From Projections to Cartesian Abstractions and Merge-and-Shrink: Comparing the Compactness of Abstraction Representations

Malte Helmert, Gabriele Röger

University of Basel, Switzerland
{malte.helmert,gabriele.roeger}@unibas.ch

Abstract

The four major classes of state space abstractions in classical planning are projections, domain abstractions, Cartesian abstractions and merge-and-shrink abstractions. We study the representations for these classes through the lens of compactness: one class is at least as compact as another if it can represent all abstractions the other can represent with a polynomially bounded increase in representation size. We complete the picture of compactness relationships among the four classes of abstractions, in particular resolving an open question from the literature on the relative compactness of Cartesian abstractions and merge-and-shrink.

Introduction

Abstractions are one of the fundamental tools in state-space search. Their main use is as a source of heuristic functions. The classical planning literature distinguishes four major classes of abstractions:

- *Projections*, on which classical pattern databases are based, perfectly represent the values of a subset of state variables (the *pattern*) and abstract away the other state variables (e.g., Culberson and Schaeffer 1998; Edelkamp 2001).
- *Domain abstractions* apply an individual abstraction to each state variable and use the product of these per-variable abstractions as the overall abstraction (Hernádvölgyi and Holte 2000).
- *Cartesian abstractions* partition the state space into a collection of *Cartesian sets*, i.e., sets of states that are the product of a subset of values for each state variable (Seipp and Helmert 2013, 2018).
- *Merge-and-shrink abstractions* apply a sequence of *merge* and *shrink* transformations to a factored state space description, where a merge step combines two state variables into one by forming their product, and a shrink transformation applies an abstraction to a state variable (Helmert, Haslum, and Hoffmann 2007; Sievers and Helmert 2021).

We provide formal definitions in the next two sections. Mathematically, each class of abstractions in this list can be seen as a special case of the next, and thus they form a hierarchy. Projections are a special case of domain abstractions, which are in turn a special case of Cartesian abstractions, and these are a special case of merge-and-shrink abstractions because *every* abstraction can be represented as a merge-and-shrink abstraction.

However, this discussion does not take into account the representation size of these abstractions. In a practical algorithm, the limiting aspect for all classes of abstractions is the available memory. If, for example, Cartesian abstractions can only be compiled into merge-and-shrink abstractions with an exponential increase in the memory requirements, then merge-and-shrink abstractions cannot fully replace Cartesian abstractions in practice.

Previous work that considered the relationship between these classes of abstractions allows us to conclude that projections can be compiled into domain abstractions and these in turn into Cartesian abstractions with polynomial space overhead (e.g., Seipp and Helmert 2018), and a polynomial compilation of domain abstractions into merge-and-shrink is a folklore result. However, these results were not previously formalized, and there is currently no discussion of whether these relationships are *strict* in the sense that, for the reverse direction, no polynomial compilations exist. In this work, we address these gaps.

For Cartesian abstractions vs. merge-and-shrink abstractions, the literature contains no results or even conjectures on their relative compactness. For example, Seipp and Helmert (2018) observe:

> It is open whether every Cartesian abstraction has an equivalent merge-and-shrink abstraction whose representation is at most polynomially larger (p. 541).

The converse question – can every merge-and-shrink abstraction be polynomially compiled into a Cartesian abstraction – has been equally unresolved. We show that neither of the two classes of abstractions can be polynomially compiled into the other by proving that for either of the two classes, there exist families of abstractions for which every equivalent abstraction from the other class must be super-polynomially larger. This shows that Cartesian abstractions and merge-and-shrink abstractions are truly complementary, and neither of them can in general serve as a substitute for the other.

Background

Representations of different classes of abstractions for state-space search are largely independent of the representation of the state space. The only aspect of the state space that is relevant to our discussion is how states are represented. We assume a representation as a finite set of finite-domain variables, which covers all commonly used representations in classical planning and related areas.

Definition 1 (variable space, assignment, state function)**.** *A* variable space $\mathcal{V} = \langle V, dom \rangle$ *consists of a finite, non-empty set* $V = \{v_1, \ldots, v_n\}$ *of finite-domain variables, where each* $v_i \in V$ *is associated with a finite domain* $dom(v_i)$*. The encoding size of* $\mathcal{V}$ *is* $\|\mathcal{V}\| = \sum_{i=1}^{n} |dom(v_i)|$*.*

A variable assignment *for* V *is a function* $\alpha : V \to \bigcup_{v \in V} dom(v)$ *with* $\alpha(v) \in dom(v)$ *for all* $v \in V$*. We refer to the set of all such assignments as* $\mathcal{A}(\mathcal{V})$*, or as* $\mathcal{A}(V)$ *if the variable domains are clear from context.*

A state function *for* $\mathcal{V}$ *is a function* $f : \mathcal{A}(\mathcal{V}) \to W$*, where* W *is any codomain.*

The encoding size of a variable space accounts for storing the possible values for each variable and follows the practice in classical planning that finite-domain variables are viewed as categorical and hence their domains are explicitly enumerated. In particular, a planning task with encoding size N can reference at most $O(N)$ different variable/value pairs in preconditions, effects, or goals, and hence a "unary" encoding of finite-domain variables does not meaningfully increase task representation size. A different definition would be appropriate for bounded numerical state variables in planning formalisms supporting operations such as incrementing a finite-domain variable by a constant, where a bound B would require $\lceil \log_2(B+1) \rceil$ bits.

While we motivated our contribution in the context of representing abstractions, it more generally applies to representing arbitrary state functions (functions defined for all variable assignments). For example, in state-space search we want to represent heuristic functions, which are state functions $h : \mathcal{A}(\mathcal{V}) \to \mathbb{R}_0^+ \cup \{\infty\}$.

For an equivalence relation $\sim$ over a set X, we write $[x]_\sim$ for the equivalence class of $x \in X$, or $[x]$ if $\sim$ is clear from context. Its *quotient set* is $X/\sim = \{[x]_\sim \mid x \in X\}$.

Representations

We now introduce the four classes of representations for state functions that we analyze in this paper.

Pattern Databases

Pattern databases are the representations used for abstractions based on projections. A pattern database is defined for a pattern $P \subseteq V$ and stores a table containing one value for every $\beta \in \mathcal{A}(P)$.

Definition 2 ($\mathcal{PDB}$)**.** *A* pattern database *(PDB)* $D = \langle P, DB \rangle$ *over a variable space* $\mathcal{V} = \langle V, dom \rangle$ *and codomain* W *consists of a* pattern $P \subseteq V$ *and a* database $DB : \mathcal{A}(P) \to W$*. Its representation size* $\|D\|$ *is* $|P| + \prod_{v \in P} |dom(v)|$*. We refer to the class of all PDBs as* $\mathcal{PDB}$*.*

The historical term *database* is a bit misleading, and the term *table* might be clearer. We follow the traditional terminology.

A PDB represents the function that maps $\alpha \in \mathcal{A}(\mathcal{V})$ to the value stored for $\alpha|_P$, the projection (restriction of the domain of definition) of α to the set P. In practice, the entries of a pattern database are stored in such a way that they can quickly be accessed by a perfect hash function.

Definition 3 (Represented function of a PDB)**.** *A PDB* $D = \langle P, DB \rangle$ *over variable space* $\mathcal{V}$ *and codomain* W represents *the state function* $[\![D]\!] : \mathcal{A}(\mathcal{V}) \to W$ *defined as* $[\![D]\!](\alpha) = DB(\alpha|_P)$*.*

Pattern databases, like the other representations we discuss in the following, can represent arbitrary state functions because they turn into a lookup table for every state if we set $P = V$. However, such a representation is prohibitive if the number of states is too large to be represented in memory, which is generally the case for the kind of state-space search problems we are interested in (or else we could just apply Dijkstra's algorithm).

