly with linear merge-and-shrink representations, no matter which variable order is used. Towards this end, we need some additional definitions relating to assignments in the context of heap predicates. Throughout the section, $V = \{v_1, \ldots, v_n\}$ refers to the variables of a given heap predicate $\varphi_{n,D}$.

We make heavy use of *partial* assignments (assignments to subsets of variables) in the following. For $V' \subseteq V$, a *V'-assignment* is an assignment for V'. When we speak of *assignments* without further qualification, we mean assignments to all variables V.

Definition 9. *Let V be the variable set of some heap predicate $\varphi_{n,D}$, and let $V' \subseteq V$. The* unassigned variables *of V' are the variables $V \setminus V'$. An assignment for the unassigned variables of V' is called a* completion *of a V'-assignment.*

A V'-assignment σ together with a completion τ forms an assignment, which we write as $\sigma \cup \tau$.

A completion τ is a valid completion *for V'-assignment σ if $\sigma \cup \tau$ satisfies the heap condition. A V'-assignment is called* consistent *if it has at least one valid completion.*

To illustrate the definition, we refer back to Figure 2. The variable subset $V' = \{v_1, v_2, v_3, v_4, v_9, v_{10}\}$ is shaded in gray, and the unassigned variables are shown in white. We verify that the V'-assignment in both parts of the figure is identical, so the figure shows a V'-assignment σ together with a valid completion τ at the top (which proves that σ is consistent) and an invalid completion τ' at the bottom.

We make the following central observation: if the variable set V' occurs as a prefix of the variable order of a given linear MSR for $\varphi_{n,D}$, then the sub-MSR $\mathcal{M}_{V'}$ with variables V' may only "combine" (map to the same value) V'-assignments that have exactly the same valid completions. Otherwise the final merge-and-shrink representation cannot faithfully represent the heap predicate. Hence, the number of elements in $\mathit{val}(\mathcal{M}_{V'})$ is at least as large as the number of V'-assignments that are *distinguishable* in this sense.[4]

[4]The same observation underlies the *Myhill-Nerode theorem* on the size of minimal deterministic finite automata (e.g., Hopcroft, Motwani, and Ullman 2001, Section 4.4) and the *Sieling-Wegener bound* for the minimal representation size of BDDs (Sieling and Wegener 1993).

Definition 10. *V'-assignments σ and σ' are called* equivalent *if they have exactly the same set of valid completions. V'-assignments that are not equivalent are called* distinguishable*.*

We will show that, no matter which variable order is chosen, some part of the linear MSR must have a large number of distinguishable V'-assignments. To prove this result, we will need some more definitions.

Definition 11. *The* neighbors *of v_i are its parent in the heap, $v_{\lfloor i/2 \rfloor}$ (if $i > 1$) and its children in the heap, v_{2i} and v_{2i+1} (if $i < 2^{k-1}$). A variable $v_i \in V'$ is called a* frontier variable *of V' if it has an unassigned neighbor.*

In the example of Figure 2, the frontier variables of V' are v_2 (due to its child v_5), v_3 (due to its children v_6 and v_7), v_4 (due to its child v_8) and v_{10} (due to its parent v_5).

The frontier variables are the ones that relate the variables assigned in V' to the unassigned variables, which makes them particularly important. It is easy to see that, given a consistent V'-assignment, *only* the values of the frontier variables matter when deciding whether a given completion is valid. (We do not prove this result because it does not help prove a lower bound, but it may serve to provide intuition for the role of frontier variables.)

We now show that for every variable order, there must be a sub-MSR of the overall MSR with a certain minimal number of frontier variables.

