A Knowledge Compilation Map for Ordered Real-Valued Decision Diagrams
Full Version — rev. 3

Hélène Fargier[1] and **Pierre Marquis**[2] and **Alexandre Niveau**[3] and **Nicolas Schmidt**[1,2]
[1] IRIT-CNRS, Univ. Paul Sabatier, Toulouse, France
[2] CRIL-CNRS, Univ. Artois, Lens, France
[3] GREYC-CNRS, Univ. Caen, France

Abstract

Valued decision diagrams (VDDs) are data structures that represent functions mapping variable-value assignments to non-negative real numbers. They prove useful to compile cost functions, utility functions, or probability distributions. While the complexity of some queries (notably optimization) and transformations (notably conditioning) on VDD languages has been known for some time, there remain many significant queries and transformations, such as the various kinds of cuts, marginalizations, and combinations, the complexity of which has not been identified so far. This paper contributes to filling this gap and completing previous results about the time and space efficiency of VDD languages, thus leading to a knowledge compilation map for real-valued functions. Our results show that many tasks that are hard on valued CSPs are actually tractable on VDDs.

1 Introduction

Valued decision diagrams (VDDs) are data structures that represent multivariate functions f having a set $\mathcal{V}$ of valuations (often a subset of $\mathbb{R}^+$) as codomain; such functions are typically cost functions, utility functions, or probability distributions, and as such, are considered in a number of AI applications. Among the various tasks of interest when dealing with such functions are *optimization* queries: find an assignment of the variables leading to an optimal valuation; find a value of a given variable that can be extended to an optimal assignment; etc. Optimization queries are particularly valuable when combined with the *conditioning* transformation, which derives a representation of a restriction of f obtained by assigning some of its variables. For instance, the following task can be handled by combining optimization with conditioning: in configuration problems, when f represents a cost function mapping each assignment (say, a car) to its price, output the cheapest car with seven seats; or, when f represents a probability distribution linking diseases to symptoms, return the most probable explanation (the most probable disease given a set of symptoms).

Many other data structures have been defined for representing such multivariate functions f, valued CSPs (Schiex, Fargier, and Verfaillie 1995), GAI nets (Bacchus and Grove 1995), and Bayes nets (Pearl 1989) being among the best-known. However, they are not adapted to the aforementioned requests when guaranteed response times are required (as is the case in Web-based applications): optimization is indeed NP-hard on CSPs, GAI nets, and Bayes nets.

Contrastingly, optimization is a tractable query on VDDs; and conditioning is tractable as well. This explains why several families of VDDs have been defined and studied in the past twenty years. These include the language ADD of algebraic decision diagrams (Bahar et al. 1993), the language AADD of affine algebraic decision diagrams (Tafertshofer and Pedram 1997; Sanner and McAllester 2005), and the language SLDD of semiring-labeled decision diagrams (Wilson 2005). Actually, SLDD is itself a family of languages $\mathtt{SLDD}_\otimes$, parameterized by a binary operator $\otimes$, yielding in particular $\mathtt{SLDD}_+$, or equivalently EVBDD (Lai and Sastry 1992; Lai, Pedram, and Vrudhula 1996; Amilhastre, Fargier, and Marquis 2002), when $\otimes$ is $+$, and $\mathtt{SLDD}_\times$ when $\otimes$ is $\times$.

There nevertheless exist many queries and transformations of interest that do not amount to a combination of conditioning and optimization. Consider for instance the following request: "tell me whether among the 'cheap' cars (say, with a price lower than 10,000 euros) of this given type, there is one that has seven seats". It requires one to focus on the set of 'cheap' cars, which is neither optimization nor conditioning. Similarly, some transformations, such as projection on variables of interest, or its dual, variable elimination (e.g., forgetting or marginalization), also being of tremendous value (e.g., to solve the "posterior marginal" problem (PM) in Bayes nets), are not reducible to optimization and conditioning. This is also the case of combination transformations (a.k.a. 'apply' operations), which given the representations of two functions f and g, compute a representation of $f \odot g$, $\odot$ being an associative and commutative operator on $\mathcal{V}$ (e.g., addition or product when $\mathcal{V} = \mathbb{R}^+$). Such transformations are very important, if only for incrementally generating representations in a bottom-up way.

The knowledge compilation (KC) map (Darwiche and Marquis 2002) identifies the complexity of requests (i.e., queries and transformations) over many propositional languages, as well as their relative succinctness. However, it has been drawn for the specific case of Boolean functions; although it has been extended in several directions, as of yet no such map exists for the VDD languages. It has been shown that AADD is strictly more succinct than ADD (Sanner and McAllester 2005), and the succinctness picture has later been completed (Fargier, Marquis, and Schmidt 2013): AADD is strictly more succinct than both SLDD_+ and $\text{SLDD}_\times$, and both SLDD_+ and $\text{SLDD}_\times$ are in turn strictly more succinct than ADD. To complete the map, one now needs to provide the set of requests that each language satisfies; in order words, for each request, and for each language among ADD, SLDD_+, $\text{SLDD}_\times$, and AADD, to either find a polynomial-time algorithm computing the request, or prove that no such algorithm exists unless $\mathsf{P} = \mathsf{NP}$.

This is the main goal of this paper, which is organized as follows. The next section presents the family of VDD languages, and the following one formally defines the queries and transformations used. Then complexity results are presented,[1] that establish whether each VDD language satisfies each query or transformation. This paves the way for a KC map for functions that are not essentially Boolean.

2 Valued Decision Diagrams

Preliminaries. Given a finite set $\mathcal{X} = \{x_1, \ldots, x_n\}$ of variables, each x_i ranging over a finite domain D_{x_i}, and any set $X \subseteq \mathcal{X}$, $\vec{x} = \{ \langle x_i, d_i \rangle \mid x_i \in X, d_i \in D_{x_i} \}$ denotes an assignment of the variables from X; D_X is the set of all such assignments (the Cartesian product of the domains of the variables in X). The concatenation of two assignments $\vec{x}$ and $\vec{y}$ of two disjoint subsets X and Y is an assignment of $X \cup Y$ denoted $\vec{x} \cdot \vec{y}$.

We consider functions f of variables from a subset $\text{Scope}(f) \subseteq \mathcal{X}$ to some set $\mathcal{V}$ (this paper often assumes $\mathcal{V} = \mathbb{R}^+$). The domain of f is denoted as $D_f = D_{\text{Scope}(f)}$. For any $Z \subseteq \text{Scope}(f)$, $f_{\vec{z}}$ denotes the *restriction* (or *semantic conditioning*) of f by $\vec{z}$, that is, the function on $\text{Scope}(f) \setminus Z$ such that for any $\vec{x} \in D_{\text{Scope}(f) \setminus Z}$, $f_{\vec{z}}(\vec{x}) = f(\vec{z} \cdot \vec{x})$.

Given a binary operator $\odot$ over $\mathcal{V}$ and two functions f and g sharing the same scope, $f \odot g$ is the function defined by $f \odot g\ (\vec{x}) = f(\vec{x}) \odot g(\vec{x})$. The $\odot$-projection of f on $Z \subseteq \text{Scope}(f)$ is defined by $f^{\odot, Z}(\vec{x}) = \bigodot_{\vec{y} \in D_{\text{Scope}(f) \setminus Z}} f_{\vec{y}}(\vec{x})$.

Slightly abusing notations, if X and Y are two disjoint sets of variables, $\text{Scope}(f) = X$, and $\vec{x} \cdot \vec{y}$ is an assignment of $X \cup Y$, then we assume that $f(\vec{x} \cdot \vec{y}) = f(\vec{x})$; in practice, we often consider complete assignments, i.e., $\vec{x} \in D_{\mathcal{X}}$.

Given $\succeq$ a reflexive and transitive relation on $\mathcal{V}$ (i.e., a preorder), of which we denote $\sim$ the symmetric part and $\succ$ the asymmetric part, we can define for any $\gamma \in \mathcal{V}$ the following sets, referred to as "cuts":

- $\mathit{CUT}^{\max}(f) = \{ \vec{x}^* \mid \forall \vec{x}, \neg(f(\vec{x}) \succ f(\vec{x}^*)) \}$;
- $\mathit{CUT}^{\min}(f) = \{ \vec{x}^* \mid \forall \vec{x}, \neg(f(\vec{x}) \prec f(\vec{x}^*)) \}$;
- $\mathit{CUT}^{\succeq\gamma}(f) = \{ \vec{x} \mid f(\vec{x}) \succeq \gamma \}$, and using similar definitions, $\mathit{CUT}^{\preceq\gamma}(f)$ and $\mathit{CUT}^{\sim\gamma}(f)$.

For instance, when optimizing is minimizing, $\mathit{CUT}^{\min}$ is the set of optimal assignments (e.g., the cheapest cars); $\mathit{CUT}^{\preceq\gamma}$ is the set of assignments satisfying the cut condition (e.g., the 'cheap' cars – those with a price $\preceq \gamma$).

A representation language over $\mathcal{X}$ w.r.t. a set $\mathcal{V}$ is a set of data structures equipped with an interpretation function that associates with each of them a mapping $f : D_{\mathcal{X}} \to \mathcal{V}$. This mapping is called the *semantics* of the data structure, and the data structure is a *representation* of the mapping.

Definition 2.1 (representation language; inspired from Gogic et al. (1995))**.** Given a valuation set $\mathcal{V}$, a *representation language* L over $\mathcal{X}$ w.r.t. $\mathcal{V}$, is a 4-tuple $\langle C_{\mathtt{L}}, \text{Var}_{\mathtt{L}}, f^{\mathtt{L}}, s_{\mathtt{L}} \rangle$, where:

- $C_{\mathtt{L}}$ is a set of data structures α (also referred to as L representations or "formulæ"),
- $\text{Var}_{\mathtt{L}} \colon C_{\mathtt{L}} \to 2^{\mathcal{X}}$ is a scope function associating with each L representation the subset of $\mathcal{X}$ it depends on,
- $f^{\mathtt{L}}$ is an interpretation function associating with each L representation α a mapping $f^{\mathtt{L}}_{\alpha}$ from the set of all assignments over $\text{Var}_{\mathtt{L}}(\alpha)$ to $\mathcal{V}$,
- $s_{\mathtt{L}}$ is a size function from $C_{\mathtt{L}}$ to $\mathbb{N}$ that provides the size of any L representation.

Two formulæ (possibly from different languages) are equivalent iff they have the same scope and semantics.

Valued decision diagrams. In the following, we consider representation languages based on data structures called *valued decision diagrams*: such diagrams target the representation of $\mathcal{V}$-valued functions by allowing their arcs and nodes to be labeled with values in some set $\mathcal{E}$ (generally, $\mathcal{E} = \mathcal{V}$, but as we will see, it is not always the case).

Definition 2.2 (valued decision diagram)**.** A *valued decision diagram* (VDD) over $\mathcal{X}$ w.r.t. $\mathcal{E}$ is a finite rooted DAG α, of which every internal node N is labeled with a variable $x \in \mathcal{X}$ and has a set $\text{Out}(N)$ of $|D_x|$ outgoing arcs, each arc $a \in \text{Out}(N)$ being labeled with a distinct value $v(a) \in D_x$. Arcs and leaves can be labeled with elements of $\mathcal{E}$: $\varphi(a)$ (resp. $\varphi(L)$) denotes the label of arc a (resp. of leaf L).
The size of a decision diagram α, denoted $|\alpha|$, is the size of the graph (its number of nodes and arcs) plus the sizes of all labels in it. VDD is the set of all valued decision diagrams.

In the following, we assume that the decision diagrams are *ordered*, i.e., a total, strict ordering $\rhd$ over $\mathcal{X}$ is chosen, and for each path from the root to a leaf in a VDD α, the associated sequence of internal node labels is required to be compatible w.r.t. this variable ordering (such diagrams are thus read-once). A path from the root to a leaf of α represents a (possibly partial) assignment of $\mathcal{X}$. Note that the structure is deterministic: an assignment $\vec{x}$ of $\mathcal{X}$ corresponds to at most one path $p_\alpha(\vec{x})$ in α.

A VDD α is *reduced* iff it does not contain any (distinct) isomorphic nodes.[2] A cache mechanism can be used

[1]For space reasons, proofs have been omitted from the published version of the paper; this full version includes all proofs in the appendix.

[2]Nodes N and M are isomorphic if they are labeled with the same variable x, and there exists a bijection B from $\text{Out}(N)$ onto $\text{Out}(M)$ such that $\forall a \in \text{Out}(N)$, a and $B(a)$ have the same end node, share the same value of D_x, and $\varphi(a) = \varphi(B(a))$.

to merge isomorphic nodes on the fly; this is why we assume in the following that the diagrams are reduced.

VDD languages. ADD, SLDD, and AADD are VDD languages, i.e., subsets of VDD. Each of them restricts the type of diagrams used (e.g., ADD allows $\mathcal{E}$-labels on leaves only), and defines how the diagram is to be interpreted. ADD, SLDD, and AADD thus differ syntactically (in the way diagrams are labeled) and semantically (in the way they are interpreted).

Definition 2.3 (ADD). ADD is the 4-tuple $\langle C_{\texttt{ADD}}, \mathrm{Var}_{\texttt{ADD}}, f^{\texttt{ADD}}, s_{\texttt{ADD}}\rangle$ where $C_{\texttt{ADD}}$ is the set of ordered VDDs α over $\mathcal{X}$ such that leaves are labeled with elements of $\mathcal{E} = \mathcal{V}$ (in general, $\mathcal{E} = \mathcal{V} = \mathbb{R}^+$), the arcs are not labeled, and $f^{\texttt{ADD}}$ is defined inductively as follows, for every assignment $\vec{x}$:

- if α is a leaf node L, then $f^{\texttt{ADD}}_\alpha(\vec{x}) = \varphi(L)$,
- otherwise, denoting N the root of α, $x \in \mathcal{X}$ its label, $d \in D_x$ such that $\langle x, d\rangle \in \vec{x}$, $a = \langle N, M\rangle$ the arc such that $v(a) = d$, and β the ADD formula rooted at node M in α, then $f^{\texttt{ADD}}_\alpha(\vec{x}) = f^{\texttt{ADD}}_\beta(\vec{x})$.

In the AADD framework of Sanner and McAllester (2005), the co-domain of the represented functions is $\mathcal{V} = \mathbb{R}^+$, but only one (unlabeled) leaf is allowed and the arcs are labeled with pairs of values from $\mathbb{R}^+$ (i.e., $\mathcal{E} = \mathbb{R}^+ \times \mathbb{R}^+ \neq \mathcal{V}$).

Definition 2.4 (AADD). AADD is the 4-tuple $\langle C_{\texttt{AADD}}, \mathrm{Var}_{\texttt{AADD}}, f^{\texttt{AADD}}, s_{\texttt{AADD}}\rangle$ where $C_{\texttt{AADD}}$ is the set of ordered VDDs α over $\mathcal{X}$ with a unique leaf L, and the arcs of which are labeled with pairs $\langle q, f\rangle$ in $\mathbb{R}^+ \times \mathbb{R}^+$. For normalization purposes, the root of α is also equipped with a pair $\langle q_0, f_0\rangle$ from $\mathbb{R}^+ \times \mathbb{R}^+$ (the "offset"). The semantics of the resulting VDD is given by, for every assignment $\vec{x}$, $f^{\texttt{AADD}}_\alpha(\vec{x}) = q_0 + (f_0 \times g^{\texttt{AADD}}_\alpha(\vec{x}))$, where $g^{\texttt{AADD}}_\alpha$ is defined inductively as follows:

- if α is the leaf node L, then $g^{\texttt{AADD}}_\alpha(\vec{x}) = 0$,
- otherwise, denoting N the root of α, $x \in \mathcal{X}$ its label, $d \in D_x$ such that $\langle x, d\rangle \in \vec{x}$, $a = \langle N, M\rangle$ the arc such that $v(a) = d$, $\varphi(a) = \langle q_a, f_a\rangle$, and β the formula rooted at node M in α, then $g^{\texttt{AADD}}_\alpha(\vec{x}) = q_a + (f_a \times g^{\texttt{AADD}}_\beta(\vec{x}))$.

The AADD framework is equipped with a normalization condition that makes (reduced) ordered AADD formulæ canonical. Canonicity is important because it ensures a unique representation for each subformula, which is a key for efficiently recognizing and caching them.

In the $\texttt{SLDD}_\otimes$ framework (Wilson 2005),[3] arcs (but not leaves) are labeled with elements in $\mathcal{E} = \mathcal{V}$, and $\langle \mathcal{E}, \otimes, 1_\otimes\rangle$ is assumed to be a commutative monoid: $f^{\texttt{SLDD}_\otimes}_\alpha(\vec{x})$ is the aggregation by $\otimes$ of the labels of the arcs along $p_\alpha(\vec{x})$.

Definition 2.5 ($\texttt{SLDD}_\otimes$). Given a commutative monoid $\langle \mathcal{E}, \otimes, 1_\otimes\rangle$, $\texttt{SLDD}_\otimes$ is the 4-tuple $\langle C_{\texttt{SLDD}_\otimes}, \mathrm{Var}_{\texttt{SLDD}_\otimes}, f^{\texttt{SLDD}_\otimes}, s_{\texttt{SLDD}_\otimes}\rangle$ where $C_{\texttt{SLDD}_\otimes}$ is the set of ordered VDDs α over $\mathcal{X}$ with a unique leaf L, such that $\varphi(L) = 1_\otimes$, and the arcs of which are labeled with elements of $\mathcal{E} = \mathcal{V}$, and $f^{\texttt{SLDD}_\otimes}$ is defined inductively as follows: for every assignment $\vec{x}$,

- if α is the leaf node L, then $f^{\texttt{SLDD}_\otimes}_\alpha(\vec{x}) = 1_\otimes$,
- otherwise, denoting N the root of α, $x \in \mathcal{X}$ its label, $d \in D_x$ such that $\langle x, d\rangle \in \vec{x}$, $a = \langle N, M\rangle$ the arc such that $v(a) = d$, and β the $\texttt{SLDD}_\otimes$ formula rooted at node M in α, $f^{\texttt{SLDD}_\otimes}_\alpha(\vec{x}) = \varphi(a) \otimes f^{\texttt{SLDD}_\otimes}_\beta(\vec{x})$.