Domain Abstraction Representations

In a pattern database, the values of each state variable are either fully represented in the abstraction, or no distinction at all is made between these values. Domain abstractions generalize this idea by specifying an equivalence relation over the domain of each variable, where equivalent values are not distinguished by the abstraction.

Definition 4 ($\mathcal{DAR}$)**.** *A* domain abstraction representation *(DAR)* $D = \langle (\sim_v)_{v \in V}, DB \rangle$ *over a variable space* $\mathcal{V} = \langle V, dom \rangle$ *and codomain* W *consists of an equivalence relation* $\sim_v$ *over* $dom(v)$ *for each variable* $v \in V$ *and a* database $DB : \mathcal{A}(V_D) \to W$*, where* $\mathcal{A}(V_D)$ *is the set of all functions* $\alpha : V \to \bigcup_{v \in V} dom(v)/\!\sim_v$ *with* $\alpha(v) \in dom(v)/\!\sim_v$ *for all* $v \in V$*. Its representation size* $\|D\|$ *is* $\sum_{v \in V} |dom(v)| + \prod_{v \in V} |dom(v)/\!\sim_v|$*. We refer to the class of all domain abstraction representations as* $\mathcal{DAR}$*.*

The term $\sum_{v \in V} |dom(v)|$ in the representation size accounts for the representation of the equivalence relations, assuming an explicit enumeration of the equivalence classes.

Definition 5 (Represented function of a DAR)**.** *A DAR* $D = \langle (\sim_v)_{v \in V}, DB \rangle$ *over variable space* $\mathcal{V} = \langle V, dom \rangle$ *and codomain* W represents *the state function* $[\![D]\!] : \mathcal{A}(\mathcal{V}) \to W$ *defined as* $[\![D]\!](\alpha) = DB(\alpha')$*, where* $\alpha'(v) = [\alpha(v)]_{\sim_v}$ *for all* $v \in V$*.*

Cartesian Abstraction Trees

A Cartesian abstraction tree is a decision tree where the inner nodes query a variable v and partition the variable space according to a partition of $dom(v)$. The leaves contain the function values.

Definition 6 ($\mathcal{CAT}$)**.** Cartesian abstraction trees *(CATs) over variable spaces* $\mathcal{V} = \langle V, dom \rangle$ *and codomains* W *are inductively defined as follows:*

- *Every value* $w \in W$ *is a CAT over* $\mathcal{V}$ *and* W*, called a* leaf with value w*. We graphically depict it as* $\boxed{w}$*.*

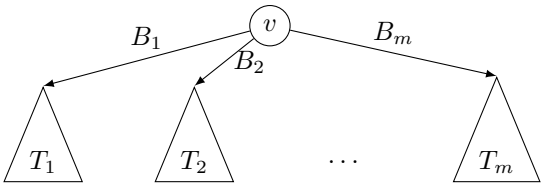

Figure 1: Cartesian abstraction tree (split on variable v).

- *Let $v \in V$, and let $\{B_1, \ldots, B_m\}$ be a partition of dom(v). For $1 \leq i \leq m$, let T_i be a CAT over $\mathcal{V}_i$ and W, where $\mathcal{V}_i$ is identical to $\mathcal{V}$ except that the domain of v is B_i. Then $\langle v, \{\langle B_1, T_1\rangle, \ldots, \langle B_m, T_m\rangle\}\rangle$ is a CAT over $\mathcal{V}$ and W, called a* split on v. *We graphically depict it as shown in Figure 1.*

The representation size $\|T\|$ of a CAT T is $\|T\| = 1$ if T is a leaf, and $\|T\| = |dom(v)| + \sum_{i=1}^{m} \|T_i\|$ if $T = \langle v, \{\langle B_1, T_1\rangle, \ldots, \langle B_m, T_m\rangle\}\rangle$ is a split on v. We refer to the class of all CATs as $\mathcal{CAT}$.

Cartesian abstraction trees are the data structures used in the existing implementations of Cartesian abstractions in the planning literature (e.g., Seipp and Helmert 2018). The term *Cartesian abstraction tree* is new to this work, as this data structure has no established name in the literature. Seipp and Helmert call it a *refinement hierarchy* and limit splits to two children, but their implementation supports splits into m children as in our definition. This distinction is not relevant to our results because it is easy to see that CATs with only binary splits and general CATs are equivalent under the notion of efficient compilability we introduce in the next section.

The state function represented by a CAT is computed by traversing the tree from the root to a leaf, following the path defined by the given assignment.

Definition 7 (Represented function of a CAT)**.** *A CAT T over variable space $\mathcal{V} = \langle V, dom\rangle$ and codomain W* represents *the state function $[\![T]\!] : \mathcal{A}(\mathcal{V}) \to W$ inductively defined as follows:*

- *If $T = w$ is a leaf, then $[\![T]\!](\alpha) = w$.*
- *If $T = \langle v, \{\langle B_1, T_1\rangle, \ldots, \langle B_m, T_k\rangle\}\rangle$ is a split on v and $\alpha(v) \in B_i$, then $[\![T]\!](\alpha) = [\![T_i]\!](\alpha)$.*

For the Cartesian abstractions considered in the planning literature, which are generated by a counterexample-guided abstraction refinement (CEGAR) algorithm (Seipp and Helmert 2018), it is easy to construct a CAT alongside the CEGAR computation such that the leaves of the CAT and the abstract states are in 1:1 correspondence, and this is what the existing implementations do.

Such a 1:1 correspondence *cannot* be established for the general definition of Cartesian abstractions in the literature, which states that an abstraction is Cartesian iff the preimage of every abstract state under the abstraction mapping is a Cartesian set (Seipp and Helmert 2018, Definition 4).

Figure 2 illustrates a Cartesian abstraction, with each of the five rectangles standing for an abstract state, where no CAT with five leaves representing the abstraction mapping exists. To see this, note that any split at the root of the CAT

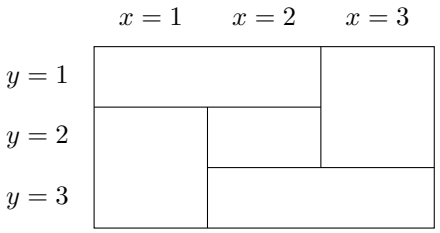

Figure 2: Cartesian abstraction not representable by a CAT.

that partitions the domain of x or y into two or more subsets cuts through at least one abstract state, and hence at least one abstract state must be represented in multiple subtrees of the root and hence in multiple leaves of the CAT.

The compilability results in this paper are not affected by whether we consider CATs or general Cartesian abstractions because all positive results (showing compact representations of Cartesian abstractions) already hold for the more limited CATs, while the negative result we prove (showing a limitation of the compactness of Cartesian abstractions) also applies to the general case.

Merge-and-Shrink Representations

For merge-and-shrink representations, we follow the definitions by Helmert, Röger, and Sievers (2015), adapted for consistency with the representations introduced earlier.