Lemma 2. *Let V be the variable set of some heap predicate $\varphi_{n,D}$, and let $\pi = \langle v_{j_1}, \ldots, v_{j_n} \rangle$ be any variable order of V. Then there exists some $r \in \{1, \ldots, n\}$ such that the set of variables $V' = \{v_{j_1}, \ldots, v_{j_r}\}$ defined by the length-r prefix of π includes at least $\lfloor \frac{k}{2} \rfloor$ frontier variables.*

Proof: The problem of determining an ordering π of the vertices of a graph that minimizes the maximal number of frontier variables of any prefix of π has been studied in graph theory. The minimal number of frontier variables required is known as the *vertex separation number* of the graph and is known to be equal to its *pathwidth* (Kinnersley 1992).

Here, the graphs we must consider are the complete binary trees T_k with k layers. Cattell, Dinneen, and Fellows (1996) show that the pathwidth of T_k is at least $\lfloor \frac{k}{2} \rfloor$. □

The basic idea of our main proof is that if there are many frontier variables, then there exist many V'-assignments that need to be distinguished. Towards this end, we introduce certain properties that will help us determine that given V'-assignments are distinguishable.

Definition 12. *A V'-assignment σ is called* sorted *if $\sigma(v_i) \leq \sigma(v_j)$ whenever $i < j$. A V'-assignment σ is called* fraternal *if, for any two variables $v, v' \in V'$ that are siblings in the heap (i.e., $v = v_{2i}$ and $v' = v_{2i+1}$ for some $1 \leq i < 2^{k-1}$), we have $\sigma(v) = \sigma(v')$.*

For example, the V'-assignment shown in gray in Figure 2 is not sorted because $\sigma(v_2) > \sigma(v_3)$ even though $2 < 3$. Furthermore, it is not fraternal because v_2 and v_3 are siblings and have different values $\sigma(v_2) \neq \sigma(v_3)$. It would be sorted and fraternal if we had $\sigma(v_2) = 3$.

Next, we show that with many frontier variables, many V'-assignments are sorted and fraternal.

Lemma 3. *Let V be the variable set of some heap predicate $\varphi_{n,D}$, and let $V' \subseteq V$ contain ℓ frontier variables. Then there exists a set $\Sigma(V')$ of sorted fraternal V'-assignments such that $|\Sigma(V')| = \binom{D}{\lceil \ell/2 \rceil}$ and any two assignments in $\Sigma(V')$ differ on at least one frontier variable of V'.*

Proof: We say that a variable is *sibling-bound* if it has a parent (i.e., is not the root), it is the right child of its parent, and both the variable and its sibling are in V'. We say that a variable is *sibling-bound in the frontier* if it is sibling-bound and additionally both the variable and its sibling are in the frontier. A *pivot candidate* is a frontier variable that is not sibling-bound in the frontier.

There can be at most $\lfloor \ell/2 \rfloor$ variables that are sibling-bound in the frontier because all such variables must have other frontier variables (that are not sibling-bound) as their siblings. Therefore, there are at least $\ell' = \ell - \lfloor \ell/2 \rfloor = \lceil \ell/2 \rceil$ pivot candidates. We arbitrarily select ℓ' pivot candidates and call them *pivots*.

We construct the set $\Sigma(V')$ by defining one sorted fraternal V'-assignment σ_X for each subset $X \subseteq \{1, \ldots, D\}$ of cardinality ℓ'. There are $\binom{D}{\ell'}$ such subsets, and the construction guarantees that σ_X and $\sigma_{X'}$ with $X \neq X'$ differ on at least one pivot and hence on at least one frontier variable.

We now describe how to construct σ_X for a given subset X. First, assign the values of X to the pivots in index order (i.e., the lowest value in X goes to the pivot with lowest index, etc.). This results in a sorted assignment to the pivots.

Next, to each variable in V' that is not sibling-bound and not a pivot, assign the value of the pivot that precedes it in index order, or 1 if no pivot precedes it. This maintains sortedness and results in an assignment to all non-sibling-bound variables in V'.

Finally, assign the value assigned to its sibling to each sibling-bound variable in V'. This results in a fraternal assignment to V' and again maintains sortedness.