We add to the root of α a value $\varphi_0 \in \mathcal{E}$ (the "offset"). The augmented interpretation function of α is $f_{\alpha,\varphi_0} = \varphi_0 \otimes f_\alpha$.

Two (ordered) monoids are particularly interesting, namely $\langle \mathbb{R}^+, +, 0\rangle$ (we call the corresponding language $\texttt{SLDD}_+$) and $\langle \mathbb{R}^+, \times, 1\rangle$ (we call the corresponding language $\texttt{SLDD}_\times$). Both $\texttt{SLDD}_+$ and $\texttt{SLDD}_\times$ formulæ have a normalization condition that provides these languages with the canonicity property; moreover, any $\texttt{SLDD}_+$ or $\texttt{SLDD}_\times$ formula can be transformed in linear time into an equivalent AADD formula. This also holds for the ADD language, targeting any of the $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, or AADD languages. Last, note that when variables are Boolean and $\mathcal{V} = \{0, 1\}$, any ADD, $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, or AADD formula can be transformed in linear time into an equivalent OBDD, and (obviously) reciprocally.

3 Queries and Transformations

We now define many queries and transformations of interest in the general framework of representation languages w.r.t. $\mathcal{V}$, that is, they are meaningful even when $\mathcal{V} \neq \mathbb{R}^+$.

Definition 3.1 (queries). Let L denote a representation language over $\mathcal{X}$ w.r.t. a set $\mathcal{V}$ totally ordered by some $\succeq$.

- L satisfies optimization $\mathbf{OPT}_{\max}$ (resp. $\mathbf{OPT}_{\min}$) iff there exists a polynomial-time algorithm that maps every L formula α to $\max_{\vec{x} \in D_f} f^{\texttt{L}}_\alpha(\vec{x})$ (resp. $\min_{\vec{x} \in D_f} f^{\texttt{L}}_\alpha(\vec{x})$).
- L satisfies equivalence $\mathbf{EQ}$ iff there exists a polynomial-time algorithm that maps every pair of L formulæ α and β to 1 if $f^{\texttt{L}}_\alpha = f^{\texttt{L}}_\beta$, and to 0 otherwise.
- L satisfies sentential entailment $\mathbf{SE}$ iff there exists a polynomial-time algorithm that maps every pair of L formulæ α and β to 1 if $\forall \vec{x}, f^{\texttt{L}}_\alpha(\vec{x}) \succeq f^{\texttt{L}}_\beta(\vec{x})$, and to 0 otherwise.
- L satisfies partial upper (resp. lower, resp. level) consistency $\mathbf{CO}_{\succeq\gamma}$ (resp $\mathbf{CO}_{\preceq\gamma}$, resp. $\mathbf{CO}_{\sim\gamma}$) iff there exists a polynomial-time algorithm that maps every $\gamma \in \mathcal{V}$ and every L formula α to 1 if $\exists \vec{x}, f^{\texttt{L}}_\alpha(\vec{x}) \succeq \gamma$ (resp. $f^{\texttt{L}}_\alpha(\vec{x}) \preceq \gamma$, resp. $f^{\texttt{L}}_\alpha(\vec{x}) \sim \gamma$) and to 0 otherwise.
- L satisfies partial upper (resp. lower, resp. level) validity $\mathbf{VA}_{\succeq\gamma}$ (resp $\mathbf{VA}_{\preceq\gamma}$, resp. $\mathbf{VA}_{\sim\gamma}$) iff there exists a polynomial-time algorithm that maps every $\gamma \in \mathcal{V}$ and every L formula α to 1 if $\forall \vec{x}, f^{\texttt{L}}_\alpha(\vec{x}) \succeq \gamma$ (resp. $f^{\texttt{L}}_\alpha(\vec{x}) \preceq \gamma$, resp. $f^{\texttt{L}}_\alpha(\vec{x}) \sim \gamma$) and to 0 otherwise.
- L satisfies max-model enumeration $\mathbf{ME}_{\max}$ iff there exists a polynomial p and an algorithm that outputs, for every L formula α, all the elements of $CUT^{\max}(f^{\texttt{L}}_\alpha)$ in time $p(|\alpha|, |CUT^{\max}(f^{\texttt{L}}_\alpha)|)$,
- L satisfies max-model counting $\mathbf{CT}_{\max}$ iff there exists a polynomial-time algorithm that outputs, for every L formula α, the number of elements in $CUT^{\max}(f^{\texttt{L}}_\alpha)$.

[3] Note that our definition of SLDD differs from the original one in two ways: (i) we consider only ordered diagrams; and (ii) we use a commutative monoid instead of a commutative semiring, since the second operation of the semiring does not take part in the data structure (Fargier, Marquis, and Schmidt 2013).

- $\mathtt{L}$ satisfies $\max$-model extraction $\mathbf{MX}_{\max}$ iff there exists a polynomial-time algorithm that maps every $\mathtt{L}$ formula α to an element $\vec{x}$ of $CUT^{\max}(f^{\mathtt{L}}_{\alpha})$.

We define $\mathbf{ME}_{\min}$, $\mathbf{ME}_{\succeq\gamma}$, $\mathbf{ME}_{\preceq\gamma}$, $\mathbf{ME}_{\sim\gamma}$ (resp. $\mathbf{MX}_{\min}$, $\mathbf{MX}_{\succeq\gamma}$, $\mathbf{MX}_{\preceq\gamma}$, $\mathbf{MX}_{\sim\gamma}$; resp. $\mathbf{CT}_{\min}$, $\mathbf{CT}_{\succeq\gamma}$, $\mathbf{CT}_{\preceq\gamma}$, $\mathbf{CT}_{\sim\gamma}$) in the same way as $\mathbf{ME}_{\max}$ (resp. $\mathbf{MX}_{\max}$; resp. $\mathbf{CT}_{\max}$), using the sets $CUT^{\min}(f^{\mathtt{L}}_{\alpha})$, $CUT^{\succeq\gamma}(f^{\mathtt{L}}_{\alpha})$, $CUT^{\preceq\gamma}(f^{\mathtt{L}}_{\alpha})$, $CUT^{\sim\gamma}(f^{\mathtt{L}}_{\alpha})$ instead of $CUT^{\max}(f^{\mathtt{L}}_{\alpha})$.

The $\mathbf{MX}$ and $\mathbf{ME}$ families of queries are crucial in Bayesian reasoning and in interactive configuration. They capture for instance the computation of a (the) most probable explanation(s) ($\mathbf{MX}_{\max}$), or cheapest configuration(s) ($\mathbf{MX}_{\min}$); counting is also useful for such applications, e.g., for characterizing the number of cars that are 'cheap' ($\mathbf{CT}_{\preceq\gamma}$), or the number of diseases that are likely enough to be considered ($\mathbf{CT}_{\succeq\gamma}$). Other queries like consistency and validity, as well as $\mathbf{EQ}$ and $\mathbf{SE}$, are useful for many reasoning problems (e.g., when pieces of information are encoded into weighted knowledge bases); they extend the corresponding queries defined for (Boolean) NNF formulæ.

Definition 3.2 (transformations)**.** Let $\mathtt{L}$ denote a representation language over $\mathcal{X}$ w.r.t. $\mathcal{V}$, and $\odot$ an associative and commutative binary operator over $\mathcal{V}$.

- $\mathtt{L}$ satisfies conditioning $\mathbf{CD}$ iff there exists a polynomial-time algorithm that maps every $\mathtt{L}$ formula α, every $X \subseteq \mathcal{X}$, and every $\vec{x} \in D_X$ to an $\mathtt{L}$ representation of $f^{\mathtt{L}}_{\alpha,\vec{x}}$.
- $\mathtt{L}$ satisfies bounded $\odot$-combination $\odot\mathbf{BC}$ iff there exists a polynomial-time algorithm that maps every pair of $\mathtt{L}$ formulæ α and β to an $\mathtt{L}$ representation of $f^{\mathtt{L}}_{\alpha} \odot f^{\mathtt{L}}_{\beta}$.
- $\mathtt{L}$ satisfies $\odot$-combination $\odot\mathbf{C}$ iff there exists a polynomial-time algorithm that maps every set of $\mathtt{L}$ formulæ $\{\alpha_1, \ldots, \alpha_n\}$ to an $\mathtt{L}$ representation of $\bigodot_{i=1}^{n} f^{\mathtt{L}}_{\alpha_i}$.
- $\mathtt{L}$ satisfies variable (resp. single variable) $\odot$-elimination $\odot\mathbf{Elim}$ (resp. $\mathbf{S}\odot\mathbf{Elim}$) iff there exists a polynomial-time algorithm that maps every $\mathtt{L}$ formula α and every subset $X \subseteq \mathcal{X}$ of variables (resp every singleton $X \subseteq \mathcal{X}$) to an $\mathtt{L}$ representation of $\bigodot_{x\in X} \bigodot_{\vec{x}\in D_x} f^{\mathtt{L}}_{\alpha,\vec{x}}$.
- $\mathtt{L}$ satisfies single bounded-variable $\odot$-elimination $\mathbf{SB}\odot\mathbf{Elim}$ iff there exists a polynomial p and an algorithm that maps every $\mathtt{L}$ formula α and every $x \in \mathcal{X}$ to an $\mathtt{L}$ representation of $\bigodot_{\vec{x}\in D_x} f^{\mathtt{L}}_{\alpha,\vec{x}}$ in time $p(|\alpha|^{|D_x|})$.
- $\mathtt{L}$ satisfies single variable $\odot$-marginalization ($\odot\mathbf{Marg}$) iff there exists a polynomial-time algorithm that maps every $\mathtt{L}$ formula α and every $x \in \mathcal{X}$ to an $\mathtt{L}$ representation of $\bigodot_{y\in\mathcal{X}\setminus\{x\}} \bigodot_{\vec{y}\in D_y} f^{\mathtt{L}}_{\alpha,\vec{y}}$.
- $\mathtt{L}$ satisfies γ-cut up $\mathbf{CUT}_{\succeq\gamma}$, w.r.t. a preorder $\succeq$ on $\mathcal{V}$, iff there exists a polynomial-time algorithm that maps every $\mathtt{L}$ formula α and every $\gamma, a, b \in \mathcal{V}$ such that $a \succ b$, to an $\mathtt{L}$ representation of the function g defined by $g(\vec{x}) = a$ if $\vec{x} \in CUT^{\succeq\gamma}(f_\alpha)$, and $g(\vec{x}) = b$ otherwise.

We define $\mathbf{CUT}_{\preceq\gamma}$, $\mathbf{CUT}_{\sim\gamma}$, $\mathbf{CUT}_{\max}$, and $\mathbf{CUT}_{\min}$ in the same way, from the sets $CUT^{\preceq\gamma}(f_\alpha)$, $CUT^{\sim\gamma}(f_\alpha)$, $CUT^{\max}(f_\alpha)$, and $CUT^{\min}(f_\alpha)$, respectively.

The classical forgetting of variables corresponds to their $\max$-elimination. Single variable marginalization is equivalent to the elimination of all variables but one. For instance, sum-marginalization is important in Bayes nets for achieving the posterior marginal request (see Darwiche (2009)); and in configuration problems, the $\min$-marginalization on a variable in a VDD formula representing some price function amounts to computing the minimal price associated with each possible value of this variable (an option, in the car example; see Astesana, Cosserat, and Fargier (2010)). Another example is value iteration in stochastic planning, which can be performed via a sequence of sum-eliminations and $\times$-combinations on ADD formulæ (Hoey et al. 1999).

Finally, the family of cuts captures the restriction of the function to the optimal assignments (e.g., the cheapest cars, the most probable explanations) or to those that are good enough (cars cheaper than γ euros, diseases with a probability greater than γ). For generality, we defined this request as a transformation within the language, but choosing $a = 1$ and $b = 0$ when $\mathcal{V} = \mathbb{R}^+$, cutting a VDD results in an MDD; the MDD language (Srinivasan et al. 1990) is a direct extension of OBDD to non-Boolean variables, and it satisfies roughly the same queries and transformations as OBDD, e.g., $\mathbf{CO}$, $\mathbf{CD}$, $\mathbf{SE}$, etc. (Amilhastre et al. 2012).

4 A KC Map for Ordered $\mathbb{R}^+$-VDDs

We can now draw the map of existing ordered VDD languages representing functions from $\mathcal{X}$ to $\mathbb{R}^+$, namely ADD, $\mathtt{SLDD}_\times$, $\mathtt{SLDD}_+$, and AADD. That is, we focus in the following on $\mathcal{V} = \mathbb{R}^+$, with $\succeq$ being the usual $\geq$ order on $\mathbb{R}^+$, and $\odot \in \{\max, \min, +, \times\}$. We refrain from explicitly investigating $\mathtt{SLDD}_{\max}$ and $\mathtt{SLDD}_{\min}$, since it has been shown that these languages are equivalent, up to a polynomial transformation, to ADD (Fargier, Marquis, and Schmidt 2013).

Each of the four languages satisfies $\mathbf{CD}$: conditioning a formula α by an assignment $\vec{x}$ can be achieved in linear time using an algorithm similar to the OBDD one. Also, since the representation of any function as a reduced ADD (resp. reduced and normalized AADD, $\mathtt{SLDD}_\times$, $\mathtt{SLDD}_+$) is unique, and the reduction and normalization procedures of each language are polynomial-time, $\mathbf{EQ}$ is satisfied by each of the four languages. Finally, $\mathbf{SE}$ is satisfied by ADD, $\mathtt{SLDD}_+$, and $\mathtt{SLDD}_\times$, using a combination of cuts, inversion of labels, and translation to MDD. Whether AADD satisfies $\mathbf{SE}$ remains open.

Proposition 4.1.

- ADD, $\mathtt{SLDD}_+$, $\mathtt{SLDD}_\times$, *and* AADD *satisfy* $\mathbf{EQ}$ *and* $\mathbf{CD}$.
- ADD, $\mathtt{SLDD}_+$, *and* $\mathtt{SLDD}_\times$ *satisfy* $\mathbf{SE}$.

Let us now draw the KC map for the requests identified in the previous section. We distinguish three classes of requests: those related to optimization, that are tractable on the VDD languages; those related to cutting with respect to some threshold γ, that may become harder; and the combination and projection transformations.

Requests related to optimization tasks. The results we obtained are summarized in Table 1. As already mentioned, $\mathbf{OPT}_{\max}$ and $\mathbf{OPT}_{\min}$ have been shown to be tractable for

Table 1: Results about basic queries, optimization, and γ-cutting; $\surd$ means "satisfies", $\bullet$ means "does not satisfy", and $\circ$ means "does not satisfy unless $\mathsf{P} = \mathsf{NP}$". Results for additive valued constraint satisfaction problems ($\texttt{VCSP}_+$) are given here as a baseline.

Query	ADD	$\texttt{SLDD}_+$	$\texttt{SLDD}_\times$	AADD	$\texttt{VCSP}_+$
$\mathbf{CD}$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
$\mathbf{EQ}$	$\surd$	$\surd$	$\surd$	$\surd$	?
$\mathbf{SE}$	$\surd$	$\surd$	$\surd$	?	$\circ$
$\mathbf{OPT}_{\max}$ / $\mathbf{OPT}_{\min}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{CT}_{\max}$ / $\mathbf{CT}_{\min}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{ME}_{\max}$ / $\mathbf{ME}_{\min}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{MX}_{\max}$ / $\mathbf{MX}_{\min}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{CUT}_{\max}$ / $\mathbf{CUT}_{\min}$	$\surd$	$\surd$	$\surd$	$\surd$	?
$\mathbf{VA}_{\sim\gamma}$	$\surd$	$\surd$	$\surd$	$\surd$	?
$\mathbf{VA}_{\succeq\gamma}$ / $\mathbf{VA}_{\preceq\gamma}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{CO}_{\sim\gamma}$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$
$\mathbf{CO}_{\succeq\gamma}$ / $\mathbf{CO}_{\preceq\gamma}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{ME}_{\sim\gamma}$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$
$\mathbf{ME}_{\succeq\gamma}$ / $\mathbf{ME}_{\preceq\gamma}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{MX}_{\sim\gamma}$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$
$\mathbf{MX}_{\succeq\gamma}$ / $\mathbf{MX}_{\preceq\gamma}$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$
$\mathbf{CUT}_{\sim\gamma}$	$\surd$	$\circ$	$\circ$	$\circ$	?
$\mathbf{CUT}_{\succeq\gamma}$ / $\mathbf{CUT}_{\preceq\gamma}$	$\surd$	$\bullet$	$\bullet$	$\bullet$	?
$\mathbf{CT}_{\sim\gamma}$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$
$\mathbf{CT}_{\succeq\gamma}$ / $\mathbf{CT}_{\preceq\gamma}$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$

normalized AADD formulæ (Sanner and McAllester 2005), for SLDD formulæ (Wilson 2005), and their satisfaction is obvious for ADD. All tractability results of Table 1 can be related to the fact that (i) VDDs are circuit-free graphs, and (ii) the aggregation of the φ values is monotonic in the classes considered. In such diagrams, minimal (resp. maximal) paths can be obtained in polynomial time, thanks to a shortest (resp. longest) path algorithm; this is the basis for a polynomial-time procedure that builds an MDD formula representing the optimal assignments, and hence implies the satisfaction of $\mathbf{CUT}_{\max}$ and $\mathbf{CUT}_{\min}$.