Definition 8 ($\mathcal{MSR}$)**.** Merge-and-shrink representations *(*MSR*s) M over variable spaces $\mathcal{V} = \langle V, dom\rangle$ and codomains W are inductively defined as follows.*

- *Let $v \in V$ and tbl : dom$(v) \to W$. Then $\langle v, tbl\rangle$ is an MSR for $\mathcal{V}$ and W called an* atomic *MSR for v.*
- *For $i \in \{1, 2\}$, let $\mathcal{V}_i = \langle V_i, dom_i\rangle$ be a variable space and W_i be a codomain, with $V_i \subseteq V$ and $V_1 \cap V_2 = \emptyset$. Let M_i be an MSR for $\mathcal{V}_i$ and W_i, and let tbl : $W_1 \times W_2 \to W$. Then $\langle M_1, M_2, tbl\rangle$ is an MSR for $\mathcal{V}$ and W called a* merge *of M_1 and M_2.*

The representation size $\|M\|$ of an MSR M is $\|M\| = |dom(v)|$ if M is an atomic MSR for v, and $\|M\| = \|M_1\| + \|M_2\| + |W_1||W_2|$ if M is a merge of M_1 with codomain W_1 and M_2 with codomain W_2.

An atomic MSR represents a lookup table indexed by a single variable. A merge MSR recursively computes the values of two sub-MSRs and then performs another table lookup to combine these two values into an overall result.

Different papers in the literature use different terminology for MSRs. For example, Helmert et al. (2014) call them *cascading tables*, and Sievers and Helmert (2021) call them *factored mappings*. Not all definitions in the literature require that the variable sets of sub-MSRs in a merge must be disjoint, but MSRs used in practice enforce this restriction because it guarantees that the evaluation of MSRs is efficient (linear in $|V|$) and all represented abstractions are induced abstractions (Sievers and Helmert 2021).

Definition 9 (Represented function of an MSR)**.** *An MSR M over variable space $\mathcal{V}$ and codomain W* represents *the state function $[\![M]\!] : \mathcal{A}(\mathcal{V}) \to W$ inductively defined as follows:*

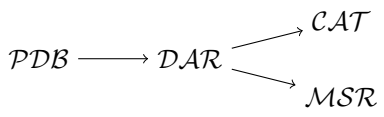

Figure 3: Result overview. For every path from $\mathcal{R}$ to $\mathcal{S}$ in the figure, we have $\mathcal{R} \preceq_{\mathrm{p}} \mathcal{S}$. Whenever no such path exists, we have $\mathcal{R} \not\preceq \mathcal{S}$.

- *If $M = \langle v, tbl \rangle$ is atomic, then $[\![M]\!](\alpha) = tbl(\alpha(v))$.*
- *If $M = \langle M_1, M_2, tbl \rangle$ is a merge, then $[\![M]\!](\alpha) = tbl([\![M_1]\!](\alpha|_{V_1}), [\![M_2]\!](\alpha|_{V_2}))$, where V_i is the set of variables of the variable space of M_i.*

Compilability

We are now ready to formally introduce the notion of compilability we study in our work. It is inspired by the definition of compact representability by Helmert, Röger, and Sievers (2015) and common notions of polynomial compilability such as Karp reductions (Karp 1972)

Definition 10 (Compilability)**.** *Let $\mathcal{R}$ and $\mathcal{S}$ be classes of representations for state functions.*

We say that $\mathcal{R}$ can be compactly compiled *into $\mathcal{S}$, written $\mathcal{R} \preceq \mathcal{S}$, if there exists a function $f : \mathcal{R} \to \mathcal{S}$ and a polynomial p such that for every representation $R \in \mathcal{R}$ for variable space $\mathcal{V}$ and codomain W, we have $[\![f(R)]\!] = [\![R]\!]$ and $\|f(R)\| \leq p(\|\mathcal{V}\| + \|R\|)$.*

If f can be computed in polynomial time in $\|\mathcal{V}\| + \|R\|$, we additionally say that $\mathcal{R}$ can be efficiently compiled *into $\mathcal{S}$, written $\mathcal{R} \preceq_{\mathrm{p}} \mathcal{S}$.*

In other words, we say that one class of representations (such as DARs) can be compactly compiled into another class of representations (such as CATs) if every instance of the first class can be converted into an equivalent instance of the second class without too much growth in the size of the representation (polynomially bounded).

The notion of compact compilability imposes no restrictions on the resources used to perform the compilation, only on the size of the end result. This is desirable for the *negative* results we show: for example, we will see that no compact compilations from CATs to MSRs or vice versa exist, even with unbounded computation. This resolves the open question discussed in the introduction. For all the *positive* results we show, we will prove efficient compilability, not just compact compilability.

In the remainder of the paper, we prove for all combinations of representations from PDBs, DARs, CATs and MSRs whether or not one can be compactly and/or efficiently compiled into the other. It is easy to see from Definition 10 that both notions of compilability are transitive, which reduces the number of cases that must be considered.

Figure 3 shows an overview of the results. In order to justify the figure, we will prove the following individual results:

- $\mathcal{PDB} \preceq_{\mathrm{p}} \mathcal{DAR}$ (Theorem 1)
- $\mathcal{DAR} \not\preceq \mathcal{PDB}$ (Theorem 4)
- $\mathcal{DAR} \preceq_{\mathrm{p}} \mathcal{CAT}$ (Theorem 5)
- $\mathcal{DAR} \preceq_{\mathrm{p}} \mathcal{MSR}$ (Theorem 6)
- $\mathcal{CAT} \not\preceq \mathcal{MSR}$ (Theorem 11)
- $\mathcal{MSR} \not\preceq \mathcal{CAT}$ (Theorem 15)

All other positive and negative results follow from these by transitivity and proof by contradiction. In particular, note that we do not need to give a direct proof that $\mathcal{CAT} \not\preceq \mathcal{DAR}$ because, by contradiction, if we had $\mathcal{CAT} \preceq \mathcal{DAR}$, then with $\mathcal{DAR} \preceq \mathcal{MSR}$ we would transitively obtain $\mathcal{CAT} \preceq \mathcal{MSR}$, a contradiction. Similarly, we do not need to prove $\mathcal{MSR} \not\preceq \mathcal{DAR}$ by a symmetric argument, swapping the roles of $\mathcal{CAT}$ and $\mathcal{MSR}$.

$\mathcal{PDB}$ Can be Efficiently Compiled into $\mathcal{DAR}$

It is well-known that PDBs are essentially a special case of DARs, not just in terms of the abstraction functions they can represent, but also in terms of the representation itself (e.g., Kreft et al. 2023). The following result proves this formally for our notion of efficient compilability.

Theorem 1. *$\mathcal{PDB}$ can be efficiently compiled into $\mathcal{DAR}$.*

Proof. Consider an arbitrary PDB $D_{\mathrm{PDB}} = \langle P, DB_{\mathrm{PDB}} \rangle$ over the variable space $\mathcal{V} = \langle V, dom \rangle$ with codomain W.

We construct a DAR $D_{\mathrm{DAR}} = \langle (\sim_v)_{v \in V}, DB_{\mathrm{DAR}} \rangle$ over $\mathcal{V}$ and W as follows. For $v \in P$, set $x \sim_v y$ iff $x = y$. For v in $V \setminus P$, set $x \sim_v y$ for all $x, y \in dom(v)$. In other words, distinguish all values of variables in P and distinguish no values for the other variables.

For every entry $DB_{\mathrm{PDB}}(\alpha) = w$ in the database of D_{PDB}, create a corresponding entry $DB_{\mathrm{DAR}}(\alpha') = w$ in the database of D_{DAR} for α' defined as $\alpha'(v) = \{\alpha(v)\}$ for all $v \in P$ and $\alpha'(v) = dom(v)$ for all $v \notin P$. It is easy to verify that $[\![D_{\mathrm{DAR}}]\!] = [\![D_{\mathrm{PDB}}]\!]$.