As an example, consider the set of variables $V' = \{v_1, v_2, v_3, v_4, v_9, v_{10}\}$ from Figure 2 with $D = 9$. The frontier variables are $\{v_2, v_3, v_4, v_{10}\}$, giving $\ell = 4$ and $\ell' = 2$. The only sibling-bound variable in V' is v_3, and the other three frontier variables are pivot candidates. Select v_2 and v_4 as the pivots. Let $X = \{5, 7\}$. We first assign the values of X to the pivots in order: $\sigma_X(v_2) = 5$ and $\sigma_X(v_4) = 7$. Next, we fill in the remaining non-sibling-bound variables: $\sigma_X(v_1) = 1$, $\sigma_X(v_9) = 7$, $\sigma_X(v_{10}) = 7$. Finally, sibling-bound v_3 receives the value of its sibling: $\sigma_X(v_3) = 5$. □

As the final preparation for the main proof, we show that all assignments constructed in Lemma 3 are distinguishable.

Lemma 4. *Let σ and σ' be two sorted fraternal V'-assignments that differ on at least one frontier variable of V'. Then σ and σ' are distinguishable.*

Proof: It is easy to see that sorted V'-assignments are always consistent. More generally, a V'-assignment σ is consistent iff for all assigned variables $v_i, v_j \in V'$ where v_i is an ancestor of v_j in the tree representation of the heap, we have $\sigma(v_i) \leq \sigma(v_j)$. We start by describing two special completions of a given consistent V'-assignment.

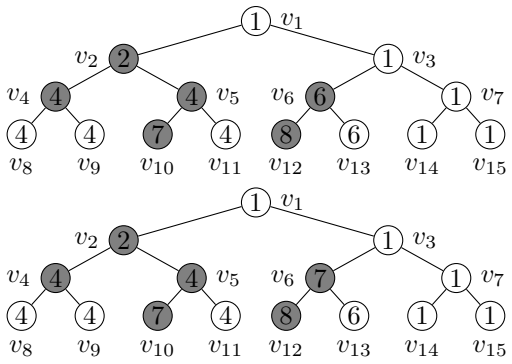


Figure 4: Top: sorted fraternal V'-assignment σ (gray nodes) with its minimal valid completion $\tau_{\min}$ (white nodes), satisfying the heap condition. Bottom: sorted fraternal V'-assignment σ' (gray nodes) with the same completion, violating the heap condition for v_6 and its child v_{13}.

The *minimal valid completion* $\tau_{\min}$ of a consistent V'-assignment σ assigns the lowest possible value that preserves consistency to each unassigned variable. It can be computed by iterating over all unassigned variables in index order (i.e., from the root towards the leaves) and assigning to each variable the lowest value that is consistent with its parent (if the variable has a parent). If the root variable v_1 is unassigned, this means setting $\tau_{\min}(v_1) := 1$. For any other unassigned variable v_i ($i > 1$), this means assigning the value of the parent: $\tau_{\min}(v_i) := (\sigma \cup \tau_{\min})(v_{\lfloor \frac{i}{2} \rfloor})$. (Note that when v_i is processed, the values of $\tau_{\min}$ for lower-index variables have already been constructed, so $(\sigma \cup \tau_{\min})(v_{\lfloor \frac{i}{2} \rfloor})$ is defined.) The top half of Figure 4 shows a V'-assignment σ together with its minimal valid completion.

Similarly, the *maximal valid completion* $\tau'_{\max}$ of a consistent V'-assignment σ' assigns the highest possible value that preserves consistency to each unassigned variable. It can be computed by iterating over all unassigned variables in reverse index order (i.e., from the leaves towards the root) and assigning to each variable the highest value that is consistent with its children (if the variable has children). For unassigned leaf variables v_i ($i \geq 2^{k-1}$), this means setting $\tau'_{\max}(v_i) := D$. For unassigned inner variables v_i ($i < 2^{k-1}$), this means assigning the minimum value of the children: $\tau'_{\max}(v_i) := \min((\sigma' \cup \tau'_{\max})(v_{2i}), (\sigma' \cup \tau'_{\max})(v_{2i+1}))$. (Again, this computation only depends on values of $\tau'_{\max}$ that have already been constructed.)