Proposition 4.2. ADD, $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, *and* AADD *satisfy* $\mathbf{OPT}_{\max}$, $\mathbf{OPT}_{\min}$, $\mathbf{CUT}_{\max}$, $\mathbf{CUT}_{\min}$, $\mathbf{ME}_{\max}$, $\mathbf{ME}_{\min}$, $\mathbf{MX}_{\max}$, $\mathbf{MX}_{\min}$, $\mathbf{CT}_{\max}$, *and* $\mathbf{CT}_{\min}$.

Requests related to γ-cuts. Requests related to pure optimization are easy to solve. However, optimization alone is not sufficient for many applications; for instance, a customer looking for a car is not always interested in finding out one of the cheapest cars: a 'cheap' car (i.e., with a cost lower than a given threshold) may be more interesting than the cheapest ones if it fulfills other desiderata of the customer. Hence the importance of requests related to γ-cuts.

Fortunately, comparing the maximal (resp. minimal) value of $f^{\mathtt{L}}_\alpha$ (which can be computed in polynomial time) to some $\gamma \in \mathcal{V}$ is enough to decide whether there exists an $\vec{x}$ such that $f^{\mathtt{L}}_\alpha(\vec{x}) \succeq \gamma$ (resp. $\preceq \gamma$). Similarly, deciding the γ-validity of a formula α only requires the comparison of γ to the maximal (resp. minimal) value of $f^{\mathtt{L}}_\alpha$.

Proposition 4.3. ADD, $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, *and* AADD *satisfy* $\mathbf{CO}_{\succeq\gamma}$, $\mathbf{VA}_{\succeq\gamma}$, $\mathbf{CO}_{\preceq\gamma}$, $\mathbf{VA}_{\preceq\gamma}$, *and* $\mathbf{VA}_{\sim\gamma}$.

Although checking that there is an assignment that leads to a valuation greater than or equal to γ is polynomial, deciding whether there exists an assignment leading *exactly* to γ is not tractable for $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, and AADD. The proof is based on a polynomial reduction from the SUBSET SUM decision problem (Garey and Johnson 1979).

Proposition 4.4. $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, *and* AADD *do not satisfy* $\mathbf{CO}_{\sim\gamma}$ *unless* $\mathsf{P} = \mathsf{NP}$.

It is quite clear that the satisfaction of $\mathbf{MX}_{\sim\gamma}$ or $\mathbf{CT}_{\sim\gamma}$ is a sufficient condition to that of $\mathbf{CO}_{\sim\gamma}$; and this holds in all generality, not only for our four $\mathbb{R}^+$-valued languages. Similarly, it can be shown that $\mathbf{ME}_{\sim\gamma}$ is a sufficient condition for $\mathbf{CO}_{\sim\gamma}$, and that $\mathbf{CUT}_{\sim\gamma}$ implies $\mathbf{MX}_{\sim\gamma}$ as long as one of the $\mathbf{MX}$ queries is satisfied.

Proposition 4.5. *Let* L *be a representation language over* $\mathcal{X}$ *w.r.t.* $\mathcal{V}$*, where* $\mathcal{V}$ *is totally ordered by a relation* $\succeq$*.*

- *If* L *satisfies both* $\mathbf{CUT}_{\sim\gamma}$ *and one of the* $\mathbf{MX}$ *queries, then it satisfies* $\mathbf{MX}_{\sim\gamma}$*.*
- *If* L *satisfies* $\mathbf{MX}_{\sim\gamma}$, $\mathbf{CT}_{\sim\gamma}$, *or* $\mathbf{ME}_{\sim\gamma}$*, then it satisfies* $\mathbf{CO}_{\sim\gamma}$*.*

Corollary 4.6. $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, *and* AADD *do not satisfy* $\mathbf{MX}_{\sim\gamma}$, $\mathbf{CUT}_{\sim\gamma}$, $\mathbf{CT}_{\sim\gamma}$, *or* $\mathbf{ME}_{\sim\gamma}$ *unless* $\mathsf{P} = \mathsf{NP}$.

Thus, except for $\mathbf{VA}_{\sim\gamma}$, queries about a precise cut of the function represented by an arc-labeled (in the wide sense) VDD are intractable. The situation is more nuanced when lower and upper cuts are considered instead.

Proposition 4.7. *Let* $\mathtt{L} \in \{\texttt{SLDD}_+, \texttt{SLDD}_\times, \texttt{AADD}\}$*.*

- L *satisfies* $\mathbf{ME}_{\succeq\gamma}$, $\mathbf{ME}_{\preceq\gamma}$, $\mathbf{MX}_{\succeq\gamma}$, *and* $\mathbf{MX}_{\preceq\gamma}$*.*
- L *does not satisfy* $\mathbf{CT}_{\succeq\gamma}$ *or* $\mathbf{CT}_{\preceq\gamma}$ *unless* $\mathsf{P} = \mathsf{NP}$.
- L *does not satisfy* $\mathbf{CUT}_{\succeq\gamma}$ *or* $\mathbf{CUT}_{\preceq\gamma}$*.*

For proving the first item, the basic idea is that from any node of a normalized AADD formula, there is a path to the leaf that has value 1, and a path to the leaf that has value 0. A path from the root to this node gives it an offset, gathered from its arcs, say $\langle p, q\rangle$. Hence we can enumerate all paths, without exploring nodes for which $p + q < \gamma$ for $\mathbf{ME}_{\succeq\gamma}$ (resp. $p > \gamma$ for $\mathbf{ME}_{\preceq\gamma}$).

The proof of the next item comes from that fact that if $\mathbf{CT}_{\succeq\gamma}$ held, then we could count the assignments $\vec{x}$ such that $f_\alpha(\vec{x}) \geq \gamma$, and also those for which $f_\alpha(\vec{x}) > \gamma$; that is, we could satisfy $\mathbf{CT}_{\sim\gamma}$, which has been shown intractable (and similarly for $\mathbf{CT}_{\preceq\gamma}$).

The intractability of $\mathbf{CUT}_{\succeq\gamma}$ or $\mathbf{CUT}_{\preceq\gamma}$ is unconditional, and the proofs are trickier. For the specific case of $\mathbf{CUT}_{\succeq\gamma}$ on $\texttt{SLDD}_+$, the proof uses the fact that the function $f(\vec{y} \cdot \vec{z}) = \sum_{i=1}^{n} y_i \cdot 2^{n-i} + \sum_{i=1}^{n} z_i \cdot 2^{n-i}$ can be represented as an $\texttt{SLDD}_+$ formula of $2n + 1$ nodes only, using the variable ordering $y_1 \lhd \cdots \lhd y_n \lhd z_1 \lhd \cdots \lhd z_n$, whereas there is no polynomial-size $\texttt{OBDD}_\lhd$ representation of the function returning 1 when $f(\vec{y} \cdot \vec{z}) \geq 2^n$ and 0 otherwise. The other proofs are similar, using well-chosen functions instead of f.

These results about AADD and SLDD contrast with the case of ADD, which satisfies each $\mathbf{CUT}$ transformation (γ-cuts are obtained thanks to a simple leaf-merging procedure), and thus satisfies each of the $\mathbf{MX}$, $\mathbf{ME}$, $\mathbf{CT}$, and $\mathbf{CO}$ queries.

Combination, variable elimination, marginalization. None of the VDD languages considered in this paper satisfies the unbounded combination or unbounded variable elimination transformations; this result is not very surprising, observing that (i) unbounded disjunctive combination ($\vee\mathbf{C}$) and forgetting ($\mathbf{FO}$) are not satisfied by OBDD; (ii) any OBDD formula can be viewed as an ADD formula, and thus be translated in polynomial time into an $\mathtt{SLDD}_+$ (resp. $\mathtt{SLDD}_\times$, AADD) formula; and (iii) the disjunction of several OBDD formulæ amounts to cutting their $+$-combination at the minimal level ($\mathbf{CUT}_{\min}$ is satisfied): by construction, the countermodels of the disjunction are the $\vec{x}$ such that $\varphi(\vec{x}) = 0$, so the resulting formula can be viewed as an OBDD formula the negation of which is equivalent to the disjunction of the original formulæ). The other proofs are based on similar arguments. Note that contrary to the OBDD case, even $\odot$-eliminating a single variable is hard (roughly speaking, the $\odot$-combination of two formulæ can be built by $\odot$-eliminating an additional variable).

Proposition 4.8. *For any* $\mathtt{L} \in \{\mathtt{ADD}, \mathtt{SLDD}_+, \mathtt{SLDD}_\times, \mathtt{AADD}\}$ *and any* $\odot \in \{\max, \min, +, \times\}$*,* L *does not satisfy* $\odot\mathbf{C}$*,* $\odot\mathbf{Elim}$*, or* $\mathbf{S}\odot\mathbf{Elim}$*.*

More difficult is the question of bounded combination. An extension of "apply" algorithm of Bryant (1986), which is polynomial-time on OBDD structures for the AND and OR operators, has been proposed (Sanner and McAllester 2005) to compute the combination (e.g., by $+$, $\times$, max, min) of two AADD formulæ; it can be easily adapted to the other VDD languages considered in this paper. However, the complexity of this "apply" algorithm had not been formally identified. One of the main results of this paper is that bounded combinations are not tractable for AADD, which implies that the extended "apply" is not a polynomial-time algorithm.

Proposition 4.9.

- $\mathtt{SLDD}_+$, $\mathtt{SLDD}_\times$*, and* AADD *do not satisfy* $\max\mathbf{BC}$*,* $\min\mathbf{BC}$*,* $\mathbf{SB}\max\mathbf{Elim}$*, or* $\mathbf{SB}\min\mathbf{Elim}$*.*
- $\mathtt{SLDD}_\times$ *and* AADD *do not satisfy* $+\mathbf{BC}$ *or* $\mathbf{SB}{+}\mathbf{Elim}$*.*
- $\mathtt{SLDD}_+$ *and* AADD *do not satisfy* $\times\mathbf{BC}$ *or* $\mathbf{SB}{\times}\mathbf{Elim}$*.*

The difficulty of $\max\mathbf{BC}$ or $\min\mathbf{BC}$ comes from that of $\mathbf{CUT}_{\succeq\gamma}$and $\mathbf{CUT}_{\preceq\gamma}$. We show the difficulty of $\times\mathbf{BC}$ by considering the two following functions on Boolean variables: $f(\vec{x}) = \sum_{i=0}^{n-1} x_i \cdot 2^i$ (representation of an integer by a bitvector) and $g(\vec{x}) = 2^{n+1} - f(\vec{x})$; each can be represented as an $\mathtt{SLDD}_+$ formula (with ordering $x_0 \lhd x_1 \lhd \cdots \lhd x_{n-1}$) with $n+1$ nodes and $2n$ arcs. Then we can show that the $\mathtt{SLDD}_+$ representation of $f \times g$ (using the same variable ordering) contains an exponential number of terminal arcs, even if all nodes at the last level have been normalized; this proves that each ordered $\mathtt{SLDD}_+$ representation of $f \times g$ is of exponential size. The other proofs are similar, using well-chosen functions f and g.

There are actually only two cases in which bounded combinations are tractable: when the language considered is

Table 2: Results about transformations (legend in Table 1).

Transformation	ADD	$\mathtt{SLDD}_+$	$\mathtt{SLDD}_\times$	AADD
$\max\mathbf{BC}$ / $\min\mathbf{BC}$	√	●	●	●
$+\mathbf{BC}$	√	√	●	●
$\times\mathbf{BC}$	√	●	√	●
$\max\mathbf{C}$ / $\min\mathbf{C}$	●	●	●	●
$+\mathbf{C}$ / $\times\mathbf{C}$	●	●	●	●
$\max\mathbf{Elim}$ / $\min\mathbf{Elim}$	●	●	●	●
$+\mathbf{Elim}$ / $\times\mathbf{Elim}$	●	●	●	●
$\mathbf{S}\max\mathbf{Elim}$ / $\mathbf{S}\min\mathbf{Elim}$	●	●	●	●
$\mathbf{S}{+}\mathbf{Elim}$ / $\mathbf{S}{\times}\mathbf{Elim}$	●	●	●	●
$\mathbf{SB}\max\mathbf{Elim}$ / $\mathbf{SB}\min\mathbf{Elim}$	√	●	●	●
$\mathbf{SB}{+}\mathbf{Elim}$	√	√	●	●
$\mathbf{SB}{\times}\mathbf{Elim}$	√	●	√	●
$\max\mathbf{Marg}$ / $\min\mathbf{Marg}$	√	√	√	√
$+\mathbf{Marg}$	√	√	√	√
$\times\mathbf{Marg}$	√	?	√	?

ADD, and when the combination operator is also the one that aggregates arc values in the language considered (i.e., addition for $\mathtt{SLDD}_+$, and product for $\mathtt{SLDD}_\times$). In these cases, it is also polynomial to eliminate a single variable with a bounded domain, by definition of variable elimination.

Proposition 4.10.

- $\mathtt{SLDD}_+$ *satisfies* $+\mathbf{BC}$ *and* $\mathbf{SB}{+}\mathbf{Elim}$*.*
- $\mathtt{SLDD}_\times$ *satisfies* $\times\mathbf{BC}$ *and* $\mathbf{SB}{\times}\mathbf{Elim}$*.*
- ADD *satisfies* $\odot\mathbf{BC}$ *and* $\mathbf{SB}\odot\mathbf{Elim}$*, for any operation* $\odot \in \{\times, +, \min, \max\}$*.*

Finally, we have proved that $\times$-marginalization is tractable for ADD and $\mathtt{SLDD}_\times$, and that max-marginalization, min-marginalization and $+$-marginalization are tractable for all languages considered. Whether AADD and $\mathtt{SLDD}_+$ satisfy $\times$-marginalization remains an open question.

5 Conclusion

In this paper, we presented a complexity analysis of VDD languages, based on a set of queries and transformations of interest. Requests related to optimization appear as tractable for all VDD languages, as well as additive marginalization and some of the queries related to cutting. Cutting queries prove computationally difficult when a level γ is has to be reached exactly, and this is also the case for the transformations we considered. One of the main results of the paper is that bounded combinations are not tractable on VDD languages, which implies that no "apply" algorithm can run in polynomial time on AADD formulæ in the general case (this is the case even for simple "Boolean-style" operations such as min and max). When bounded additive (resp. multiplicative) combination is required to be achieved efficiently, the time/space tradeoff offered by $\mathtt{SLDD}_+$ (resp. $\mathtt{SLDD}_\times$) is valuable. Finally, it turns out that the complexity of the various queries related to optimization is better for the VDD languages than for other languages dedicated to the representation of non-Boolean functions, such as $\mathtt{VCSP}_+$ (and similar results are expected for GAI nets or Bayes nets, for which optimization is also hard).

Acknowledgements

This work was partially supported by the project BR4CP ANR-11-BS02-008 of the French National Agency for Research.

References

Amilhastre, J.; Fargier, H.; Niveau, A.; and Pralet, C. 2012. Compiling CSPs: A complexity map of (non-deterministic) multivalued decision diagrams. In *Proceedings of ICTAI'12*, 1–8.

Amilhastre, J.; Fargier, H.; and Marquis, P. 2002. Consistency restoration and explanations in dynamic CSPs: Application to configuration. *Artificial Intelligence* 135(1–2):199–234.

Astesana, J.-M.; Cosserat, L.; and Fargier, H. 2010. Constraint-based vehicle configuration: A case study. In *Proceedings of ICTAI'10*, 68–75.

Bacchus, F., and Grove, A. J. 1995. Graphical models for preference and utility. In *Proceedings of UAI'95*, 3–10.

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 *Proceedings of ICCAD'93*, 188–191.

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* 35:677–691.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research (JAIR)* 17:229–264.

Darwiche, A. 2009. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

Fargier, H.; Marquis, P.; and Schmidt, N. 2013. Semiring labelled decision diagrams, revisited: Canonicity and spatial efficiency issues. In *Proceedings of IJCAI'13*, 884–890.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Gogic, G.; Kautz, H.; Papadimitriou, C.; and Selman, B. 1995. The comparative linguistics of knowledge representation. In *Proceedings of IJCAI'95*, 862–869.

Hadzic, T., and Andersen, H. R. 2006. A BDD-based polytime algorithm for cost-bounded interactive configuration. In *Proceedings of AAAI'06*, 62–67.

Hoey, J.; St-Aubin, R.; Hu, A. J.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of UAI'99*, 279–288.

Lai, Y.-T., and Sastry, S. 1992. Edge-valued binary decision diagrams for multi-level hierarchical verification. In *Proceedings of DAC'92*, 608–613.

Lai, Y.-T.; Pedram, M.; and Vrudhula, S. B. K. 1996. Formal verification using edge-valued binary decision diagrams. *IEEE Transactions on Computers* 45(2):247–255.

Pearl, J. 1989. *Probabilistic reasoning in intelligent systems – networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann.

Sanner, S., and McAllester, D. A. 2005. Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In *Proceedings of IJCAI'05*, 1384–1390.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of IJCAI'95 (1)*, 631–639.