We can construct $(\sim_v)_{v \in V}$ in linear time and space in $\|\mathcal{V}\|$. We can construct DB_{DAR} in linear time and space in $\|D_{\mathrm{PDB}}\|$. (In fact, with a typical implementation via perfect hash functions, the two tables are represented exactly identically.) Hence, the overall construction has time and space bounded by $\|\mathcal{V}\| + \|D_{\mathrm{PDB}}\|$, a (linear) polynomial. □

$\mathcal{DAR}$ Cannot be Compiled into $\mathcal{PDB}$

To show that $\mathcal{DAR}$ cannot be compactly compiled into $\mathcal{PDB}$, we need a family of state functions for which the size of PDBs must grow super-polynomially in the size of DARs. For this purpose, we define *parity functions*.

Definition 11 (parity functions)**.** *For $n \geq 1$ and $d \geq 2$, define $V_n = \{v_1, \ldots, v_n\}$ and let $\mathcal{V}_{n,d}$ be the variable space $\langle V_n, dom \rangle$ with $dom(v) = \{0, \ldots, d-1\}$ for all $v \in V_n$.*

The parity function *$p_{n,d}$ is the state function $p_{n,d} : \mathcal{A}(\mathcal{V}_{n,d}) \to \{0, 1\}$ defined as*

$$p_{n,d}(\alpha) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} \alpha(v_i) \text{ is odd} \\ 0 & \text{otherwise} \end{cases}$$

Because a sum of natural numbers is odd iff it contains an odd number of odd terms, we can equivalently say that $p_{n,d}(\alpha) = 1$ iff $\alpha(v_i)$ is odd for an odd number of variables.

PDBs representing parity functions must be large.

Theorem 2. *Every PDB representing $p_{n,d}$ has size $n + d^n$.*

Proof. Let $n \geq 1$ and $d \geq 2$, and let $D = \langle P, \mathit{DB} \rangle$ be a PDB representing $p_{n,d}$. Then its pattern must be $P = V_n$: assume that there exists a variable $v_\star \in V_n \setminus P$ that is not part of the pattern. Consider an arbitrary $\alpha \in \mathcal{A}(\mathcal{V}_{n,d})$. Define $\alpha' \in \mathcal{A}(\mathcal{V}_{n,d})$ as $\alpha'(v) = \alpha(v)$ for all $v \neq v_\star$ and $\alpha'(v_\star) = \alpha(v_\star) - 1$ if $\alpha(v_\star) > 0$ and $\alpha'(v_\star) = \alpha(v_\star) + 1$ if $\alpha(v_\star) = 0$. Then the number of variables with an odd value differs by 1 between α and α', and hence $p_{n,d}(\alpha) \neq p_{n,d}(\alpha')$. But since $\alpha|_P = \alpha'|_P$, we have $[\![D]\!](\alpha) = [\![D]\!](\alpha')$, which contradicts $[\![D]\!] = p_{n,d}$.

We can conclude that every PDB D representing $p_{n,d}$ must have the pattern V_n and therefore $\|D\| = n + d^n$. □

DARs representing parity functions must also be exponentially large in n because they cannot fully abstract away a state variable for the same reason that PDBs cannot. Nevertheless, their size is more modest than for PDBs, especially if d is large compared to n.

Theorem 3. *The function $p_{n,d}$ can be represented by a DAR with size $dn + 2^n$.*

Proof. We define a DAR $D_{n,d}$ with $[\![D_{n,d}]\!] = p_{n,d}$ as follows. For all variables $v \in V_n$, use the same equivalence relation $\sim$ with equivalence classes $B_\mathrm{e} = \{z \in \{0, \ldots, d-1\} \mid z \text{ is even}\}$ and $B_\mathrm{o} = \{z \in \{0, \ldots, d-1\} \mid z \text{ is odd}\}$, i.e., $x \sim y$ if x and y have the same parity. The database maps every assignment $\beta : V_n \to \{B_\mathrm{e}, B_\mathrm{o}\}$ to 1 if $\beta(v) = B_\mathrm{o}$ for an odd number of $v \in V_n$ and to 0 otherwise. $D_{n,d}$ has size $\|D_{n,d}\| = dn + 2^n$. □

We now prove that the previous two results imply that no compact compilation from DARs to PDBs exists.

Theorem 4. *$\mathcal{DAR}$ cannot be compactly compiled into $\mathcal{PDB}$.*

Proof. To prove the result, we show that for parity functions, the increase in representation size from DARs to PDBs cannot be polynomially bounded. We focus on parity functions of the form $p_{n,n}$ with $n \geq 2$, i.e., the special case of $p_{n,d}$ where $n = d$.

The proof is by contradiction. Assume that q is a polynomial that satisfies the criteria of Definition 10 to show that $\mathcal{DAR} \preceq \mathcal{PDB}$. From Theorem 3, there exists a DAR D_DAR representing $p_{n,n}$ with $\|D_\mathrm{DAR}\| = n \cdot n + 2^n$. From Theorem 2, every PDB D_PDB representing $p_{n,n}$ has size $\|D_\mathrm{PDB}\| = n + n^n$. It follows that $\|D_\mathrm{PDB}\| = n + n^n \leq q(\|\mathcal{V}\| + \|D_\mathrm{DAR}\|) = q((n \cdot n) + (n \cdot n + 2^n)) = q(2n^2 + 2^n)$ for all $n \geq 2$. This is impossible: if q has degree k, the right-hand side of this inequality has asymptotic growth $\Theta(2^{nk}) = \Theta(c^n)$ for constant $c = 2^k$, while the left-hand side has the strictly larger asymptotic growth $\Theta(n^n)$. □

$\mathcal{DAR}$ Can be Efficiently Compiled into $\mathcal{CAT}$

Next, we show that domain abstraction representations can be efficiently compiled into Cartesian abstraction trees. Compared to the compilation from PDBs to DARs this result is slightly less obvious because the two classes of representations are more dissimilar.

Theorem 5. *$\mathcal{DAR}$ can be efficiently compiled into $\mathcal{CAT}$.*

Proof. Let $D = \langle (\sim_v)_{v \in V}, \mathit{DB} \rangle$ be a domain abstraction representation over a variable space $\mathcal{V} = \langle V, \mathit{dom} \rangle$ with variables $V = \{v_1, \ldots, v_n\}$ and codomain W.

The CAT for D is a split on v_1 with one child for each equivalence class of $\mathit{dom}(v_1)/\!\sim_{v_1}$ that narrows down the domain of v_1 to this equivalence class. Each of these children is a split on v_2 that follows the same pattern, all the way down to v_n. The splits on v_n associate v_n with an equivalence class of $\mathit{dom}(v_n)/\!\sim_{v_n}$ and lead to leaf CATs that hold the *DB* entry of the corresponding mapping from each variable to an equivalence class. If the resulting CAT contains any splits with only one child, we repeatedly replace one such split by its only child until all splits have at least two children. This does not affect the represented function.

It is easy to see that this construction can be performed in polynomial time in $|\mathcal{V}|$ and the size of the resulting CAT T and that this CAT represents $[\![D]\!]$. It remains to show the polynomial size bound: $\|T\| \leq p(\|\mathcal{V}\| + \|D\|)$ for some fixed polynomial p.

We have $\|D\| = \sum_{v \in V} |\mathit{dom}(v)| + N = \|\mathcal{V}\| + N$ for $N = \prod_{v \in V} |\mathit{dom}(v)/\!\sim_v|$. From the construction, T has N leaves. Because all splits have at least two children, T has at most N splits. (Trees where all inner nodes have two or more children have more leaves than inner nodes.)