We now prove the claim. Let σ and σ' be sorted fraternal V'-assignments that differ on some frontier variable v_i. Without loss of generality, we can assume $\sigma(v_i) < \sigma'(v_i)$ (otherwise swap the roles of σ and σ'). We will show that σ and σ' are distinguishable by describing a completion that is valid for one of σ or σ', but not the other.

Because v_i is a frontier variable of V', it belongs to V' and has a parent or child that is unassigned. We first consider the easier case where v_i has a child v_j ($j = 2i$ or $j = 2i + 1$) that is unassigned.

Consider the minimal valid completion $\tau_{\min}$ of σ. By con-

struction, $\tau_{\min}$ is a valid completion of σ. From the definition of minimal valid completions, we have $\tau_{\min}(v_j) = \sigma(v_i)$. We also have $\sigma(v_i) < \sigma'(v_i)$ and hence $\tau_{\min}(v_j) < \sigma'(v_i)$, so the child v_j has a lower value than its parent v_i in $\sigma' \cup \tau$. This shows that $\tau_{\min}$ is not a valid completion for σ', which concludes this case of the proof.

Figure 4 illustrates this proof argument, showing two sorted fraternal V'-assignments σ (top half) and σ' (bottom half) together with the minimal valid completion $\tau_{\min}$ of σ. We see that $\sigma \cup \tau_{\min}$ at the top satisfies the heap condition, while $\sigma' \cup \tau_{\min}$ at the bottom violates it for v_6 (a variable in V' with $\sigma(v_6) < \sigma'(v_6)$) and its child v_{13}.

We now consider the remaining case, where v_i with $\sigma(v_i) < \sigma'(v_i)$ has an unassigned parent v_j ($j = \lfloor \frac{i}{2} \rfloor$). (The two cases are of course not disjoint, but they are exhaustive.) This time, we consider the maximal valid completion $\tau'_{\max}$ of σ', which by construction is a valid completion of σ'. We show that it is not a valid completion of σ by demonstrating $\tau'_{\max}(v_j) > \sigma(v_i)$, so that the heap condition is violated between v_j and its child v_i. More precisely, we will show $\tau'_{\max}(v_j) = \sigma'(v_i)$, from which the result follows with $\sigma'(v_i) > \sigma(v_i)$.

Let v_s be the sibling of v_i, i.e., the other child of v_j. By definition of maximal valid completions, we have $\tau'_{\max}(v_j) = \min((\sigma' \cup \tau'_{\max})(v_i), (\sigma' \cup \tau'_{\max})(v_s))$. If $v_s \in V'$, the values of v_i and v_s are both defined by σ', and we obtain $\tau'_{\max}(v_j) = \min(\sigma'(v_i), \sigma'(v_s)) = \sigma'(v_i)$ because σ' is fraternal and hence $\sigma'(v_i) = \sigma'(v_s)$.

If $v_s \notin V'$, then it is easy to see from the definition of maximal valid completions that $\tau(v_s)$ is a value assigned by σ' to some *descendant* v_d of v_s that belongs to V' (possibly several layers removed), or the maximal possible value D if no such descendant exists. In either case, this value is at least as large as $\sigma'(v_i)$ because $\sigma'(v_i) \leq D$ unconditionally, and if the value derives from a descendant v_d, then we must have $i < d$, from which $\sigma'(v_i) \leq \sigma'(v_d)$ follows because σ' is sorted. So also in this case we obtain $\tau'_{\max}(v_j) = \sigma'(v_i)$, which concludes the proof. □

We are now ready to put the pieces together.