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

Srinivasan, A.; Kam, T.; Malik, S.; and Brayton, R. K. 1990. Algorithms for discrete function manipulation. In *Proceedings of ICCAD'90*, 92–95.

Tafertshofer, P., and Pedram, M. 1997. Factored edge-valued binary decision diagrams. *Formal Methods in System Design* 10(2/3).

Wegener, I. 1987. *The complexity of Boolean functions*. Wiley-Teubner.

Wilson, N. 2005. Decision diagrams for the computation of semiring valuations. In *Proceedings of IJCAI'05*, 331–336.

Appendix: Proofs

Preliminaries

We first introduce a couple of technical propositions that are only evoked in the main text, for space reasons, but are useful for proving the following results. For simplicity, we often consider that the offset of SLDD and AADD formulæ is the valuation of a pseudo-arc pointing to the root, called the "*offset-arc*".

Proposition A.1. *Any* VDD *formula can be turned in polynomial time into an equivalent reduced formula in the same language.*

Proof. The proof is straightforwardly adapted from the well-known bottom-up reduction algorithm of OBDD (Bryant 1986). The goal of the reduction procedure is to merge the isomorphic nodes (two nodes N and N' in a VDD formula are isomorphic when they are leaf nodes with the same label, or when they are internal nodes labeled with the same variable x, have the same number k of outgoing arcs $a_1, \dots, a_k$ and $a'_1, \dots, a'_k$ and for each $i \in \{1, \dots, k\}$, $v(a_i) = v(a'_i)$ and $\varphi(a_i) = \varphi(a'_i)$) and to remove the redundant nodes (an internal node N is redundant in a VDD formula when it has a single child M and every arc from N to M has the same φ-label). It simply traverses the diagram while maintaining a cache of already encountered nodes, recording their variable and the label, value, and destination of each of their outgoing arcs. When it finds a node N that is isomorphic to some node N' in the cache, it removes the duplicate N, redirecting all incoming arcs to N'. When it finds a node N with all outoing arcs pointing to the same node M and having the same valuation, it removes N and redirects its incoming arcs to M, updating their valuations accordingly (this update operation depends on the language under consideration). □

Definition A.2. An AADD formula α is normalized if and only if

Algorithm 1: NormalizeAADD(α)

```
input : an AADD formula $\alpha$
output: a normalized AADD formula equivalent to $\alpha$
1  foreach arc $a$ pointing to the leaf do $f_a := 0$
2  foreach non-leaf node $N$ of $\alpha$ in reverse topological ordering do
3      $q_{min} := \min_{a \in \text{Out}(N)} q_a$
4      $range := \max_{a \in \text{Out}(N)}(q_a + f_a) - q_{min}$
5      foreach $a' \in \text{In}(N)$ do
6          $q_{a'} := q_{a'} + q_{\min} \times f_{a'}$
7          $f_{a'} := f_{a'} \times range$
8          if $f_{a'} = 0$ then
9              redirect $a'$ to the leaf
10     if $range = 0$ then
11         remove $N$ and its outgoing arcs (the incoming arcs have been redirected)
12     else
13         foreach $a \in \text{Out}(N)$ do
14             $q_a := (q_a - q_{min})/range$
15             $f_a := f_a/range$
16 return $\alpha$
```

- for each node N in α, $\min_{a \in \text{Out}(N)} q_a = 0$;
- for each node N in α, $\max_{a \in \text{Out}(N)}(q_a + f_a) = 1$;
- for any arc a (including the offset-arc), $f_a = 0$ if and only if a points to the leaf.

Proposition A.3. *There exists a linear algorithm transforming any* AADD *formula into an equivalent normalized* AADD *formula.*

Proof. The backward normalization procedure of AADD is outlined in Algorithm 1 (note that the offset-arc is considered as a normal arc, pointing to the root). It should be clear that (i) the processing of each node leaves its semantics identical (modifications of outgoing arcs are reported on incoming arcs); (ii) each node is modified so as to obey the normalization conditions; and (iii) each arc is modified at most two times, so the procedure is linear in the number of arcs in α. □

Definition A.4. An SLDD_+ (resp. $\text{SLDD}_\times$) formula α is normalized if and only if each node N in α verifies $\min_{a \in \text{Out}(N)} \varphi(a) = 0$ (resp. $\max_{a \in \text{Out}(N)} \varphi(a) = 1$).

Proposition A.5. *There exists a linear algorithm transforming any* SLDD_+ *(resp.* $\text{SLDD}_\times$*) formula into an equivalent normalized* SLDD_+ *(resp.* $\text{SLDD}_\times$*) formula.*

Proof. The backward normalization procedure for SLDD_+ and $\text{SLDD}_\times$ is outlined in Algorithm 2 (note that the offset-arc is considered as a normal arc, pointing to the root). □

There is no normalization condition for ADD; to simplify the statement of the following proposition, we implicitly consider that a "normalized ADD formula" is simply an ADD formula.

Algorithm 2: NormalizeSLDD(α)

```
input : an SLDD formula $\alpha$
output: a normalized SLDD formula equivalent to $\alpha$
// For $\text{SLDD}_+$, let $\otimes$ be $+$, $\otimes^{-1}$ be $-$, and proj be min
// For $\text{SLDD}_\times$, let $\otimes$ be $\times$, $\otimes^{-1}$ be $\div$, and proj be max
1  foreach non-leaf node $N$ of $\alpha$ in reverse topological ordering do
2      $\varphi_m := \text{proj}_{a \in \text{Out}(N)} \varphi(a)$
       // Rem: when proj is max
       // $\varphi_m = 0$ iff $\forall a, \varphi(a) = 0$
3      foreach $a \in \text{Out}(N)$ do
4          if $\varphi(a) = \varphi_m$ then
5              $\varphi(a) := 1_\otimes$
6          else $\varphi(a) := \varphi(a) \otimes^{-1} \varphi_m$
7
8      foreach $a \in In(N)$ do
9          $\varphi(a) := \varphi(a) \otimes \varphi_m$
10     return $\alpha$
```

Proposition A.6. *For any* $\text{L} \in \{\text{ADD}, \text{SLDD}_+, \text{SLDD}_\times, \text{AADD}\}$ *formula* α*,*

- *there is a unique (up to isomorphism) reduced and normalized* L *representation* α' *of* f^{L}_α*;*
- *no* L *representation of* f^{L}_α *can be smaller than* $|\alpha'|$*.*

Proof. This has been proven by Bahar et al. (1993) for ADD, Wilson (2005) for SLDD_+ and $\text{SLDD}_\times$, and Sanner and McAllester (2005) for AADD. □

Proposition A.7. *Denoting* $\text{L} \geq_\ell \text{L}'$ *the fact that there exists a linear-time algorithm that associates with any* L *formula an equivalent* L' *formula, the following results hold:*

- $\text{ADD} \geq_\ell \text{SLDD}_+ \geq_\ell \text{AADD}$*;*
- $\text{ADD} \geq_\ell \text{SLDD}_\times \geq_\ell \text{AADD}$.

Proof.

- $\text{SLDD}_+ \geq_\ell \text{AADD}$, $\text{SLDD}_\times \geq_\ell \text{AADD}$: Because any SLDD_+ (resp. $\text{SLDD}_\times$) representation of a function f can be transformed in linear time into an AADD representation α of the same function: for each arc a (including the offset-arc), its valuation $\varphi(a)$ becomes $\langle \varphi(a), 1 \rangle$ (resp. $\langle 0, \varphi(a) \rangle$); the initial offset of α is $\langle q_0, 1 \rangle$.
- $\text{ADD} \geq_\ell \text{SLDD}_\times$, $\text{ADD} \geq_\ell \text{SLDD}_+$, $\text{ADD} \geq_\ell \text{AADD}$: any ADD representation of a function f can be transformed in linear time into an SLDD_+ (resp. $\text{SLDD}_\times$, resp. AADD) representation of the same function; the transformation process is as follows: each arc a entering a terminal node N receives the valuation $\varphi(a) = \varphi(N)$ (resp. $\varphi(a) = \varphi(N)$, resp. $\langle \varphi(N), 0 \rangle$); the other arcs are labeled with the neutral element of the language considered (0 for SLDD_+, 1 for $\text{SLDD}_\times$, $\langle 0, 1 \rangle$ for AADD); the labels of the terminal nodes are deleted and those nodes are merged so as to become the leaf of the resulting VDD formula. The offset is 0 (resp. 1, resp. $\langle 0, 1 \rangle$). □

Proposition A.8. *Any* ADD*,* SLDD$_+$*,* SLDD$_\times$*, or* AADD *representation of a function taking its values in $\mathcal{V} = \{0, 1\}$ can be translated in polynomial time into an* MDD *representation of the same function.*

Proof. We will show the result for AADD; the others can be inferred from it, since any formula in one of these languages can be transformed in linear time into an equivalent AADD formula (Proposition A.7). Note however that the result is actually obvious for the ADD language: when $\mathcal{V} = \mathcal{E} = \{0, 1\}$, ADD = MDD.

Any AADD formula can be normalized in linear time (Proposition A.3). Let α be a normalized AADD representation of a function taking its values in $\mathcal{V} = \{0, 1\}$. When α has no non-leaf node, it represents a constant function, so the translation to MDD is trivial. Suppose α contains at least one non-leaf node; we show by induction that all arcs that do not point to the leaf (including the offset-arc) have valuation $\langle 0, 1\rangle$.

Suppose that this is true for all arcs on some path from the offset-arc to a node N. Let $a \in \mathrm{Out}(N)$ be an arc that does not point to the leaf. Denoting M the destination node of a, the normalization condition also ensures that there exists one path from M to the leaf that has value 0, and another that has value 1. All in all, there exists a complete path of value q_a and another complete path of value $q_a + f_a$; these two values must be equal to 0 or 1. Since by the normalization condition, $f_a \neq 0$, the only possibility is that $q_a = 0$ and $f_a = 1$. This reasoning clearly applies to the offset arc, so by induction, all arcs not pointing to the leaf have valuation $\langle 0, 1\rangle$.

Now, let a be an arc pointing to the leaf. By the normalization condition, $f_a = 0$. Any complete path traversing a has value q_a (recall that along any path, all arcs except the last one have valuation $\langle 0, 1\rangle$), so q_a is either 0 or 1.

The conversion to MDD is thus direct: create a new leaf labeled with 1, redirect the $\langle 1, 0\rangle$ arcs to this new leaf, and remove all the arc valuations. □

Queries

Proposition A.9. ADD*,* SLDD$_+$*,* SLDD$_\times$*, and* AADD *satisfy* **EQ**.

Proof. This query is satisfied since each language L in $\{\mathtt{ADD}, \mathtt{SLDD}_+, \mathtt{SLDD}_\times, \mathtt{AADD}\}$ offers the canonicity property, i.e., any function has a unique reduced, normalized L representation given a fixed variable ordering (Proposition A.6), and since normalizing and reducing an L formula can be done in linear time (Proposition A.1 for the reduction, Proposition A.3 for the normalization of AADD, and Proposition A.5 for the normalization of SLDD). □

Proposition A.10. ADD*,* SLDD$_+$*,* SLDD$_\times$*, and* AADD *satisfy* **CD**.

Proof. We first show that single variable conditioning is in linear time on VDD formulæ. Assume that the conditioning of α by $x = v$ is asked for. For any node N labeled with x, the arcs in $\mathrm{Out}(N)$ corresponding to a value different from v must be discarded to get a VDD representation of the restriction $f_{\alpha,\langle x,v\rangle}$ of f_α. Because VDD formulæ are deterministic, there is only one arc a in $\mathrm{Out}(N)$ corresponding to value v.

In order to derive a VDD representation of $f_{\alpha,\langle x,v\rangle}$, each arc b in $\mathrm{In}(N)$ has to be redirected to the destination node of a and its label must be updated to take account of the removal of a. This update depends on the language considered: for ADD, there is nothing to do; for AADD, the label $\langle q_b, f_b\rangle$ of b must be replaced by $\langle q_b + f_b \times q_a, f_b \times f_a\rangle$; for SLDD$_+$ (resp. SLDD$_\times$), the label φ_b of b must be replaced by $\varphi_b + \varphi_a$ (resp. $\varphi_b \times \varphi_a$).

Accordingly, single variable conditioning can be achieved in time linear in the size of α, and the resulting formula has strictly fewer nodes and arcs that α (each node processing removes $|D_x| - 1$ arcs). Full conditioning simply amounts to iterating single variable conditioning for each conditioned variable, thus it runs in time polynomial in the input size. □

Proposition A.11. ADD *satisfies* $\mathbf{CUT}_{\sim\gamma}$*,* $\mathbf{CUT}_{\preceq\gamma}$*, and* $\mathbf{CUT}_{\succeq\gamma}$.

Proof. This is trivial, because the values taken by a function are listed as the leaves of its ADD representation. Hence, $CUT^{\sim\gamma}$ (resp. $CUT^{\preceq\gamma}$, $CUT^{\succeq\gamma}$) is the set of assignments the paths of which lead to a leaf with value $\varphi = \gamma$ (resp. $\varphi \leq \gamma$, $\varphi \geq \gamma$). Replacing the label of every such leaf by a and that of all other leaves by b, which can be done in time linear in the size of the formula, we obtain an ADD representation of the function g defined by $g(\vec{x}) = a$ if $\vec{x}$ is in the cut and $g(\vec{x}) = b$ otherwise. □

Corollary A.12. ADD *satisfies*

- $\mathbf{CO}_{\sim\gamma}$*,* $\mathbf{CO}_{\preceq\gamma}$*, and* $\mathbf{CO}_{\succeq\gamma}$*;*
- $\mathbf{VA}_{\sim\gamma}$*,* $\mathbf{VA}_{\preceq\gamma}$*, and* $\mathbf{VA}_{\succeq\gamma}$*;*
- $\mathbf{ME}_{\sim\gamma}$*,* $\mathbf{ME}_{\preceq\gamma}$*, and* $\mathbf{ME}_{\succeq\gamma}$*;*
- $\mathbf{MX}_{\sim\gamma}$*,* $\mathbf{MX}_{\preceq\gamma}$*, and* $\mathbf{MX}_{\succeq\gamma}$*;*
- $\mathbf{CT}_{\sim\gamma}$*,* $\mathbf{CT}_{\preceq\gamma}$*, and* $\mathbf{CT}_{\succeq\gamma}$.

Proof. This comes directly from the previous result: ADD satisfies $\mathbf{CUT}_{\sim\gamma}$ (resp. $\mathbf{CUT}_{\preceq\gamma}$; resp. $\mathbf{CUT}_{\succeq\gamma}$), so an ADD formula α can be turned in polynomial time into an ADD formula β such that $f_\beta(\vec{x}) = 1$ if $\vec{x}$ is in $CUT^{\sim\gamma}(f_\alpha)$ (resp. $CUT^{\preceq\gamma}(f_\alpha)$; resp. $CUT^{\succeq\gamma}(f_\alpha)$) and 0 otherwise (we simply take $a = 1$ and $b = 0$). Thanks to Proposition A.8, β can be turned in polynomial time into an equivalent MDD formula. Since MDD satisfies the consistency, validity, model enumeration, model extraction, and model counting queries (Amilhastre et al. 2012), this proves that ADD satisfies $\mathbf{CO}_{\sim\gamma}$, $\mathbf{VA}_{\sim\gamma}$, $\mathbf{ME}_{\sim\gamma}$, $\mathbf{MX}_{\sim\gamma}$, and $\mathbf{CT}_{\sim\gamma}$ (resp. $\mathbf{CO}_{\preceq\gamma}$, $\mathbf{VA}_{\preceq\gamma}$, $\mathbf{ME}_{\preceq\gamma}$, $\mathbf{MX}_{\preceq\gamma}$, and $\mathbf{CT}_{\preceq\gamma}$; resp. $\mathbf{CO}_{\succeq\gamma}$, $\mathbf{VA}_{\succeq\gamma}$, $\mathbf{ME}_{\succeq\gamma}$, $\mathbf{MX}_{\succeq\gamma}$, and $\mathbf{CT}_{\succeq\gamma}$). □

Proposition A.13. ADD*,* SLDD$_+$*,* SLDD$_\times$*, and* AADD *satisfy* $\mathbf{OPT}_{\max}$ *and* $\mathbf{OPT}_{\min}$.

Proof. The satisfaction of $\mathbf{OPT}_{\max}$ and $\mathbf{OPT}_{\min}$ by AADD is due to Sanner and McAllester (2005); they showed that the maximal (resp. minimal) value reached by $f_\alpha^{\mathtt{AADD}}$ is $q_0 + f_0$

(resp. q_0), where $\langle q_0, f_0\rangle$ is the offset of α (assumed to be normalized). Then the satisfaction of $\mathbf{OPT}_{\max}$ and $\mathbf{OPT}_{\min}$ by ADD,[4] SLDD$_+$, and SLDD$_\times$[5] is a consequence of Proposition A.7. $\square$

Proposition A.14. ADD, SLDD$_+$, SLDD$_\times$, *and* AADD *satisfy* $\mathbf{CUT}_{\max}$ *and* $\mathbf{CUT}_{\min}$.