We can upper-bound the size of any CAT with N leaves and at most N splits by $N + \max_{v \in V} |\mathit{dom}(v)| \cdot N$ because every leaf contributes 1 to the representation size and every split contributes the size of some variable domain. (Note that the variable domains in all splits are subsets of the original variable domains.) We can bound N by $\|D\|$ and $\max_{v \in V} |\mathit{dom}(v)|$ by $\|\mathcal{V}\|$ and thus obtain: $\|T\| \leq \|D\| + \|\mathcal{V}\| \cdot \|D\| \leq (\|\mathcal{V}\| + \|D\|)^2 = p(\|\mathcal{V}\| + \|D\|)$ for the polynomial $p(x) = x^2$, concluding the proof. □

$\mathcal{DAR}$ Can be Efficiently Compiled into $\mathcal{MSR}$

We now present our final positive compilation result, showing that domain abstraction representations can be efficiently compiled into merge-and-shrink representations.

Theorem 6. *$\mathcal{DAR}$ can be efficiently compiled into $\mathcal{MSR}$.*

Proof. Let $D = \langle (\sim_v)_{v \in V}, \mathit{DB} \rangle$ be a domain abstraction representation over a variable space $\mathcal{V} = \langle V, \mathit{dom} \rangle$ with variables $V = \{v_1, \ldots, v_n\}$ and codomain W.

We construct a linear merge-and-shrink representation where each atomic MSR abstracts the variable v into values $\{1, \ldots, k\}$ corresponding to the equivalence classes of $\mathit{dom}(v)/\!\sim_v$, and these atomic MSRs are then merged with no further shrinking. This allows us to make the same distinctions as D at the root MSR, so that the table of the root MSR can compute $[\![D]\!]$. This construction can clearly be performed in polynomial time in the size of the resulting MSR.

We now describe the construction more formally. If $n = 1$, the MSR M for D is the atomic MSR $\langle v_1, \mathit{tbl} \rangle$ with $\mathit{tbl}(d) = [\![D]\!](\{v_1 \mapsto d\})$ for all $d \in \mathit{dom}(v_1)$. The size of this MSR is $|\mathit{dom}(v_1)|$, which is bounded by $\|D\|$.

Otherwise we construct an MSR M_n for D as follows: for each variable v, define the set $N_v = \{1, \ldots, |\mathit{dom}(v)/\!\sim_v|\}$

and an arbitrary bijection $b_v : \mathit{dom}(v)/\!\sim_v \to N_v$ that associates every equivalence class with a unique number in N_v. For $i \in \{2, \dots, n\}$, define auxiliary bijections $h_i : N_{v_1} \times \cdots \times N_{v_i} \to \{1, \dots, m_i\}$, where $m_i = \prod_{k \in \{1,\dots,i\}} |N_{v_k}|$. We can use easily invertible perfect hash functions for this.

For $v \in V$, let M_v be the atomic MSR $\langle v, \mathit{tbl}_v \rangle$ with codomain N_v and $\mathit{tbl}_v(d) = b_v([d]_{\sim_v})$ for all $d \in \mathit{dom}(v)$.

For $i \in \{2, \dots, n\}$, let M_i be the merge MSR $\langle M_\mathrm{L}, M_\mathrm{R}, \mathit{tbl}_i \rangle$ with codomain $\{1, \dots, m_i\}$, where $M_\mathrm{L} = M_{i-1}$ if $i > 2$ and $M_\mathrm{L} = M_{v_1}$ if $i = 2$, and $M_\mathrm{R} = M_{v_i}$.

For $i < n$, we set $\mathit{tbl}_i(w, w') = h_i(w_1, \dots, w_i)$, where $\langle w_1, \dots, w_{i-1} \rangle = h_{i-1}^{-1}(w)$ and $w_i = w'$ for all w in the codomain of M_{i-1} and $w' \in N_{v_i}$.

For $i = n$, we set $\mathit{tbl}_i(w, w') = \mathit{DB}(\beta)$, where $\beta(v_i) = b_{v_i}^{-1}(w_i)$ with $w_1, \dots, w_i$ defined as in the case $i < n$.

Without loss of generality, we assume that $|N_v| > 1$ for all variables v: otherwise, we can skip the atomic MSR for v in the above construction. We can bound the representation size $\|M_n\|$ by $\sum_{i=1}^{n} |\mathit{dom}(v_i)| = \|\mathcal{V}\|$ contributed by the atomic MSRs plus $N = \sum_{i=2}^{n} m_i$ contributed by the merge MSRs. Because $|N_v| \geq 2$ for all variables v, we have $m_i = m_{i-1} \cdot |N_{v_i}| \geq 2m_{i-1}$ for all $i \in \{3, \dots, n\}$, and therefore each term in N is at least twice the previous term, from which it follows that $N \leq 2m_n$. (Compare the geometric sum $1 + 2 + 4 + \cdots + 2^k < 2 \cdot 2^k$.)

We have $\|D\| = \sum_{v \in V} |\mathit{dom}(v)| + \prod_{v \in V} |\mathit{dom}(v)/\!\sim_v| = \|\mathcal{V}\| + \prod_{k=1}^{n} |N_{v_k}| = \|\mathcal{V}\| + m_n$. Comparing this to $\|M\| = \|\mathcal{V}\| + N \leq \|\mathcal{V}\| + 2m_n$, we see that $\|M\| \leq 2\|D\| \leq 2(\|\mathcal{V}\| + \|D\|)$, showing a polynomial size bound with the polynomial $p(x) = 2x$. □

$\mathcal{CAT}$ Cannot be Compiled into $\mathcal{MSR}$

In this section we show the central result of this paper: that $\mathcal{CAT}$ cannot compactly be compiled into $\mathcal{MSR}$. This is by far the most complex proof because MSRs are very flexible representations, and it is nontrivial to show that something cannot be compactly represented by any MSR.

Our construction exploits one weakness of MSRs: that they must decompose the variables of the variable space into a fixed tree, generalizing the *variable orders* of binary decision diagrams and related data structures (Bryant 1986). The state functions we consider require moving between the variables of the task in a context-dependent way, where the next variable to consider depends on the values of previously considered variables. Such context-dependency can be more naturally handled by Cartesian abstraction trees.

To prove the result, we define a family of state functions f_n for which the size of merge-and-shrink representations must grow at least exponentially in n, while there exist Cartesian abstraction trees of polynomial size.

Family $(f_n)_{n \in \mathbb{N}_1}$ is defined in terms of a family of *composition functions* $g_{n,k,i}$.

Definition 12 (composition functions)**.** *For $n \in \mathbb{N}_1, k \in \mathbb{N}_0$ and $i \in \{1, \dots, n\}$, define the variable space $\mathcal{V}_n = \langle V_n, \mathit{dom}_n \rangle$ with $V_n = \{1, \dots, n\}$ and $\mathit{dom}_n(v) = \{1, \dots, n\}$ for all $v \in V_n$.*

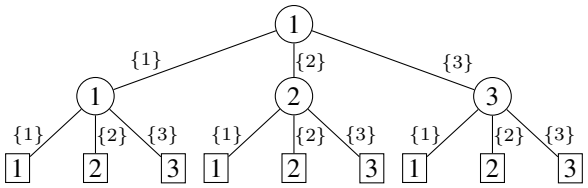

Figure 4: Pseudo-CAT $\tilde{T}_{3,2,1}$ for $g_{3,2,1}$.