Theorem 3. *The heap predicates do not have compact linear MSRs.*

Proof: We must show that the minimum representation size required by linear MSRs for $\varphi_{n,D}$ grows faster than any polynomial in nD, no matter which merging strategy (variable order) is chosen. We describe a sequence of heap predicates with a size parameter $s \in \mathbb{N}_1$ for which the result can already be established. Then it of course extends to the family of heap predicates as a whole.

For $s \in \mathbb{N}_1$, we consider the heap predicate $\varphi_{n,D}$ with $k = 4s$ (and hence $n = 2^k - 1$) and $D = s \cdot n$. We observe that $D = \frac{1}{4}kn = \frac{1}{4}\log_2(n+1)n = O(n \log n)$, and hence a polynomial size bound in nD exists for the given subset of heap predicates iff a polynomial size bound in n exists. Therefore, to show the overall result, it is sufficient to show that heap predicates of the chosen form do not have linear MSRs of size $O(n^c)$ for any constant c.

Let $s \in \mathbb{N}_1$, and let $k = 4s$, $n = 2^k - 1$ and $D = s \cdot n$. Let π be any variable order for $\varphi_{n,D}$.

From Lemma 2, π has a prefix consisting of a variable set V' with at least $\ell = \lfloor \frac{k}{2} \rfloor = \lfloor \frac{4s}{2} \rfloor = 2s$ frontier variables.

From Lemma 3, there exists a set $\Sigma(V')$ of sorted fraternal V'-assignments differing on at least one frontier variable, where $|\Sigma(V')| = \binom{D}{\lceil \ell/2 \rceil} = \binom{D}{\lceil 2s/2 \rceil} = \binom{D}{s}$.

From Lemma 4, all assignments in $\Sigma(V')$ are distinguishable from each other. Therefore, $\binom{D}{s}$ is a lower bound on the size of the value set in the sub-MSR for V', and hence it is also a lower bound on the overall representation size.

We have $\binom{D}{s} \geq \left(\frac{D}{s}\right)^s = \left(\frac{s \cdot n}{s}\right)^s = n^s = n^{\frac{1}{4}k} = n^{\frac{1}{4}\log_2(n+1)}$, which grows faster than any polynomial in n, concluding the proof. □

We briefly remark that for the application of merge-and-shrink representations to classical planning, it is easy to construct planning tasks whose perfect heuristics are in 1:1 correspondence with the heap predicates $\varphi_{n,D}$. This implies that there exist families of planning tasks that can be perfectly solved in polynomial time with non-linear merge-and-shrink heuristics, but not with linear ones. Unfortunately, we cannot provide details due to space limitations, as a detailed discussion would require additional background on planning tasks, transition systems, etc.

Discussion

We showed that general merge-and-shrink representations are strictly more expressive than linear ones: a general-to-linear conversion is possible, but incurs an unavoidable super-polynomial representational blow-up. We bounded this blow-up from above and below, and the upper and lower bounds coincide up to a constant factor in the exponent: general representations of size $\|\mathcal{M}\|$ can always be converted to linear ones of size $\|\mathcal{M}\|^{\Theta(\log |vars(\mathcal{M})|)}$ and more compact linear representations do not exist in general. We also presented a refined upper bound in terms of the Horton-Strahler number of the merging strategy, which measures how close a given merging strategy is to the linear case.

Our results offer theoretical justification for the recent interest in non-linear merge-and-shrink abstractions (Sievers, Wehrle, and Helmert 2014; Fan, Müller, and Holte 2014; Sievers et al. 2015) and motivate further research in this area. They also indicate that it may be worth questioning the ubiquity of *BDD representations* (which are polynomially equivalent to linear merge-and-shrink representations) for symbolic search in automated planning and other areas. General merge-and-shrink representations retain many of the properties of BDDs that make them useful for symbolic search, but offer a larger than polynomial advantage over BDDs in cases where linear variable orders cannot capture information dependencies well.