Proof. Let us recall that, for any normalized AADD representation α, the assignments $\vec{x}$ such that $f_\alpha^{\mathtt{AADD}}(\vec{x})$ is maximal correspond to the paths following only the arcs e such that $q_e + f_e = 1$ (they always exist when α is normalized). It is then possible to get an ADD representation of $CUT^{\max}(f_\alpha^{\mathtt{AADD}})$ by replacing the leaf with a new one labeled with a (the a-node) and creating a second leaf labeled with b (the b-node): the arcs such that $q_e + f_e \neq 1$ are then redirected to the b-node (this is done in time linear in the number of arcs). The structure is cleaned by recursively deleting nodes without incoming arcs; this cleaning process is also in linear time. The procedure is thus a polynomial algorithm transforming an AADD formula α into an ADD representation of $CUT^{\max}(f_\alpha^{\mathtt{AADD}})$. This algorithm allow us to conclude about $\mathbf{CUT}_{\max}$ for all four languages, thanks to Proposition A.7: since ADD $\geq_\ell$ AADD, AADD satisfies $\mathbf{CUT}_{\max}$ (we can transform the resulting ADD formula into AADD) and ADD too (we can transform the initial formula into AADD, and keep the resulting ADD formula); SLDD$_+$ (resp. SLDD$_\times$) satisfies $\mathbf{CUT}_{\max}$ because SLDD$_+$ $\geq_\ell$ AADD (resp. SLDD$_\times$ $\geq_\ell$ AADD), which allows us to "prepare" the formula to use the algorithm, and ADD $\geq_\ell$ SLDD$_+$ (resp. ADD $\geq_\ell$ SLDD$_\times$), which allows us to translate the result into the targeted language.

The proof is similar for $\mathbf{CUT}_{\min}$, since for any normalized AADD representation α, the assignments $\vec{x}$ such that $f_\alpha^{\mathtt{AADD}}(\vec{x})$ is minimal correspond to the paths following only the arcs e such that $q_e = 0$ (they always exist when α is normalized). The leaf becomes the a-node and the arcs e such that $q_e \neq 0$ are redirected to the b-node. $\square$

Corollary A.15. ADD, SLDD$_+$, SLDD$_\times$, *and* AADD *satisfy* $\mathbf{ME}_{\max}$, $\mathbf{ME}_{\min}$, $\mathbf{MX}_{\max}$, $\mathbf{MX}_{\min}$, $\mathbf{CT}_{\max}$, $\mathbf{CT}_{\min}$.

Proof. This comes directly from the previous result: for any L $\in$ {ADD, SLDD$_\times$, SLDD$_+$, AADD}, L satisfies $\mathbf{CUT}_{\max}$ (resp. $\mathbf{CUT}_{\min}$), so an L formula can be turned in polynomial time into an L formula α such that $f_\alpha^{\mathtt{L}}(\vec{x}) = 1$ if $\vec{x}$ is a maximal (resp. minimal) value of $f_\alpha^{\mathtt{L}}$, and 0 otherwise (we simply take $a = 1$ and $b = 0$). Thanks to Proposition A.8, α can be turned in polynomial time into an equivalent MDD formula. Since MDD satisfies the model enumeration, model counting, and model extraction queries, this proves that L satisfies $\mathbf{ME}_{\max}$, $\mathbf{CT}_{\max}$, and $\mathbf{MX}_{\max}$ (resp. $\mathbf{ME}_{\min}$, $\mathbf{CT}_{\min}$, and $\mathbf{MX}_{\min}$). $\square$

Proposition A.16. SLDD$_+$, SLDD$_\times$, *and* AADD *satisfy* $\mathbf{CO}_{\succeq\gamma}$, $\mathbf{VA}_{\succeq\gamma}$, $\mathbf{CO}_{\preceq\gamma}$, $\mathbf{VA}_{\preceq\gamma}$.

Proof.
- To determine whether there exists an $\vec{x}$ such that $f_\alpha^{\mathtt{L}}(\vec{x}) \geq \gamma$ (resp. $f_\alpha^{\mathtt{L}}(\vec{x}) \leq \gamma$), it is sufficient to compute the maximal (resp. minimal) value v^* (resp. v_*) reached by $f_\alpha^{\mathtt{L}}$; this can be done in polynomial time since $\mathbf{OPT}_{\max}$ (resp. $\mathbf{OPT}_{\min}$) is satisfied by all three languages (Proposition A.13). Then it is enough to compare this value to γ, since $\exists\vec{x}, f_\alpha^{\mathtt{L}}(\vec{x}) \geq \gamma \iff v^* \geq \gamma$ (resp. $\exists\vec{x}, f_\alpha^{\mathtt{L}}(\vec{x}) \leq \gamma \iff v_* \leq \gamma$).
- To determine whether all the $\vec{x}$ are such that $f_\alpha^{\mathtt{L}}(\vec{x}) \geq \gamma$ (resp. $f_\alpha^{\mathtt{L}}(\vec{x}) \leq \gamma$), it is sufficient to compute the minimal (resp. maximal) value v_* (resp. v^*) reached by $f_\alpha^{\mathtt{L}}$; this can be done in polynomial time since $\mathbf{OPT}_{\min}$ (resp. $\mathbf{OPT}_{\max}$) is satisfied by all three languages. Then it is enough to compare this value to γ, since it holds that $\forall\vec{x}, f_\alpha^{\mathtt{L}}(\vec{x}) \geq \gamma \iff v_* \geq \gamma$ (resp. $\forall\vec{x} f_\alpha^{\mathtt{L}}(\vec{x}) \leq \gamma \iff v^* \leq \gamma$). $\square$

Corollary A.17. SLDD$_+$, SLDD$_\times$, *and* AADD *satisfy* $\mathbf{VA}_{\sim\gamma}$.

Proof. Checking whether all the $\vec{x}$ are such that $f_\alpha^{\mathtt{L}}(\vec{x}) = \gamma$ is equivalent to checking whether for each $\vec{x}$, both $f_\alpha^{\mathtt{L}}(\vec{x}) \geq \gamma$ and $f_\alpha^{\mathtt{L}}(\vec{x}) \leq \gamma$ hold. Since all three languages satisfy both $\mathbf{VA}_{\succeq\gamma}$ and $\mathbf{VA}_{\preceq\gamma}$ (Proposition A.16), they also satisfy $\mathbf{VA}_{\sim\gamma}$. Note that this proof does not work for $\mathbf{CO}_{\sim\gamma}$. $\square$

Proposition A.18. SLDD$_+$, SLDD$_\times$, *and* AADD *do not satisfy* $\mathbf{CO}_{\sim\gamma}$, *unless* P = NP.

Proof. We first show that SLDD$_+$ does not satisfy $\mathbf{CO}_{\sim\gamma}$unless P = NP. This follows from Hadzic and Andersen (2006, Theorem 3), by reduction of SUBSET SUM (given a set of integers $E = \{i_1, \ldots, i_n\}$ of cardinality n and an integer k, is there a subset of E summing to k?) which is NP-complete (Garey and Johnson 1979) even if all the integers in E are positive.

Let us associate with E in polynomial time an SLDD$_+$ formula α_E over a set $\mathcal{X} = \{x_1, \ldots, x_n\}$ of n Boolean variables as follows (see Figure 1): the root of α_E is labeled with x_1, α_E contains n internal nodes respectively labeled with $x_1, \ldots, x_n$ plus one leaf; for each i_j of E, we build two arcs e_j and e'_j in α_E from the x_j node to the x_{j+1} node (or to the leaf when $j = n$): e_j corresponds to $x_j = 0$ and $\varphi(e_j) = 0$; e'_j corresponds to $x_j = 1$ and $\varphi(e'_j) = i_j$. This construction is done in time linear in $|E|$. Each $\vec{x}$ (and thus each path in α) is in bijection with a subset of E and $f_\alpha^{\mathtt{SLDD}_+}(\vec{x})$ is equal to the sum of the elements in this subset.

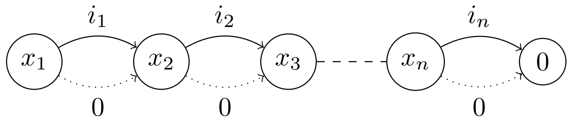

Figure 1: The SLDD$_+$ formula α_E

[4]The satisfaction of $\mathbf{OPT}_{\max}$ and $\mathbf{OPT}_{\min}$ by ADD is obvious (just explore the terminal nodes to get the maximal or minimal value labeling one of them).

[5]For SLDD$_+$ and SLDD$_\times$, our result also coheres with that of Wilson (2005), despite the fact that $(\mathbb{R}^+, \min, +)$ and $(\mathbb{R}^+, \max, +)$ are not semirings. Indeed, this does not matter, the important point being the addition-is-max (or addition-is-min) assumption. See the original paper from Wilson (2005) for details.

If $\texttt{SLDD}_+$ satisfied $\mathbf{CO}_{\sim\gamma}$, then it would be possible to check in polynomial time whether there exists a subset of E summing to k – i.e., to solve the SUBSET SUM problem. Hence $\texttt{SLDD}_+$ does not satisfy $\mathbf{CO}_{\sim\gamma}$, unless $\mathsf{P} = \mathsf{NP}$.

Now, a similar construction holds for the $\texttt{SLDD}_\times$ language, by considering the SUBSET PRODUCT problem (given a set of integers $E = \{i_1, \dots, i_n\}$ of cardinality n and an integer k, is there a subset whose product is k?) which is NP-complete as well (Garey and Johnson 1979).

Finally, if $\texttt{AADD}$ satisfied $\mathbf{CO}_{\sim\gamma}$, then $\texttt{SLDD}_+$ also would, since any $\texttt{SLDD}_+$ formula can be turned into an equivalent $\texttt{AADD}$ formula in linear time (Proposition A.7). However, this is not the case unless $\mathsf{P} = \mathsf{NP}$, hence $\texttt{AADD}$ does not satisfy $\mathbf{CO}_{\sim\gamma}$ unless $\mathsf{P} = \mathsf{NP}$. $\square$

Proposition A.19. *Let* $\texttt{L}$ *be a representation language over* $\mathcal{X}$ *w.r.t.* $\mathcal{V}$*, where* $\mathcal{V}$ *is totally ordered by a relation* $\succeq$*. If* $\texttt{L}$ *satisfies* $\mathbf{MX}_{\sim\gamma}$ *or* $\mathbf{CT}_{\sim\gamma}$*, then* $\texttt{L}$ *satisfies* $\mathbf{CO}_{\sim\gamma}$*.*

Proof. If $\texttt{L}$ satisfies $\mathbf{MX}_{\sim\gamma}$, then we can find in polynomial time an $\vec{x}$ such that $f^{\texttt{L}}_\alpha(\vec{x}) \sim \gamma$ or the information that there is no such assignment, so we can check in polynomial time whether there exists an $\vec{x}$ such that $f^{\texttt{L}}_\alpha(\vec{x}) \sim \gamma$. This shows that $\mathbf{MX}_{\sim\gamma}$ implies $\mathbf{CO}_{\sim\gamma}$. Now, it is obvious that counting the number of $\vec{x}$ such that $f^{\texttt{L}}_\alpha(\vec{x}) \sim \gamma$ indicates whether there exists an $\vec{x}$ such that $f^{\texttt{L}}_\alpha(\vec{x}) \sim \gamma$ (the answer is yes if and only if this number is positive); hence if $\texttt{L}$ satisfies $\mathbf{CT}_{\sim\gamma}$, it also satisfies $\mathbf{CO}_{\sim\gamma}$. $\square$

Proposition A.20. *Let* $\texttt{L}$ *be a representation language over* $\mathcal{X}$ *w.r.t.* $\mathcal{V}$*, where* $\mathcal{V}$ *is totally ordered by a relation* $\succeq$*. If* $\texttt{L}$ *satisfies* $\mathbf{ME}_{\sim\gamma}$*, then* $\texttt{L}$ *satisfies* $\mathbf{CO}_{\sim\gamma}$*.*

Proof. If $\texttt{L}$ satisfies $\mathbf{ME}_{\sim\gamma}$, then there exists a polynomial $p(\cdot,\cdot)$ and an algorithm A taking as input an $\texttt{L}$ formula α and listing every element of $CUT^{\sim\gamma}(f^{\texttt{L}}_\alpha)$ in time bounded by $p(|\alpha|, n)$, where n is the number of elements in $CUT^{\sim\gamma}(f^{\texttt{L}}_\alpha)$.

Now, let A' be the algorithm that, taking as input an $\texttt{L}$ formula α, simulates the execution of A on input α, and stops after at most $p(|\alpha|, 0)$ steps. The result it returns depends on what happened during the simulation. There are three possible cases:

(i) if A stopped without returning anything, then A' returns 0;

(ii) if A stopped after having returned at least one assignment, then A' returns 1;

(iii) if A did not stop by itself, then A' returns 1.

Obviously A' runs in time polynomial in the size of α. Now, it is not hard to see that A' returns 1 if there exists an assignment $\vec{x}$ such that $f^{\texttt{L}}_\alpha(\vec{x}) \sim \gamma$, and 0 otherwise.

Indeed, suppose that no such assignment exists. Then by definition, $CUT^{\sim\gamma}(f^{\texttt{L}}_\alpha)$ is empty, so A stops without having returned anything after at most $p(|\alpha|, 0)$ steps. This is case i: A' returns 0.

Now, suppose that there exists an assignment $\vec{x}$ such that $f^{\texttt{L}}_\alpha(\vec{x}) \sim \gamma$. This means that $CUT^{\sim\gamma}(f^{\texttt{L}}_\alpha)$ is not empty; let us denote n its cardinal. The algorithm A returns n assignments and stops after at most $p(|\alpha|, n)$ steps. Nothing prevents $p(|\alpha|, n)$ to be lower than $p(|\alpha|, 0)$: this corresponds to case ii, in which A' returns 1. Finally, in the (probably less unusual) case when $p(|\alpha|, n) > p(|\alpha|, 0)$, A' has to interrupt the simulation of A; this is case iii, in which A' returns 1.

In conclusion, if $\texttt{L}$ satisfies $\mathbf{ME}_{\sim\gamma}$, then there exists a polynomial-time algorithm deciding whether there exists an assignment $\vec{x}$ such that $f^{\texttt{L}}_\alpha(\vec{x}) \sim \gamma$, that is, $\texttt{L}$ satisfies $\mathbf{CO}_{\sim\gamma}$. $\square$

Proposition A.21. $\texttt{SLDD}_+$*,* $\texttt{SLDD}_\times$*, and* $\texttt{AADD}$ *do not satisfy* $\mathbf{MX}_{\sim\gamma}$*,* $\mathbf{CT}_{\sim\gamma}$*, or* $\mathbf{ME}_{\sim\gamma}$*, unless* $\mathsf{P} = \mathsf{NP}$*.*

Proof. From Proposition A.18, none of these languages satisfies $\mathbf{CO}_{\sim\gamma}$ unless $\mathsf{P} = \mathsf{NP}$. This implies, from Propositions A.19 and A.20, that they cannot satisfy $\mathbf{MX}_{\sim\gamma}$, $\mathbf{CT}_{\sim\gamma}$, or $\mathbf{ME}_{\sim\gamma}$, unless $\mathsf{P} = \mathsf{NP}$. $\square$

Proposition A.22. $\texttt{SLDD}_+$*,* $\texttt{SLDD}_\times$*, and* $\texttt{AADD}$ *satisfy* $\mathbf{MX}_{\succeq\gamma}$ *and* $\mathbf{MX}_{\preceq\gamma}$*.*

Proof. To get an assignment $\vec{x}$ such that $f^{\texttt{L}}_\alpha(\vec{x}) \geq \gamma$, simply check whether γ is strictly greater than the maximal value v^*, which can be obtained in polynomial time since all the languages satisfy $\mathbf{OPT}_{\max}$ (Proposition A.13). If so, we know that there is no assignment $\vec{x}$ such that $f^{\texttt{L}}_\alpha(\vec{x}) \geq \gamma$. In the remaining case (i.e., when $v^* \geq \gamma$), the optimal assignments $\vec{x}^*$ are such that $f^{\texttt{L}}_\alpha(\vec{x}^*) = v^* \geq \gamma$; since all three languages satisfy $\mathbf{MX}_{\max}$ (Proposition A.15), one can get such an $\vec{x}^*$ in polynomial time. Since by construction $\vec{x}^* \in CUT^{\succeq\gamma}$, this proves that all three languages satisfy $\mathbf{MX}_{\succeq\gamma}$. We prove in the same way that they satisfy $\mathbf{MX}_{\preceq\gamma}$, as a consequence of the satisfaction of $\mathbf{OPT}_{\min}$ (Proposition A.13) and $\mathbf{MX}_{\min}$ (Proposition A.15). $\square$

Proposition A.23. $\texttt{SLDD}_+$*,* $\texttt{SLDD}_\times$*, and* $\texttt{AADD}$ *satisfy* $\mathbf{ME}_{\succeq\gamma}$ *and* $\mathbf{ME}_{\preceq\gamma}$*.*

Proof. We prove the result for $\texttt{AADD}$ first, using the basic idea that at any node N of a normalized $\texttt{AADD}$ formula, there is at least one path to the leaf of valuation 1 and one path to the leaf of valuation 0. A path reaching N provides an offset, gathered from its arcs, say $\langle p, q\rangle$, so the cheapest path traversing N has valuation p and the most expensive one has valuation $p + q$.

This allows us to traverse the graph in reverse topological order, only developing a node if it yields at least one element of the cut (e.g., if we want the elements of $CUT^{\succeq\gamma}$, node N is only developed if $p + q \geq \gamma$). We thus get the entire list of elements in the cut by a tree search algorithm that is backtrack-free, and thus polynomial in the number of leaves reached, which is exactly the number of assignments $\vec{x}$ such that $f^{\texttt{L}}_\alpha(\vec{x}) \geq \gamma$ (resp. $f^{\texttt{L}}_\alpha(\vec{x}) \leq \gamma$). A recursive implementation of this procedure is detailed in Algorithm 3 (the top call is supposed to pass $\vec{x} = \varnothing$). Note that the assignments listed are not always complete, in the sense that they only feature variables encountered on the corresponding path. It is straightforward to extend each resulting partial assignment $\vec{z}$ into the full list of complete assignments of which $\vec{z}$ is a restriction.