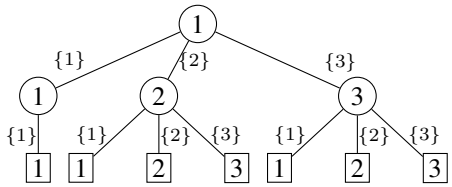

Figure 5: CAT $T_{3,2,1}$ for $g_{3,2,1}$ obtained by pruning $\tilde{T}_{3,2,1}$.

The composition function $g_{n,k,i}$ *is the state function* $g_{n,k,i} : \mathcal{A}(\mathcal{V}) \to \{1, \dots, n\}$ *defined as*

$$g_{n,0,i}(\alpha) = i$$
$$g_{n,k+1,i}(\alpha) = g_{n,k,\alpha(i)}(\alpha) \text{ for } k \geq 0$$

Function f_n is the composition function for the special case $k = 3$ and $i = 1$ and is thus defined as $f_n = g_{n,3,1}$.

Observe that $g_{n,0,i}(\alpha) = i$, $g_{n,1,i}(\alpha) = \alpha(i)$, and $g_{n,2,i}(\alpha) = \alpha(\alpha(i))$. In general, $g_{n,k,i}(\alpha) = \alpha^{\circ k}(i)$, where $\alpha^{\circ k}$ is the k-fold composition of α.

In particular, $f_n(\alpha) = \alpha(\alpha(\alpha(1)))$.

Theorem 7. *The function f_n can be represented by a CAT with size at most $2n^3 + n^2 + n$.*

Proof. For $g_{n,k,i}$, we define the "pseudo-CAT" $\tilde{T}_{n,k,i}$ over $\mathcal{V}_n$ and codomain $\{1, \dots, n\}$ as $\tilde{T}_{n,0,i} = i$ and $\tilde{T}_{n,k+1,i} = \langle i, \{\langle \{1\}, \tilde{T}_{n,k,1} \rangle, \dots, \langle \{n\}, \tilde{T}_{n,k,n} \rangle\} \rangle$ for $k \geq 0$. Figure 4 illustrates this construction for the example of $g_{3,2,1}$.

$\tilde{T}_{n,k,i}$ is a CAT except that some of its splits operate on the domain $\{1, \dots, n\}$ for a variable i whose domain has already been restricted to a singleton $\{j\}$ further up the tree. The left child of the root in Figure 4 shows this for $i = j = 1$. Variable 1 has already been reduced to the domain $\{1\}$, so we cannot have children for the other values 2 and 3. We convert the pseudo-CAT into a CAT $T_{n,k,i}$ by pruning any such children. Figure 5 illustrates this for the example.

To see that $[\![T_{n,k,i}]\!] = g_{n,k,i}$, consider an arbitrary $n \in \mathbb{N}_1$ and $\alpha \in \mathcal{A}(\mathcal{V}_n)$. We prove $[\![T_{n,k,i}]\!](\alpha) = g_{n,k,i}(\alpha)$ for all $i \in \{1, \dots, n\}$ with an induction over k. For the basis $k = 0$, we have $[\![T_{n,0,i}]\!](\alpha) = i = g_{n,0,i}(\alpha)$ for all $i \in \{1, \dots, n\}$. As induction hypothesis (IH), we use $[\![T_{n,k,i}]\!](\alpha) = g_{n,k,i}(\alpha)$ for all $i \in \{1, \dots, n\}$. For the inductive step from k to $k + 1$, we see that $[\![T_{n,k+1,i}]\!](\alpha) = [\![T_{n,k,\alpha(i)}]\!](\alpha) \overset{\text{IH}}{=} g_{n,k,\alpha(i)}(\alpha) = g_{n,k+1,i}(\alpha)$.

Regarding the representation size, we compute $\|\tilde{T}_{n,k,i}\|$ for the pseudo-CAT as if it were a CAT. Because $T_{n,k,i}$ is obtained from $\tilde{T}_{n,k,i}$ by pruning the tree, this gives an upper bound for $\|T_{n,k,i}\|$.

We have $\|\tilde{T}_{n,0,i}\| = 1$. For $k \geq 1$, we have $\|\tilde{T}_{n,k,i}\| = |dom(i)| + \sum_{j=1}^{n} \|\tilde{T}_{n,k-1,j}\| = n + \sum_{j=1}^{n} \|\tilde{T}_{n,k-1,j}\|$. Since the representation size of $\tilde{T}_{n,k,j}$ only depends on n and k but not on j, we can simplify this to $\|\tilde{T}_{n,k,i}\| = n + n\|\tilde{T}_{n,k-1,i}\|$. Solving this recurrence, we get $\|\tilde{T}_{n,k,i}\| = n^k + \sum_{j=1}^{k} n^j$ for all $k \geq 1$. For the CAT $T_{n,3,1}$ representing f_n, we thus have $\|T_{n,3,1}\| \leq n^3 + \sum_{j=1}^{3} n^j = 2n^3 + n^2 + n$. □

In the following we prove that all merge-and-shrink representations of f_n are (at least) exponentially large in n.

We begin with the following definition for state functions f, which does not depend on how f is represented.

Definition 13 (distinguish)**.** *Let f be a state function for the variable space $\mathcal{V} = \langle V, dom \rangle$ and let $U \subseteq V$. We say that:*

- *f* distinguishes *assignments $\alpha, \beta \in \mathcal{A}(U)$ if there exists an assignment $\gamma \in \mathcal{A}(V \setminus U)$ with $f(\alpha \cup \gamma) \neq f(\beta \cup \gamma)$.*
- *f* distinguishes *a set of assignments $A \subseteq \mathcal{A}(U)$ if f distinguishes all assignments $\alpha, \beta \in A$ with $\alpha \neq \beta$.*

In the following, we consider MSRs as trees with merges as inner nodes and atomic MSRs as leaves. For an MSR M, we write $V(M)$ for the set of all variables that occur in its leaves. We say that M has the variables $V(M)$.

Note that, from the definition of MSRs, the children of merges have disjoint sets of variables. This restricts the information flow within an MSR. If a sub-MSR M' (subtree) of an MSR M has the variables U, then the function $[\![M']\!]$ must capture all information about the assignment to U that is needed to compute $[\![M]\!]$. The following theorem formalizes this observation.

Theorem 8. *Let M be an MSR with sub-MSR M'. Let $U = V(M')$. If $[\![M]\!]$ distinguishes a set of assignments $A \subseteq \mathcal{A}(U)$, then $\|M\| \geq \|M'\| \geq |A|$.*

Proof. From the inductive definition of the functions represented by MSRs, if M' is a sub-MSR of M and α and β are assignments to $V(M')$ with $[\![M']\!](\alpha) = [\![M']\!](\beta)$, then $[\![M]\!](\alpha \cup \gamma) = [\![M]\!](\beta \cup \gamma)$ for all assignments γ to the remaining variables. It follows that $[\![M]\!]$ does not distinguish α and β. By contraposition, whenever $[\![M]\!]$ distinguishes α and β, we must have $[\![M']\!](\alpha) \neq [\![M']\!](\beta)$.

Hence, if $[\![M]\!]$ distinguishes a set of assignments to U of size N, $[\![M']\!]$ must map to at least N different values, which implies that the table of M' has size at least N and hence $\|M'\| \geq N$. $\|M\| \geq \|M'\|$ is obvious because M' is a sub-MSR of M. □

This idea is the merge-and-shrink equivalent of the *Sieling-Wegener bound* for the minimal size of BDDs (Sieling and Wegener 1993) and the *Myhill-Nerode theorem* on the size of minimal deterministic finite automata (e.g., Hopcroft, Motwani, and Ullman 2001, Section 4.4). We will use it to show an exponential lower bound on the size of MSRs for f_n. Towards this end, we first prove two technical lemmas about f_n.