Can we build strong symbolic search algorithms using general merge-and-shrink representations instead of BDDs, replacing *variable orders* by *variable trees*? The same question has been asked about *sentential decision diagrams* (Darwiche 2011), which extend BDDs with a different generalization from linear to tree structures and have shown much initial promise. We believe that there is significant scope for further investigations in this direction.

Acknowledgments

This work was supported by the Swiss National Science Foundation (SNSF) as part of the project "Abstraction Heuristics for Planning and Combinatorial Search" (AHPACS).

References

Bahar, R. I.; Frohm, E. A.; Gaona, C. M.; Hachtel, G. D.; Macii, E.; Pardo, A.; and Somenzi, F. 1993. Algebraic decision diagrams and their applications. In Lightner, M. R., and Jess, J. A. G., eds., *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1993)*, 188–191.

Bryant, R. E. 1985. Symbolic manipulation of Boolean functions using a graphical representation. In Ofek, H., and O'Neill, L. A., eds., *Proceedings of the 22nd ACM/IEEE Conference on Design Automation (DAC 1985)*, 688–694.

Cattell, K.; Dinneen, M. J.; and Fellows, M. R. 1996. A simple linear-time algorithm for finding path-decompositions of small width. *Information Processing Letters* 57:197–203.

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1990. *Introduction to Algorithms*. The MIT Press.

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

Darwiche, A. 2011. SDD: A new canonical representation of propositional knowledge bases. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 819–826.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2006. Directed model checking with distance-preserving abstractions. In Valmari, A., ed., *Proceedings of the 13th International SPIN Workshop (SPIN 2006)*, volume 3925 of *Lecture Notes in Computer Science*, 19–34. Springer-Verlag.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2009. Directed model checking with distance-preserving abstractions. *International Journal on Software Tools for Technology Transfer* 11(1):27–37.

Edelkamp, S.; Kissmann, P.; and Torralba, Á. 2012. Symbolic A^* search with pattern databases and the merge-and-shrink abstraction. In De Raedt, L.; Bessiere, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P., eds., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 306–311. IOS Press.

Fan, G.; Müller, M.; and Holte, R. 2014. Non-linear merging strategies for merge-and-shrink based on variable interactions. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS 2014)*, 53–61. AAAI Press.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM* 61(3):16:1–63.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In Boddy, M.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 176–183. AAAI Press.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2008. Explicit-state abstraction: A new method for generating heuristic functions. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 1547–1550. AAAI Press.

Hopcroft, J. E.; Motwani, R.; and Ullman, J. D. 2001. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2nd edition.

Horton, R. E. 1945. Erosional development of streams and their drainage basins; hydrophysical approach to quantitative morphology. *Bulletin of the Geological Society of America* 56:275–370.

Kinnersley, N. G. 1992. The vertex separation number of a graph equals its path-width. *Information Processing Letters* 42:345–350.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011a. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1983–1990.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011b. The Merge-and-Shrink planner: Bisimulation-based abstraction for optimal planning. In *IPC 2011 planner abstracts*, 106–107.

Sieling, D., and Wegener, I. 1993. NC-algorithms for operations on binary decision diagrams. *Parallel Processing Letters* 3(1):3–12.

Sievers, S.; Wehrle, M.; Helmert, M.; Shleyfman, A.; and Katz, M. 2015. Factored symmetries for merge-and-shrink abstractions. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3378–3385. AAAI Press.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized label reduction for merge-and-shrink heuristics. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2358–2366. AAAI Press.

Torralba, Á.; Linares López, C.; and Borrajo, D. 2013. Symbolic merge-and-shrink for cost-optimal planning. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2394–2400.

Torralba, Á. 2015. *Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning*. Ph.D. Dissertation, Universidad Carlos III de Madrid.