Algorithm 3: $\mathsf{EnumModelsAADD}(\alpha, \vec{x}, \mathcal{R}, \gamma)$

```
input : an AADD formula $\alpha$, of root $N$, with offset
        $\langle q_0, f_0 \rangle$; a current assignment $\vec{x}$; a relation
        $\mathcal{R} \in \{\leq, \geq\}$, and a threshold $\gamma \in \mathbb{R}^+$
output: the list of assignments $\vec{x} \cdot \vec{y}$ such that
        $f_\alpha^{\texttt{AADD}}(\vec{x} \cdot \vec{y}) \mathcal{R} \gamma$
1  if $\mathcal{R} = \leq$ then
2      if $q_0 > \gamma$ then
3          return
4  else
5      if $q_0 + f_0 < \gamma$ then
6          return
7  if $N$ is the leaf then
8      print $\vec{x}$
9      return
10 let $y \coloneqq \mathrm{Var}(N)$
11 foreach arc $a$ going out of $N$ do
12     let $\vec{y} = \langle y, v(a) \rangle$
13     let $M$ be the destination node of $a$
14     let $\alpha'$ be the AADD formula rooted at the destination
       node of $a$, with offset $\langle q_0 + f_0 \times q_a, f_0 \times f_a \rangle$
15     EnumModelsAADD($\alpha, \vec{x} \cdot \vec{y} \rangle, \mathcal{R}, \gamma$)
```

This proves that `AADD` satisfies $\mathbf{ME}_{\succeq\gamma}$ and $\mathbf{ME}_{\preceq\gamma}$; we can deduce from Proposition A.7 that $\texttt{SLDD}_+$ and $\texttt{SLDD}_\times$ also satisfy these two queries. □

Lemma A.24. *Let $\otimes \in \{+, \times\}$. There exists a polynomial-time algorithm mapping any $\texttt{SLDD}_\otimes$ formula α and every set of variables $X \supseteq \mathrm{Var}(\alpha)$ to an equivalent $\texttt{SLDD}_\otimes$ formula in which each path mentions all the variables in X.*

Proof. Let α be an $\texttt{SLDD}_\otimes$ formula, in which variables are ordered with respect to $\lhd$. Let $X = \{x_1, \dots, x_n\}$ be a set of variables containing all variables in $\mathrm{Var}(\alpha)$; without loss of generality, let us assume that $x_1 \lhd \cdots \lhd x_n$.

What we want is to build an `L` formula α' in which every path from the root to the leaf is of the form $\langle a_1, \dots, a_n \rangle$ where each a_i (with $i \in \{1, \dots, n-1\}$) is an arc from a node labeled with x_i to a node labeled with x_{i+1}, and a_n is an arc from a node labeled with x_n to the leaf node.

This property can be easily guaranteed in polynomial time: for each arc a of α from a node N_i labeled with x_i ($i \in \{1, \dots, n-1\}$) to a node M such that either M is the leaf or $\mathrm{Var}(M) = x_{i+j}$ with $j > 1$,

1. add $j-1$ new nodes $N_{i+1}, \dots, N_{i+j-1}$ respectively labeled $x_{i+1}, \dots, x_{i+j-1}$;
2. redirect a to N_{i+1}; and
3. for each of the new nodes N_k ($k \in \{i, \dots, i+j-1\}$), for each value $d \in D_{x_k}$, add an arc $a_{k,d}$ from N_k to N_{k+1} (conveniently considering N_{i+j} to be M) with $v(a_{k,d}) = d$ and $\varphi(a_{k,d})$ is the neutral element of $\otimes$ (0 for $+$ and 1 for $\times$). □

Lemma A.25. *There exists a polynomial algorithm mapping any $\texttt{SLDD}_+$ formula α of maximal value v^* and any constant $K \in \mathbb{R}^+$ such that $K \geq v^*$, to an $\texttt{SLDD}_+$ representation of the function $g = K - f_\alpha^{\texttt{SLDD}_+}$.*

Proof. This is not hard to prove, but care must be taken to always remain in the fragment (negative arc valuations are forbidden by the definitions).

We first compute v^*, the maximal value taken by $f_\alpha^{\texttt{SLDD}_+}$; this can be done in polynomial time, since $\texttt{SLDD}_+$ satisfies $\mathbf{OPT}_{\max}$ (Proposition A.13). Then we modify α so that all of its paths mentions all the variables in $\mathrm{Var}(\alpha)$ (Lemma A.24 states that it can be done in polynomial time), and every arc label φ (including the offset) is replaced by $K - \varphi$, which is guaranteed to be in $\mathbb{R}^+$ since $K \geq v^*$. We call α' the modified formula.

It should be clear that for any $\vec{x}$, $f_{\alpha'}^{\texttt{SLDD}_+}(\vec{x}) = K \times (1 + |\mathrm{Var}(\alpha)|) - f_\alpha(\vec{x})$, since along each path, we added K as many times as there are arcs, plus one time for the offset. Applying the normalization procedure on α' (in linear time, thanks to Proposition A.5), we get an $\texttt{SLDD}_+$ formula α'' with offset $K \times (1 + |\mathrm{Var}(\alpha)|) - v^*$: indeed, the offset of a normalized $\texttt{SLDD}_+$ formula is the minimal value taken by the function it represents (each node has at least one 0-valued outgoing arc).

We now simply have to substract $K \times |\mathrm{Var}(\alpha)|$ to the offset of α'' to obtain an $\texttt{SLDD}_+$ representation of the function $g = K - f_\alpha^{\texttt{SLDD}_+}$. □

Proposition A.26. *$\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, and `AADD` do not satisfy $\mathbf{CT}_{\succeq\gamma}$ or $\mathbf{CT}_{\preceq\gamma}$ unless $\mathsf{P} = \mathsf{NP}$.*

Proof. We first show that $\texttt{SLDD}_+$ cannot satisfy $\mathbf{CT}_{\succeq\gamma}$ unless $\mathsf{P} = \mathsf{NP}$. Let us suppose it is the case; then we can compute in polynomial time the number $M_{\geq\gamma}$ of assignments $\vec{x}$ such that $f_\alpha^{\texttt{SLDD}_+}(\vec{x}) \geq \gamma$ (i.e., compute $|CUT^{\succeq\gamma}(f_\alpha^{\texttt{SLDD}_+})|$). It is moreover always possible to compute the maximal value v^* reached by $f_\alpha^{\texttt{SLDD}_+}$ ($\texttt{SLDD}_+$ satisfies $\mathbf{OPT}_{\max}$, Proposition A.13) and to get a representation in the same language of the function $g_\alpha = v^* - f_\alpha^{\texttt{SLDD}_+}$ (thanks to Lemma A.25). Since we suppose that $\mathbf{CT}_{\succeq\gamma}$ is satisfied, then it is possible to compute in polynomial time the number M of assignments $\vec{x}$ such that $v^* - f_\alpha^{\texttt{SLDD}_+}(\vec{x}) \geq v^* - \gamma$, taking $v^* - \gamma$ as the threshold. We can then compute the number $M_{>\gamma} = |D_1 \times \cdots \times D_n| - M$ of assignments $\vec{x}$ such that $v^* - f_\alpha^{\texttt{SLDD}_+}(\vec{x}) < v^* - \gamma$: by construction, $M_{>\gamma}$ is equal to the number of assignments $\vec{x}$ such that $f_\alpha^{\texttt{SLDD}_+}(\vec{x}) > \gamma$.

Since we know that there are $M_{\geq\gamma}$ assignments $\vec{x}$ such that $f_\alpha^{\texttt{SLDD}_+}(\vec{x}) \geq \gamma$, we can deduce that there are precisely $M_{\geq\gamma} - M_{>\gamma}$ assignments $\vec{x}$ such that $f_\alpha^{\texttt{SLDD}_+}(\vec{x}) = \gamma$.

We have proved that the satisfaction of $\mathbf{CT}_{\succeq\gamma}$ implies that of $\mathbf{CT}_{\sim\gamma}$; since $\texttt{SLDD}_+$ does not satisfy $\mathbf{CT}_{\sim\gamma}$ unless $\mathsf{P} = \mathsf{NP}$ (Proposition A.21), we conclude that $\texttt{SLDD}_+$ does not satisfy $\mathbf{CT}_{\succeq\gamma}$ unless $\mathsf{P} = \mathsf{NP}$.

We prove that $\texttt{SLDD}_+$ does not satisfy $\mathbf{CT}_{\preceq\gamma}$ unless $\mathsf{P} = \mathsf{NP}$ in a similar way: the satisfaction of $\mathbf{CT}_{\preceq\gamma}$ would allow us to compute $M_{\leq\gamma}$ and $M_{<\gamma}$ using the same technique, hence it would imply the satisfaction of $\mathbf{CT}_{\sim\gamma}$.

These two results imply that $\texttt{SLDD}_\times$ cannot satisfy $\mathbf{CT}_{\preceq\gamma}$ or $\mathbf{CT}_{\succeq\gamma}$ unless $\mathsf{P} = \mathsf{NP}$. Indeed, an $\texttt{SLDD}_+$ representation of a function g can be transformed in polynomial time into an $\texttt{SLDD}_\times$ representation of the function $h = 2^g$, simply by replacing the label of the leaf by 1 and the value φ of each arc (including the offset) by 2^φ. If $\texttt{SLDD}_\times$ satisfied $\mathbf{CT}_{\preceq\gamma}$ (resp. $\mathbf{CT}_{\succeq\gamma}$), we could obtain in polynomial time the number of assignments $\vec{x}$ such that $2^{g(\vec{x})} \leq 2^\gamma$ (resp. $2^{g(\vec{x})} \geq 2^\gamma$), which is the same number as $\mathit{CUT}^{\preceq\gamma}(g)$ (resp. $\mathit{CUT}^{\succeq\gamma}(g)$). Thus the satisfaction of $\mathbf{CT}_{\preceq\gamma}$ (resp. $\mathbf{CT}_{\succeq\gamma}$) on $\texttt{SLDD}_\times$ would imply its satisfaction on $\texttt{SLDD}_+$, yet we have just proved that it is not satisfied by $\texttt{SLDD}_+$ unless $\mathsf{P} = \mathsf{NP}$.

Finally, $\texttt{AADD}$ does not satisfy $\mathbf{CT}_{\succeq\gamma}$ (resp. $\mathbf{CT}_{\preceq\gamma}$), because this would imply that $\texttt{SLDD}_+$ also satisfies it, since $\texttt{SLDD}_+ \geq_\ell \texttt{AADD}$ (Proposition A.7). $\square$

Proposition A.27. $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$*, and* $\texttt{AADD}$ *do not satisfy* $\mathbf{CUT}_{\sim\gamma}$.

Proof. We consider a set $\mathcal{X} = \{y_1, \ldots, y_n, z_1, \ldots, z_n\}$ of $2n$ Boolean variables, and $\lhd$ the total order on $\mathcal{X}$ defined by $y_1 \lhd \cdots \lhd y_n \lhd z_1 \lhd \cdots \lhd z_n$. Let f^{sum} and f^{prod} be the functions defined by $f^{\mathrm{sum}}(\vec{y} \cdot \vec{z}) = \sum_{i=1}^n y_i 2^{n-i} + \sum_{i=1}^n z_i 2^{n-i}$, and $f^{\mathrm{prod}}(\vec{y} \cdot \vec{z}) = 2^{f^{\mathrm{sum}}(\vec{y}\cdot\vec{z})}$. Figure 2 shows how f^{sum} (resp. f^{prod}) can be represented as an $\texttt{SLDD}_+$ (resp. $\texttt{SLDD}_\times$) formula ordered by $\lhd$ containing only $2n+1$ nodes and $2n$ arcs. Since $\texttt{SLDD}_+ \geq_\ell \texttt{AADD}$ (Proposition A.7), f^{sum} also has a linear-sized $\texttt{AADD}$ representation.

Let f^{eq} be the Boolean function defined by $f^{\mathrm{eq}}(\vec{y} \cdot \vec{z}) = 1$ if $f^{\mathrm{sum}}(\vec{y} \cdot \vec{z}) = 2^n$, and 0 otherwise. Note that f^{eq} also equals 1 if and only if $f^{\mathrm{prod}}(\vec{y} \cdot \vec{z}) = 2^{2^n}$.

We now show that there is no polynomial-sized representation of f^{eq} as an $\texttt{OBDD}_\lhd$ formula. The proof basically relies on a result by Sieling and Wegener (1993), showing that for any Boolean function f over $\mathcal{X}$, if the number of restrictions on $\{y_1, \ldots, y_n\}$ which are distinct and depends on z_1 is equal to m, then any $\texttt{OBDD}_\lhd$ formula representing f contains at least m nodes labeled with z_1. We show that it is the case for f^{eq}.

Clearly enough, every assignment $\vec{y}$ over $\{y_1, \ldots, y_n\}$ (resp. $\vec{z}$ over $\{z_1, \ldots, z_n\}$) is associated in a bijective way with a natural number $\mathrm{N}(\vec{y}) = \sum_{i=1}^n y_i 2^{n-i}$ (resp. $\mathrm{N}(\vec{z}) = \sum_{i=1}^n z_i 2^{n-i}$) which belongs to $[0, 2^n - 1]$. In particular, there are 2^n distinct assignments over $\{y_1, \ldots, y_n\}$. Consider now two distinct assignments $\vec{y}$ and $\vec{y}'$ over $\{y_1, \ldots, y_n\}$. The two restrictions $f^{\mathrm{eq}}_{\vec{y}}$ and $f^{\mathrm{eq}}_{\vec{y}'}$ are distinct, since they have distinct support sets (i.e., the set of assignments over $\{z_1, \ldots, z_n\}$ that makes the restriction equal to 1), namely $\{\, \vec{z} \mid \mathrm{N}(\vec{y}) + \mathrm{N}(\vec{z}) = 2^n \,\}$ and $\{\, \vec{z} \mid \mathrm{N}(\vec{y}') + \mathrm{N}(\vec{z}) = 2^n \,\}$, which are singletons (there is exactly one $\vec{z}$ for each $\vec{y}$).

Furthermore, for each assignment $\vec{y}$ over $\{y_1, \ldots, y_n\}$, $f^{\mathrm{eq}}_{\vec{y}}$ depends on z_1, i.e., $f^{\mathrm{eq}}_{\vec{y}\cdot\langle z_1,0\rangle} \neq f^{\mathrm{eq}}_{\vec{y}\cdot\langle z_1,1\rangle}$: this is obvious since the support set of $f^{\mathrm{eq}}_{\vec{y}}$ is a singleton (one of the two functions always returns 0, whereas the other one returns 1 for exactly one assignment).

Consequently, according to the abovementioned result, any $\texttt{OBDD}_\lhd$ formula representing f^{eq} contains at least $2^n - 1$ nodes labeled with z_1: there can be no polynomial-sized $\texttt{OBDD}_\lhd$ representation of f^{eq}.

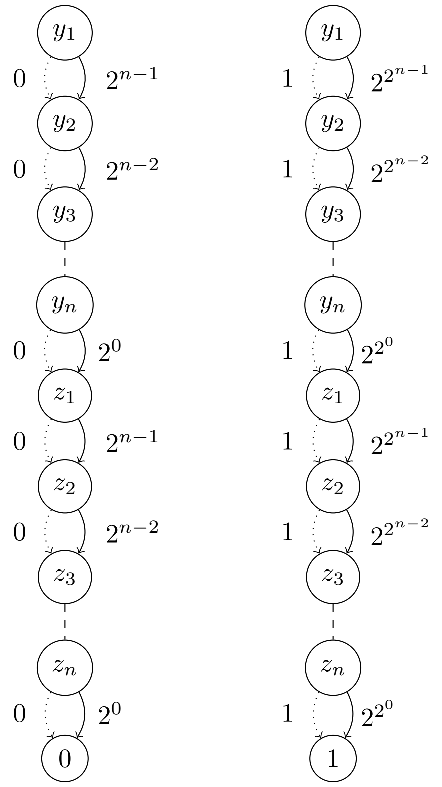

Figure 2: Left: An $\texttt{SLDD}_+$ representation of f^{sum}. Right: An $\texttt{SLDD}_\times$ representation of f^{prod}.

Now, note that the model set of f^{eq} is exactly $\mathit{CUT}^{\sim\gamma}(f^{\mathrm{sum}}) = \mathit{CUT}^{\sim\gamma}(f^{\mathrm{prod}})$ with $\gamma = 2^n$. This means that if $\texttt{SLDD}_+$ (resp. $\texttt{SLDD}_\times$, $\texttt{AADD}$) satisfied $\mathbf{CUT}_{\sim\gamma}$, we could get in time polynomial in n an $\texttt{SLDD}_+$ (resp. $\texttt{SLDD}_\times$, $\texttt{AADD}$) representation of f^{eq} with $a = 1$ and $b = 0$, that we could in turn translate in polynomial time into an $\texttt{OBDD}_\lhd$ formula (using Proposition A.8, and remarking that an MDD over Boolean variables is an OBDD). By contradiction, $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, and $\texttt{AADD}$ do not satisfy $\mathbf{CUT}_{\sim\gamma}$. $\square$

Proposition A.28. $\texttt{SLDD}_+$ *does not satisfy* $\mathbf{CUT}_{\succeq\gamma}$ *or* $\mathbf{CUT}_{\preceq\gamma}$.