For $X, Y \subseteq V_n$, define $\mathcal{A}(X, Y)$ as the set of assignments to X that map only to the set Y, i.e., $\mathcal{A}(X, Y) = \{\alpha \in \mathcal{A}(X) \mid \alpha(v) \in Y \text{ for all } v \in X\}$. (Recall that in the variable space $\mathcal{V}_n$, $V_n = \{1, \ldots, n\}$ is both the set of variables and the domain of these variables.)

Lemma 1. *Let $U \subseteq V_n$ with $1 \notin U$ and $|\overline{U}| \geq 2$, where $\overline{U} = V_n \setminus U$. Then f_n distinguishes the set $\mathcal{A}(U, \overline{U})$.*

Proof. Consider arbitrary $\alpha, \beta \in \mathcal{A}(U, \overline{U})$ with $\alpha \neq \beta$. We must show that f_n distinguishes α and β by constructing some $\gamma \in \mathcal{A}(\overline{U})$ with $f_n(\alpha \cup \gamma) \neq f_n(\beta \cup \gamma)$.

Consider any $j \in U$ with $\alpha(j) \neq \beta(j)$, which exists because $\alpha \neq \beta$. Let z be any variable in $\overline{U} \setminus \{1\}$, which exists because $|\overline{U}| \geq 2$.

Let $\gamma \in \mathcal{A}(\overline{U})$ be any assignment with $\gamma(1) = z$ and $\gamma(z) = j$.

For $\sigma = \alpha \cup \gamma$ or $\sigma = \beta \cup \gamma$, we get:

$$\begin{aligned} f_n(\sigma) &= \sigma(\sigma(\sigma(1))) \\ &= \sigma(\sigma(z)) && \text{from } \gamma(1) = z \\ &= \sigma(j) && \text{from } \gamma(z) = j \end{aligned}$$

and hence $f_n(\alpha \cup \gamma) = \alpha(j)$ and $f_n(\beta \cup \gamma) = \beta(j)$. Because $\alpha(j) \neq \beta(j)$, we get $f_n(\alpha \cup \gamma) \neq f_n(\beta \cup \gamma)$, concluding the proof. □

Lemma 2. *Let $U \subseteq V_n$ with $1 \in U$. Let $z \in \overline{U}$, where $\overline{U} = V_n \setminus U$. Then f_n distinguishes the set $\{\alpha \in \mathcal{A}(U, \overline{U}) \mid \alpha(1) = z\}$.*

Proof. Consider arbitrary $\alpha, \beta \in \mathcal{A}(U, \overline{U})$ with $\alpha(1) = \beta(1) = z$ and $\alpha \neq \beta$. As in the previous lemma, we must construct $\gamma \in \mathcal{A}(\overline{U})$ with $f_n(\alpha \cup \gamma) \neq f_n(\beta \cup \gamma)$.

Consider a $j \in U$ with $\alpha(j) \neq \beta(j)$ and let $\gamma \in \mathcal{A}(\overline{U})$ be any assignment with $\gamma(z) = j$.

For $\sigma = \alpha \cup \gamma$ or $\sigma = \beta \cup \gamma$, we get:

$$\begin{aligned} f_n(\sigma) &= \sigma(\sigma(\sigma(1))) \\ &= \sigma(\sigma(z)) && \text{from } \alpha(1) = \beta(1) = z \\ &= \sigma(j) && \text{from } \gamma(z) = j \end{aligned}$$

and hence $f_n(\alpha \cup \gamma) = \alpha(j)$ and $f_n(\beta \cup \gamma) = \beta(j)$. Because $\alpha(j) \neq \beta(j)$, we get $f_n(\alpha \cup \gamma) \neq f_n(\beta \cup \gamma)$, concluding the proof. □

To put the pieces together, we need one more technical result that shows that every MSR M has a sub-MSR M' such that $V(M')$ is "not too small and not too large".

Theorem 9. *Let M be an MSR with $|V(M)| = n$. Then M has a sub-MSR M' with $\lfloor n/3 \rfloor \leq |V(M')| \leq \lceil 2n/3 \rceil$.*

Proof. Descend downwards from the root of M, always moving into the larger subtree until the current subtree has the required number of variables. This approaches the target size interval from above and can never undershoot it: if the current tree has more than $\lceil 2n/3 \rceil$ variables, it cannot be the case that both children have fewer than $\lfloor n/3 \rfloor$ variables. □

We can now prove the lower bound.

Theorem 10. *For $n \geq 6$, if M is an MSR with $[\![M]\!] = f_n$, then $\|M\| \geq \lfloor n/3 \rfloor^{\lfloor n/3 \rfloor - 1}$.*

Proof. It is easy to see that we must have $V(M) = V_n$ because f_n depends on all its inputs. From Theorem 9, M has a sub-MSR M' with $\lfloor n/3 \rfloor \leq |V(M')| \leq \lceil 2n/3 \rceil$. Set $U = V(M')$ and $\overline{U} = V_n \setminus U$ and apply either Lemma 1 or 2 (using an arbitrary z), depending on $1 \notin V(M')$ or $1 \in V(M')$. Observe that $|U| \leq \lceil 2n/3 \rceil$ implies $|\overline{U}| \geq \lfloor n/3 \rfloor$. Because $n \geq 6$, this also shows that the condition $|\overline{U}| \geq 2$ in Lemma 1 is satisfied and that a $z \in \overline{U}$ exists for Lemma 2.

The lemmas provide us with a distinguished set that we can use in Theorem 8. Its size is $|\overline{U}|^{|U|}$ for Lemma 1 and $|\overline{U}|^{|U|-1}$ for Lemma 2, both of which are greater or equal to $\lfloor n/3 \rfloor^{\lfloor n/3 \rfloor - 1}$. □

We remark that the bound from this theorem is at least exponential. For example, for $n \geq 48$ we have $\lfloor n/3 \rfloor \geq 16$ and $\lfloor n/3 \rfloor - 1 \geq n/4$ and can very loosely bound $\lfloor n/3 \rfloor^{\lfloor n/3 \rfloor - 1} \geq 16^{n/4} = (16^{1/4})^n = 2^n$.

In summary, there exist polynomial-size Cartesian abstraction trees that represent the functions f_n (Theorem 7), while MSRs representing f_n have at least exponential size in n (Theorem 10). The following result immediately follows.

Theorem 11. *$\mathcal{CAT}$ cannot be compactly compiled into $\mathcal{MSR}$.*

$\mathcal{MSR}$ Cannot be Compiled into $\mathcal{CAT}$

For our final result, we show that merge-and-shrink representations cannot be compactly compiled into Cartesian abstraction trees. The proof for this result is much less involved than for the converse direction and reuses the parity functions considered earlier (Definition 11).

Specifically, we consider the parity functions $p_{n,2}$, where all variables have domain $\{0, 1\}$ and we need to determine if the sum of values in an assignment is odd. We show that polynomial-sized MSRs exist for $p_{n,2}$, whereas every $\mathcal{CAT}$ for $p_{n,2}$ is exponentially large.

Definition 14. *The merge-and-shrink representation P_n is the MSR over V_n with codomain $\{0, 1\}$ that is inductively defined as follows:*

- *P_1 is the atomic MSR $I_1 = \langle v_1, \{0 \mapsto 0, 1 \mapsto 1\}\rangle$.*
- *For $i \in \{2, \ldots, n\}$, P_i is the merge MSR $\langle P_{i-1}, I_i, tbl\rangle$, where I_i is the atomic MSR $\langle v_i, \{0 \mapsto 0, 1 \mapsto 1\}\rangle$ and the table for P_i is $tbl = \{\langle 0, 0\rangle \to 0, \langle 0, 1\rangle \to 1, \langle 1, 0\rangle \to 1, \langle 1, 1\rangle \to 0\}$.*

Figure 6 illustrates the construction for the case $n = 4$. We now prove that P_n represents $p_{n,2}$.