Proof. The first step is to show that $\texttt{SLDD}_+$ satisfies $\mathbf{CUT}_{\succeq\gamma}$ if and only if it satisfies $\mathbf{CUT}_{\preceq\gamma}$. Let $a, b \in \mathbb{R}^+$ such that $a > b$, and let $\gamma \in \mathbb{R}^+$.

Suppose $\texttt{SLDD}_+$ satisfies $\mathbf{CUT}_{\succeq\gamma}$. Computing the maximal value v^* of $f_\alpha^{\texttt{SLDD}_+}$, which can be done in polynomial time (Proposition A.13), and choosing a constant $K \geq \max(v^*, \gamma)$, Lemma A.25 states that we can build in polynomial time an $\texttt{SLDD}_+$ representation α' of the function $K - f_\alpha^{\texttt{SLDD}_+}$. Now, applying the $\mathbf{CUT}_{\succeq\gamma}$ transformation to α' with a threshold of $K - \gamma$, which by hypothesis can be done in polynomial time, we get an $\texttt{SLDD}_+$ representation of the function g defined by $g(\vec{x}) = a$ if $f_{\alpha'}^{\texttt{SLDD}+}(\vec{x}) \geq K - \gamma$, i.e., if $f_{\alpha}^{\texttt{SLDD}+}(\vec{x}) \leq \gamma$, and $g(\vec{x}) = b$ otherwise: $\mathbf{CUT}_{\preceq\gamma}$ is satisfied. This proves that if $\texttt{SLDD}_+$ satisfies $\mathbf{CUT}_{\succeq\gamma}$, it also satisfies $\mathbf{CUT}_{\preceq\gamma}$. The same reasoning can be used to show that if it satisfies $\mathbf{CUT}_{\preceq\gamma}$, it also satisfies $\mathbf{CUT}_{\succeq\gamma}$. Hence, the satisfaction of either of the two implies the satisfaction of both. We now show that it also implies the satisfaction of $\mathbf{CUT}_{\sim\gamma}$.

Let $\gamma \in \mathbb{R}^+$ and let α be an $\texttt{SLDD}_+$ formula. We apply the $\mathbf{CUT}_{\succeq\gamma}$ and $\mathbf{CUT}_{\preceq\gamma}$ transformations on α, with $a = 1$ and $b = 0$. The resulting $\texttt{SLDD}_+$ formulæ can be turned in polynomial time into ordered MDDs (Proposition A.8), and since $\texttt{MDD}$ satisfies $\wedge\mathbf{BC}$ (Amilhastre et al. 2012), we can obtain in polynomial time an MDD the models of which are exactly the assignments that are in $CUT^{\succeq\gamma}(f_\alpha) \cap CUT^{\preceq\gamma}(f_\alpha)$, i.e., an MDD representing $CUT^{\sim\gamma}(f_\alpha)$. We can replace the 0-leaf by some $b \in \mathbb{R}^+$ and the 1-leaf by some $a \in \mathbb{R}^+$ such that $a > b$: we get an $\texttt{ADD}$ representation of the function h defined by $h(\vec{x}) = a$ if $\vec{x} \in CUT^{\sim\gamma}(f_\alpha)$ and $h(\vec{x}) = b$ otherwise. Since $\texttt{ADD} \geq_\ell \texttt{SLDD}_+$ (Proposition A.7), this means that $\texttt{SLDD}_+$ satisfies $\mathbf{CUT}_{\sim\gamma}$.

All in all, if $\texttt{SLDD}_+$ satisfies $\mathbf{CUT}_{\succeq\gamma}$ (resp. $\mathbf{CUT}_{\preceq\gamma}$), it also satisfies $\mathbf{CUT}_{\preceq\gamma}$ (resp. $\mathbf{CUT}_{\succeq\gamma}$), which in turn implies that it satisfies $\mathbf{CUT}_{\sim\gamma}$, yet we have shown that it does not (Proposition A.27). Hence $\texttt{SLDD}_+$ does not satisfy $\mathbf{CUT}_{\succeq\gamma}$ or $\mathbf{CUT}_{\preceq\gamma}$. □

Corollary A.29. $\texttt{SLDD}_\times$ *and* $\texttt{AADD}$ *do not satisfy* $\mathbf{CUT}_{\succeq\gamma}$ *or* $\mathbf{CUT}_{\preceq\gamma}$.

Proof. We show that if $\texttt{SLDD}_\times$ satisfied $\mathbf{CUT}_{\succeq\gamma}$ (resp. $\mathbf{CUT}_{\preceq\gamma}$), $\texttt{SLDD}_+$ would also satisfy it. Indeed, an $\texttt{SLDD}_+$ representation α of a function f can be transformed in polynomial time into an $\texttt{SLDD}_\times$ representation α' of the function $g = 2^f$, simply by replacing the label of the leaf by 1 and the value φ of each arc (including the offset) by 2^φ. Suppose $\texttt{SLDD}_\times$ satisfies $\mathbf{CUT}_{\succeq\gamma}$ (resp. $\mathbf{CUT}_{\preceq\gamma}$): taking $a = 1$ and $b = 0$, we could obtain in polynomial time an $\texttt{SLDD}_\times$ representing $CUT^{\succeq\gamma}(f)$ (resp. $CUT^{\preceq\gamma}(f)$), by taking the cut of α' w.r.t. the threshold 2^γ. We can transform the result into an MDD, which only takes polynomial time (Proposition A.8), then replace the 0-leaf by some $b \in \mathbb{R}^+$ and the 1-leaf by some $a \in \mathbb{R}^+$ such that $a > b$: we get an $\texttt{ADD}$ representation of the function h defined by $h(\vec{x}) = a$ if $\vec{x}$ is in $CUT^{\succeq\gamma}(f)$ (resp. $CUT^{\preceq\gamma}(f)$) and $h(\vec{x}) = b$ otherwise. Since $\texttt{ADD} \geq_\ell \texttt{SLDD}_+$ (Proposition A.7), this means that $\texttt{SLDD}_+$ satisfies $\mathbf{CUT}_{\succeq\gamma}$ (resp. $\mathbf{CUT}_{\preceq\gamma}$), yet we have shown that it does not (Proposition A.28). Consequently, $\texttt{SLDD}_\times$ cannot satisfy $\mathbf{CUT}_{\succeq\gamma}$ (resp. $\mathbf{CUT}_{\preceq\gamma}$).

Now, using a reasoning similar to that of the previous paragraph, we show that if $\texttt{AADD}$ satisfied $\mathbf{CUT}_{\succeq\gamma}$ (resp. $\mathbf{CUT}_{\preceq\gamma}$), $\texttt{SLDD}_+$ also would. Indeed, an $\texttt{SLDD}_+$ formula can be turned in polynomial time into an $\texttt{AADD}$ formula (Proposition A.7); then we could apply $\mathbf{CUT}_{\succeq\gamma}$ (resp. $\mathbf{CUT}_{\preceq\gamma}$) on values 0 and 1, so that the resulting $\texttt{AADD}$ formula can be turned in polynomial time into an MDD (Proposition A.8); then the leaves of the MDD can be re-labeled with any $a, b \in \mathbb{R}^+$, and the resulting $\texttt{ADD}$ formula be turned in polynomial time into an $\texttt{SLDD}_+$ (Proposition A.7). □

Combinations

Proposition A.30. $\texttt{ADD}$, $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, *and* $\texttt{AADD}$ *do not satisfy* $+\mathbf{C}$, $\times\mathbf{C}$, $\max\mathbf{C}$, *or* $\min\mathbf{C}$.

Proof. The proofs rely on results in the Boolean case: Wegener (1987) has shown that there exists a family of formulæ Σ_n over $\{x_1, \dots, x_n\}$ where each Σ_n has a number of prime implicates cubic in n (hence each Σ_n has a polynomial-sized CNF representation) but every $\texttt{OBDD}$ representation of Σ_n has a size exponential in n. We use the Σ_n family in all the following proofs.

$\texttt{ADD}$ **does not satisfy** $+\mathbf{C}$. For each n, for each of the clauses δ_i of Σ_n, let α_i be an $\texttt{ADD}$ formula of size linear in the size of δ_i (hence linear in n) representing a term equivalent to $\neg\delta_i$, i.e., such that for each assignment $\vec{x}$ over $\{x_1, \dots, x_n\}$, $f_{\alpha_i}^{\texttt{ADD}}(\vec{x}) = 0$ if $\vec{x}$ satisfies δ_i, $f_{\alpha_i}^{\texttt{ADD}}(\vec{x}) = 1$ otherwise. Suppose that $\texttt{ADD}$ satisfied $+\mathbf{C}$. In this case, it would be possible to compute in time polynomial in n an $\texttt{ADD}$ representation of the function that associates with each assignment $\vec{x}$ the number of clauses of Σ_n violated by $\vec{x}$. Since $\texttt{ADD}$ satisfies $\mathbf{CUT}_{\min}$ (Proposition A.14), we could then compute from it in polynomial time an $\texttt{ADD}$ representation with $a = 1$ and $b = 0$ of the set of assignments $\vec{x}$ such that $f_{\Sigma_n}(\vec{x}) = 1$ if $\vec{x}$ does not violate any clause, $f_{\Sigma_n}(\vec{x}) = 0$ otherwise. By construction, this $\texttt{ADD}$ representation is also an $\texttt{OBDD}$ representation of Σ_n. This would contradict the fact that every $\texttt{OBDD}$ representation of Σ_n has exponential size.

$\texttt{ADD}$ **does not satisfy** $\max\mathbf{C}$. Consider the $\texttt{ADD}$ formulæ α_i as in the proof for the $+\mathbf{C}$ case. If $\texttt{ADD}$ satisfied $\max\mathbf{C}$, then it would be possible to compute in time polynomial in n an $\texttt{ADD}$ representation of the function f_{Σ_n} that associates with each assignment $\vec{x}$ the value $\max_i f_{\alpha_i}^{\texttt{ADD}}(\vec{x})$. By construction, this value is equal to 1 if Σ is violated by $\vec{x}$ and is equal to 0 otherwise. Hence the $\texttt{ADD}$ representation of the function f_{Σ_n} also is an $\texttt{OBDD}$ representation of $\neg\Sigma_n$. An $\texttt{OBDD}$ representation of Σ_n could be easily obtained from it by labeling the 0-leaf with 1 and the 1-leaf with 0. This would contradict the fact that every $\texttt{OBDD}$ representation of Σ_n has exponential size.

$\texttt{ADD}$ **does not satisfy** $\times\mathbf{C}$. For each of the clauses δ_i of Σ_n, let α_i be an $\texttt{ADD}$ representation of δ_i. Each α_i can be easily generated in time linear in the size of δ_i, hence in time linear in n. If $\texttt{ADD}$ satisfied $\times\mathbf{C}$, it would be possible to compute in time polynomial in n an $\texttt{ADD}$ representation of the function f_{Σ_n} that associates with each assignment $\vec{x}$ the value $\prod_i f_{\alpha_i}^{\texttt{ADD}}(\vec{x})$. This value is equal to 1 if Σ_n is satisfied by $\vec{x}$ and to 0 otherwise. Hence it is an $\texttt{OBDD}$ representation of Σ_n, contradiction.

$\texttt{ADD}$ **does not satisfy** $\min\mathbf{C}$. For each of the clauses δ_i of Σ_n, let α_i be an $\texttt{ADD}$ representation of δ_i. Each α_i can be easily generated in time linear in the size of δ_i, hence in time linear in n. If $\texttt{ADD}$ satisfied $\min\mathbf{C}$, then it would be possible to compute in time polynomial in n an $\texttt{ADD}$ representation of the function f_{Σ_n} that associates with each assignment $\vec{x}$

the value $\min_i f^{\texttt{ADD}}_{\alpha_i}(\overrightarrow{x})$,. This value is equal to 0 if Σ is violated by $\overrightarrow{x}$ and is equal to 1 otherwise. Hence it is an OBDD representation of Σ_n, contradiction.

$\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, and AADD do not satisfy $+$C, $\times$C, $\max$C, or $\min$C. Let $\texttt{L} \in \{\texttt{SLDD}_+, \texttt{SLDD}_\times, \texttt{AADD}\}$. Any ADD formula can be transformed in linear time into an equivalent L formula (Proposition A.7). Furthermore, L satisfies $\mathbf{CUT}_{\min}$ (Proposition A.14). Finally, for each transformation, each one of the functions f_{Σ_n} considered in the above proofs takes its values in $\{0, 1\}$. Hence, thanks to Proposition A.8, each of its L representations (as computed as in the above proofs) could be turned in polynomial time into an equivalent MDD representation (which would be an OBDD representation, since every variable is a Boolean one). Once again, this would contradict the fact that every OBDD representation of Σ_n has exponential size. □

Proposition A.31. *$\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, and AADD do not satisfy* $\max\mathbf{BC}$ *or* $\min\mathbf{BC}$.

Proof. We first show that the satisfaction of $\max\mathbf{BC}$ implies that of $\mathbf{CUT}_{\preceq\gamma}$.

Let $\gamma \in \mathcal{V}$; we can easily generate in constant time an ADD representation of the constant function f_γ defined by $\forall \overrightarrow{x}, f_\gamma(\overrightarrow{x}) = \gamma$. Given Proposition A.7, we can also generate in constant time an $\texttt{SLDD}_+$ (resp. $\texttt{SLDD}_\times$, resp. AADD) representation of f_γ.

Suppose that $\texttt{L} \in \{\texttt{SLDD}_+, \texttt{SLDD}_\times, \texttt{AADD}\}$ satisfied $\max\mathbf{BC}$. Then for any L formula α, it would be possible to build in time polynomial in the size of α an L representation β of the function $g = \max(f_\alpha, f_\gamma)$. Because each of the three languages satisfies $\mathbf{OPT}_{\min}$ (Proposition A.13), we could compute in polynomial time the minimum value v_* reached by g. If $v_* > \gamma$, then $\forall \overrightarrow{x}$, $f_\alpha(\overrightarrow{x}) > \gamma$. Accordingly, $CUT^{\preceq\gamma}(f_\alpha) = \varnothing$. So the cut $f_{\preceq\gamma}$ on $\{b, c\}$ is the constant function such that $\forall \overrightarrow{x}, f_{\preceq\gamma}(\overrightarrow{x}) = c$, which can be represented in constant time as an L formula with only one node.

The case when $v_* \leq \gamma$ is more complicated, but we show that it is possible to build in polynomial time an L representation of $f_{\preceq\gamma}$, by redirecting the arcs not taking part in valuations equal to γ.

First, we can compute in polynomial time, for any arc a of an L formula α, the cost $mincost(a)$ of the cheapest assignment $\overrightarrow{x}$ such that $p(\overrightarrow{x})$ contains a.

This is clear when L is $\texttt{SLDD}_+$ or $\texttt{SLDD}_\times$: using a shortest path algorithm, we can compute in polynomial time, for any node N, the cost $min_{\text{out}}(N)$ of the cheapest path from N to the leaf, and the cost $min_{\text{in}}(N)$ of the cheapest path from the root to N. Then for any arc a in α, denoting M and N its source and destination nodes, $mincost(a) = min_{\text{in}}(M) \otimes \varphi_a \otimes min_{\text{out}}(N)$, where $\otimes$ is $+$ for $\texttt{SLDD}_+$ and $\times$ for $\texttt{SLDD}_\times$.

In the case of AADD, this is less direct. We use the fact that for any node N in a normalized AADD formula, there always exist a path from N to the leaf with valuation 0. This allows the computation of $mincost(a)$ for each arc a in one traversal of the graph from the root to the sink in topological order. The procedure is described in Algorithm 4; the idea is to compute an "offset" for each node N, representing the aggregation of the valuation pairs of each arc along the cheapest path from the root to N.

Algorithm 4: MinCostArcsAADD(α)

```
input : an AADD formula $\alpha$, of root $R$, with offset
        $\langle q_0, f_0 \rangle$
output: the value of $mincost(a)$ for each arc $a$ in $\alpha$

1  foreach node $N$ of $\alpha$ in reverse topological ordering do
2      if $N$ is the root node then
3          let $q_{\min}(N) := q_0$
4          let $f_{\min}(N) := f_0$
5      else
6          let $a_{\min} := \arg\min_{a \in \mathrm{In}(N)} mincost(a)$
7          let $q_{\min}(N) := q_{\min}(a_{\min})$
8          let $f_{\min}(N) := f_{\min}(a_{\min})$
9      foreach arc $a$ going out of $N$ do
10         let $q_{\min}(a) := q_{\min}(N) + f_{\min}(N) * q_a$
11         let $f_{\min}(N) := f_{\min}(N) * f_a$
12         let $mincost(a) = q_{\min}(a) + f_{\min}(a)$
13 return $\{ \langle a, mincost(a) \rangle \mid a$ arc in $\alpha \}$
```

It is thus possible, for any $\texttt{L} \in \{\texttt{SLDD}_+, \texttt{SLDD}_\times, \texttt{AADD}\}$, to obtain in polynomial time the value $mincost(a)$ for any arc a in the L representation α' of the function $g = \max(f_\alpha, f_\gamma)$. With a simple transformation, we can turn α' into an L representation of the cut $f_{\preceq\gamma}$ on $\{b, c\}$.

The procedure is as follows: label the leaf with b; add a new c-labeled leaf; redirect every arc a such that $mincost(a) > \gamma$ to the new c leaf; remove all arc valuations. The result is an ADD formula, which represents $f_{\preceq\gamma}$. Indeed, let $\overrightarrow{x} \in CUT^{\preceq\gamma}(f_\alpha)$; clearly, $g(\overrightarrow{x}) = \gamma$, so each arc a along the path $p(\overrightarrow{x})$ in α' is such that $mincost(a) = \gamma$: the path is still the same in β, and leads to the b-leaf.