Theorem 12. $[\![P_n]\!] = p_{n,2}$.

Proof. Each atomic MSR I_i for variable v_i computes the value of v_i, $[\![I_i]\!](\alpha) = \alpha(v_i)$, which correctly encodes the parity of v_i. By induction over i, we see that each merge MSR P_i correctly computes the parity of an assignment α to $\{v_1, \ldots, v_i\}$ from the parity of $\alpha|_{\{v_1,\ldots,v_{i-1}\}}$ and $\alpha(v_i)$. □

MSRs representing parity functions are very compact.

Theorem 13. *The parity function $p_{n,2}$ can be represented by an MSR that has linear size in n.*

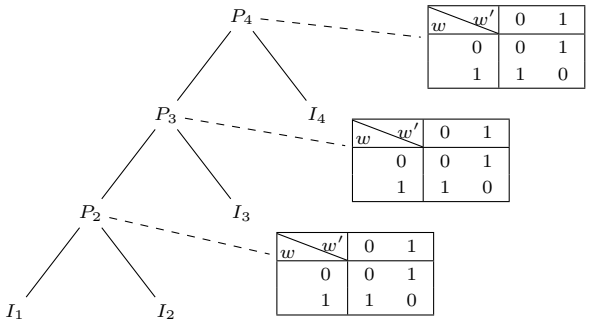

Figure 6: MSR P_4 for $p_{4,2}$. The leaves are atomic MSRs $I_i = \langle v_i, \{0 \mapsto 0, 1 \mapsto 1\}\rangle$.

Proof. P_n represents $p_{n,2}$. It has n leaves, each contributing 2 to the size, and $n - 1$ inner nodes, each contributing 4. We obtain $\|P_n\| = n \cdot 2 + (n - 1) \cdot 4 = 6n - 4 \in \Theta(n)$. □

In contrast, Cartesian abstraction trees for $p_{n,2}$ are exponentially large in n.

Theorem 14. *Every CAT representing $p_{n,2}$ has a representation size of at least 2^n.*

Proof. Every split in a CAT T partitions the variable space over which it is defined into the variable spaces over which its children are defined. It follows that the variable spaces over which the leaves of T are defined form a partition of the variable space of T. For every assignment α, $[\![T]\!](\alpha)$ is then the value of the leaf whose variable space includes α.

Consider a leaf L of such a CAT T representing $p_{n,2}$, and let $\mathcal{V}_L$ be its variable space. If $|\mathcal{A}(\mathcal{V}_L)| > 1$, there must be a variable v_i whose domain in $\mathcal{V}_L$ has 2 values. Then $\mathcal{A}(\mathcal{V}_L)$ contains two assignments α and α' that differ exactly on the variable v_i. Because α and α' belong to the same leaf, we have $[\![T]\!](\alpha) = [\![T]\!](\alpha')$, but from the definition of $p_{n,2}$, we have $p_{n,2}(\alpha) \neq p_{n,2}(\alpha')$ if α and α' differ on exactly one variable, contradicting $[\![T]\!] = p_{n,2}$. Thus no leaf with $|\mathcal{A}(\mathcal{V}_L)| > 1$ exists, and hence $|\mathcal{A}(\mathcal{V}_L)| = 1$ for all leaves.

Because the variable spaces of the leaves partition $\mathcal{A}(V_n)$ and $|\mathcal{A}(V_n)| = 2^n$, it follows that T has 2^n leaves, which shows $\|T\| \geq 2^n$, concluding the proof. □

The following result is an immediate consequence of Theorems 13 and 14.

Theorem 15. *$\mathcal{MSR}$ cannot be compactly compiled into $\mathcal{CAT}$.*

Conclusion

We studied the relative compactness of the major classes of abstractions in classical planning. Because memory usage is the limiting aspect when using abstraction heuristics, such a study can show which families of abstraction heuristics make contributions that are fundamentally different from those of other families.

Our study confirms the expected result that, from the perspective of compact representation, domain abstraction representations are strictly more powerful than pattern

databases for projections, and that both Cartesian abstractions and merge-and-shrink representations are strictly more powerful than domain abstraction representations.

Comparing Cartesian abstractions to merge-and-shrink representations, our results prove that their strengths are complementary, with neither being able to compactly represent the other in general, closing an open question posed by Seipp and Helmert (2018). This incompilability raises the question if there exists a natural class of abstraction representations that subsumes both Cartesian abstraction and merge-and-shrink, combining their complementary strengths.

Intuitively, the reason why Cartesian abstraction trees cannot be efficiently compiled into merge-and-shrink representations is that they can decide dynamically which variables to consider in any given context, rather than using a static "variable tree" decomposition (*merge strategy*). A generalization of merge-and-shrink representations that permits a more dynamic form of merge strategy could in principle generalize both classes of representations.

However, such a representation would be so general that it appears challenging to come up with a good approach for finding good representatives of such abstractions with an automated algorithm. This is somewhat comparable to the ubiquitous use of ordered BDDs in the BDD literature even though it is well-known that BDDs without a fixed variable order (so-called "free BDDs") are more powerful (**?**).

Ultimately, the best representations are not necessarily the most general ones but the ones that offer the best trade-offs between expressiveness and efficient algorithms for synthesis and manipulation. In this regard, both Cartesian abstractions and merge-and-shrink representations have earned their place in the state of the art.

Acknowledgments

This work was funded by the Swiss National Science Foundation (SNSF) as part of the project "Unifying the Theory and Algorithms of Factored State-Space Search" (UTA).

References

Bryant, R. E. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8): 677–691.

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

Edelkamp, S. 2001. Planning with Pattern Databases. In Cesta, A.; and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 84–90. AAAI Press.

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.; 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.; Röger, G.; and Sievers, S. 2015. On the Expressive Power of Non-Linear Merge-and-Shrink Representations. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 106–114. AAAI Press.

Hernádvölgyi, I. T.; and Holte, R. C. 2000. Experiments with Automatically Created Memory-Based Heuristics. In Choueiry, B. Y.; and Walsh, T., eds., *Proceedings of the 4th International Symposium on Abstraction, Reformulation and Approximation (SARA 2000)*, volume 1864 of *Lecture Notes in Artificial Intelligence*, 281–290. Springer-Verlag.

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

Karp, R. M. 1972. Reducibility among combinatorial problems. In Miller, R. E.; and Thatcher, J. W., eds., *Complexity of Computer Computations*, 85–103. Plenum Press.

Kreft, R.; Büchner, C.; Sievers, S.; and Helmert, M. 2023. Computing Domain Abstractions for Optimal Classical Planning with Counterexample-Guided Abstraction Refinement. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*, 221–226. AAAI Press.

Seipp, J.; and Helmert, M. 2013. Counterexample-guided Cartesian Abstraction Refinement. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 347–351. AAAI Press.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *Journal of Artificial Intelligence Research*, 62: 535–577.

Sieling, D.; and Wegener, I. 1993. NC-Algorithms for Operations on Binary Decision Diagrams. *Parallel Processing Letters*, 3(1): 3–12.

Sievers, S.; and Helmert, M. 2021. Merge-and-Shrink: A Compositional Theory of Transformations of Factored Transition Systems. *Journal of Artificial Intelligence Research*, 71: 781–883.