Conversely, consider an assignment $\overrightarrow{x}$ such that $f_\beta(\overrightarrow{x}) = b$; we show that $\overrightarrow{x} \in CUT^{\preceq\gamma}(f_\alpha)$. Let us denote as $p_\beta(\overrightarrow{x})$ the path corresponding to $\overrightarrow{x}$ in β, and as $p_{\alpha'}(\overrightarrow{x})$ the corresponding path in α'. It is clear that these two paths contain the same arcs, since $p_\beta(\overrightarrow{x})$ leads to the b-leaf (no arc has been modified). Denoting $\langle a_1, \dots, a_n \rangle$ the sequence of arcs along these paths, we know by construction that for each a_i, $mincost(a_i) = \gamma$. We show by induction that for all $k \in \{1, n\}$, there exists a path starting with $a_1, \dots, a_k$ of valuation γ.

This is trivial for $k = 1$, since $mincost(a_1) = \gamma$. Suppose the hypothesis holds for some $k \in \{1, n-1\}$; we show it holds for $k + 1$. Let p_k be the path in α' from the source node of a_{k+1} to the leaf such that $\varphi(\langle a_1, \dots, a_k \rangle \cdot p_k) = \gamma$, where $\varphi(p)$ denotes the valuation of some path p. Since $mincost(a_{k+1}) = \gamma$, there exists a path of valuation γ containing a_{k+1}: let us denote p_{in} the part before a_{k+1} and p_{k+1} the part after a_{k+1}.

By construction, each path p in α' verifies $\varphi(p) \geq \gamma$ (recall that α' represents $\max(f_\alpha, f_\gamma)$). Hence, we know that the following inequalities hold:

$$\varphi(p_{\text{in}} \cdot p_k) \geq \gamma$$
$$\varphi(\langle a_1, \ldots, a_k, a_{k+1}\rangle \cdot p_{k+1}) \geq \gamma$$

(it can be easily checked that the two paths are legal paths). Now, since $\varphi(p_{\text{in}} \cdot \langle a_{k+1}\rangle \cdot p_{k+1}) = \gamma$, we can deduce from the first inequation that $\varphi(p_k) \geq \varphi(\langle a_{k+1}\rangle \cdot p_{k+1})$, of which we can deduce that $\varphi(\langle a_1, \ldots, a_k\rangle \cdot p_k) \geq \varphi(\langle a_1, \ldots, a_k, a_{k+1}\rangle \cdot p_{k+1})$ (it can be checked that these deductions are valid for any $\mathtt{L} \in \{\mathtt{SLDD}_+, \mathtt{SLDD}_\times, \mathtt{AADD}\}$).

Since $\varphi(\langle a_1, \ldots, a_k\rangle \cdot p_k) = \gamma$, it holds that $\gamma \geq \varphi(\langle a_1, \ldots, a_k, a_{k+1}\rangle \cdot p_{k+1})$; combining this result with the second inequation, we get that $\varphi(\langle a_1, \ldots, a_k, a_{k+1}\rangle \cdot p_{k+1}) = \gamma$: the proposition holds for $k+1$, therefore by induction it holds for all $k \in \{1, n\}$. The fact that it holds for $k = n$ implies that $\varphi(\langle a_1, \ldots, a_n\rangle) = \gamma$: $f_{\alpha'}(\vec{x}) = \gamma$, so $\vec{x} \in CUT^{\preceq\gamma}(f_\alpha)$.

This proves that β is an ADD representation of $f_{\preceq\gamma}$ on $\{b, c\}$; it could be transformed in linear time into an equivalent L representation (Proposition A.7). Hence, for $\mathtt{SLDD}_+$, $\mathtt{SLDD}_\times$, or AADD, the satisfaction of $\max\mathbf{BC}$ implies that of $\mathbf{CUT}_{\preceq\gamma}$; since these languages do not satisfy $\mathbf{CUT}_{\preceq\gamma}$, they cannot satisfy $\max\mathbf{BC}$.

It can be shown in a similar way that for these three languages, the satisfaction of $\min\mathbf{BC}$ implies that of $\mathbf{CUT}_{\succeq\gamma}$, considering the min-combination of α with an L representation of the constant function f_γ. Since $\mathbf{CUT}_{\succeq\gamma}$ is not satisfied by $\mathtt{SLDD}_+$, $\mathtt{SLDD}_\times$, and AADD, we conclude in the same way that these languages do not satisfy $\min\mathbf{BC}$. □

Proposition A.32.

- $\mathtt{SLDD}_+$ *and* AADD *do not satisfy* $\times\mathbf{BC}$*;*
- $\mathtt{SLDD}_\times$ *and* AADD *do not satisfy* $+\mathbf{BC}$.

Proof. The proofs of the two items work in a similar way; we first consider the case of $\times\mathbf{BC}$ on $\mathtt{SLDD}_+$ and AADD.

Let f be the function of n Boolean variables (with $n \geq 3$) defined by $\forall\vec{x}$, $f(\vec{x}) = \sum_{i=0}^{n-1} x_i \cdot 2^i$ (this is the function associating a n-bit vector with its corresponding integer), and g the function of n Boolean variables defined by $\forall\vec{x}$, $g(\vec{x}) = 2^n + \sum_{i=0}^{n-1}(1 - x_i) \cdot 2^i$. It holds that $\forall\vec{x}$, $f(\vec{x}) + g(\vec{x}) = 2^{n+1} - 1$.

We consider the variable ordering given by $x_0 \lhd x_1 \lhd \cdots \lhd x_{n-1}$.

Each of the two functions f and g has an $\mathtt{SLDD}_+$ representation with $n+1$ nodes and $2n$ arcs (one node per variable, two arcs per node). In the one of f (see Figure 3), the node at level x_i has one arc with label 1 and valuation 2^i and the other with label 0 and valuation 0. In the one of g (see Figure 4), the arc valuations are inverted at each level, and the root is associated with an offset $\varphi_0 = 2^n$.

Let us consider the following ADD representation α of $h = f \times g$. There are 2^n paths in α; let us denote them $p_0, \ldots, p_{2^n-1}$, with p_k being the one that corresponds to the assignment $\vec{x}$ such that $f(\vec{x}) = k$. The path p_k leads to a leaf of value $k(2^{n+1} - k - 1)$, that we denote as h_k. The sequence $\langle h_k\rangle_{0 \leq k < 2^n}$ is strictly increasing (since $k \leq 2^{n-1}$, $\delta_k = h_k - h_{k-1} = 2(2^n - k)$, $\delta_k > 4$ for $0 \leq k \leq 2^{n-1}$),

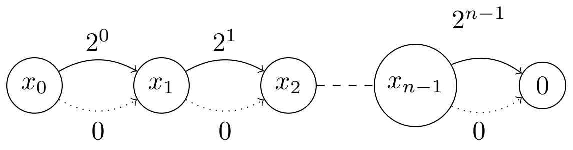

Figure 3: A $\mathtt{SLDD}_+$ representation of f

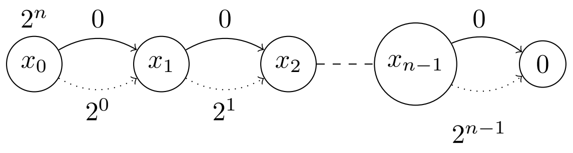

Figure 4: A $\mathtt{SLDD}_+$ representation of g

so all leaves are labeled with a different value. Now, let us build a AADD representation of h from α. Following the procedure outlined in the proof of Proposition A.7, we start with the ADD formula α, and add a neutral valuation ($\langle 0, 1\rangle$) to every arc except for those pointing to a leaf L, which receive the value $\langle\varphi(L), 0\rangle$. Stated otherwise, the last arc of each path p_k is valued $\langle h_k, 0\rangle$. Then the leaves are merged into a unique leaf, and the AADD formula obtained is normalized, following Algorithm 1. Let us suppose the algorithm is treating some node at level x_{n-1} (the level closest to the leaf); it is traversed by two paths, say p_{k-1} and p_k. We have $q_{min} = h_{k-1}$ and $range = h_k - h_{k-1} = \delta_k$, which is greater than 0, so the h_{k-1} arc receives value $\langle 0, 0\rangle$, the h_k arc receives value $\langle 1, 0\rangle$, and the unique incoming arc receives value $\langle h_{k-1}, h_k - h_{k-1}\rangle$. Clearly, all nodes at level x_{n-1} are isomorphic but none of them is redundant.

Now, let us suppose the algorithm is treating some node N at level x_{n-2}. It is traversed by four paths, say p_{4j}, p_{4j+1}, p_{4j+2}, and p_{4j+3}. Its two outgoing arcs are respectively valued with $\langle h_{4j}, \delta_{4j+1}\rangle$ and $\langle h_{4j+2}, \delta_{4j+3}\rangle$. In this case, $q_{min} = h_{4j}$ and $range = h_{4j+3} - h_{4j}$. Since $range > 0$, the 0-labeled outgoing arc of N receives the value $\langle 0, \delta_{4j+1}/range\rangle$. Let us denote v_j the multiplicative constant of this valuation: $v_j = \delta_{4j+1}/(h_{4j+3} - h_{4j})$.

It is tedious, yet not difficult, to check that the sequence $\langle v_j\rangle_{0 \leq j \leq \lfloor(2^n-1)/4\rfloor}$ is strictly increasing (recall that we suppose $n \geq 3$). Hence, the 0-labeled outgoing arcs of all nodes at level x_{n-2} are all valued with a distinct multiplicative constant: none of them can be isomorphic to another one. Furthermore, the 1-labeled outgoing arc of N receives the value $\langle(h_{4j+2} - h_{4j})/(h_{4j+3} - h_{4j}), (h_{4j+3} - h_{4j+2})/(h_{4j+3} - h_{4j})\rangle$. Since $h_{4j+2} - h_{4j} > 0$, this label differs from the one of the 0-labeled outgoing arc of N, hence N is not redundant.

Consequently, after applying the normalization and the reduction procedures, the resulting AADD formula has at least $\lfloor(2^n - 1)/4\rfloor + 1 = 2^{n-2}$ distinct nodes at level x_{n-2}. Recall that no AADD formula can be strictly smaller than an equivalent reduced and normalized AADD formula (Proposition A.6). All in all, we have the following:

(i) f and g have $\texttt{SLDD}_+$ representations of size polynomial in n;

(ii) the size of the smallest $\texttt{AADD}$ representation of $f \times g$ is exponential in n.

Since any $\texttt{SLDD}_+$ representation can be transformed into an $\texttt{AADD}$ formula in linear time (Proposition A.7):

(a) f and g have an $\texttt{AADD}$ representation of size polynomial in n, because of (i);

(b) the size of the smallest $\texttt{SLDD}_+$ representation of $f \times g$ is exponential in n, because of (ii).

To sum up, $\texttt{SLDD}_+$ does not satisfy $\times\mathbf{BC}$ (thanks to (i) and (b)), and $\texttt{AADD}$ does not satisfy $\times\mathbf{BC}$ either (thanks to (ii) and (a)).

Now, for $+\mathbf{BC}$ on $\texttt{SLDD}_\times$ and $\texttt{AADD}$, the proof is very similar to the previous one, using different functions f and g: we take $f(\vec{x}) = 2^{\sum_{i=0}^{n-1} x_i \cdot 2^i}$, and g such that $f(\vec{x}) \times g(\vec{x}) = 2^{2^{n+1}-1}$.

Both functions have $\texttt{SLDD}_\times$ representations with $n+1$ nodes and $2n$ arcs, and only exponential $\texttt{AADD}$ representations (following the same mechanism as in the previous proof). Since a $\texttt{SLDD}_\times$ representation can be transformed into an $\texttt{AADD}$ one in linear time (Proposition A.7), the results follow. □

Proposition A.33.

- $\texttt{SLDD}_+$ *satisfies* $+\mathbf{BC}$*;*
- $\texttt{SLDD}_\times$ *satisfies* $\times\mathbf{BC}$*;*
- $\texttt{ADD}$ *satisfies* $\times\mathbf{BC}$,$+\mathbf{BC}$, $\max\mathbf{BC}$, $\min\mathbf{BC}$.

Proof. Let α and α' be two $\texttt{SLDD}_\otimes$ formulæ, over $\{x_1, \dots, x_n\}$, with $\otimes \in \{+, \times\}$. They are supposed to be ordered in the same way, using the ordering $x_1 \lhd \cdots \lhd x_n$. We aim at building an $\texttt{SLDD}_\otimes$ representation of $g = f_\alpha^{\texttt{SLDD}_\otimes} \otimes f_{\alpha'}^{\texttt{SLDD}_\otimes}$ based on the same variable ordering.

First, we are going to modify α (resp. α') such that every path from the root to the leaf of the formula is of the form $\langle a_1, \dots, a_n \rangle$ where each a_i (with $i \in \{1, \dots, n-1\}$) is an arc from a node labeled with x_i to a node labeled with x_{i+1}, and a_n is an arc from a node labeled with x_n to the leaf node. Lemma A.24 states that this can be done in polynomial time.

Consider now an assignment $\vec{x}$ and let $p(\vec{x}) = \langle a_1, \dots, a_n \rangle$ and $p'(\vec{x}) = \langle a'_1, \dots, a'_n \rangle$ be the corresponding paths in α and α'. It holds that:

$$g(\vec{x}) = (\varphi(a_1) \otimes \cdots \otimes \varphi(a_n)) \otimes (\varphi(a'_1) \otimes \cdots \otimes \varphi(a'_n)).$$

Because $\otimes$ is associative and commutative, we get:

$$g(\vec{x}) = (\varphi(a_1) \otimes \varphi(a'_1)) \otimes \cdots \otimes (\varphi(a_n) \otimes \varphi(a'_n)).$$

It is thus possible to get an $\texttt{SLDD}_\otimes$ representation of g by making the product of the two graphs, levelwise: for any N of α and N' of α' such that $\mathrm{Var}(N) = \mathrm{Var}(N') = x$, the new graph, δ, contains the node denoted $N \otimes N'$ such that $\mathrm{Var}(N \otimes N') = x$; we add a leaf labeled with the neutral element for $\otimes$. For each value d in the domain of x, let a be the arc in $\mathrm{Out}(N)$ such that $v(a) = d$ and a' be the arc in $\mathrm{Out}(N')$ such that $v(a') = d$; add to the new graph one arc a_δ from node $N \otimes N'$ to node $M \otimes M'$ with value $v(a_\delta) = d$ and $\varphi(a_\delta) = \varphi(a) \otimes \varphi(a')$; then recursively delete unreachable nodes and arcs. The offset of δ is set to $\mathrm{Offset}(\alpha) \otimes \mathrm{Offset}(\alpha')$. This construction is feasible in time polynomial in the sizes of α and α'.

The resulting structure δ is a (typically non-normalized) $\texttt{SLDD}_\otimes$ formula. For any assignment $\vec{x}$, consider the corresponding path $p(\vec{x}) = \langle a''_1, \dots, a''_n \rangle$ in δ. We have:

$$\begin{aligned} f_\delta^{\texttt{SLDD}_\otimes}(\vec{x}) &= \mathrm{Offset}(\delta) \otimes \varphi(a''_1) \otimes \cdots \otimes \varphi(a''_n) \\ &= \mathrm{Offset}(\alpha) \otimes \mathrm{Offset}(\alpha') \\ &\quad \otimes \varphi(a_1) \otimes \varphi(a'_1) \otimes \cdots \otimes \varphi(a_n) \otimes \varphi(a'_n) \\ &= (\mathrm{Offset}(\alpha) \otimes \varphi(a_1) \otimes \cdots \otimes \varphi(a_n)) \\ &\quad \otimes (\mathrm{Offset}(\alpha') \otimes \varphi(a'_1) \otimes \cdots \otimes \varphi(a'_n)) \\ &= f_\alpha^{\texttt{SLDD}_\otimes}(\vec{x}) \otimes f_{\alpha'}^{\texttt{SLDD}_\otimes}(\vec{x}) \\ &= (f_\alpha^{\texttt{SLDD}_\otimes} \otimes f_{\alpha'}^{\texttt{SLDD}_\otimes})(\vec{x}). \end{aligned}$$

That is to say, δ is an $\texttt{SLDD}_+$ representation of $f_\alpha^{\texttt{SLDD}_\otimes} \otimes f_{\alpha'}^{\texttt{SLDD}_\otimes}$. This proves that $\texttt{SLDD}_+$ satisfies $+\mathbf{BC}$ and $\texttt{SLDD}_\times$ satisfies $\times\mathbf{BC}$.

Bounded combinations by $\otimes \in \{\min, \max, +, \times\}$ hold on $\texttt{ADD}$ formulæ, using the same idea of automata product. There are no valuations on the arcs, but on the leaf nodes: for each leaf N in α and each leaf M in β, the leaves of δ are nodes $N \otimes M$ labeled with $\varphi(N \otimes M) = \varphi(N) \otimes \varphi(M)$. □

Sentential Entailment

Proposition A.34. $\texttt{ADD}$ *satisfies* $\mathbf{SE}$.

Proof. Let α and β be two $\texttt{ADD}$ formulæ. We have to show that it is possible to decide in polynomial time whether $\forall \vec{x}, f_\alpha(\vec{x}) \leq f_\beta(\vec{x})$. The point is that this property holds if and only if for every value γ taken by f_α on some assignments $\vec{x}$, the set of these assignments is included in the set of assignments $\vec{y}$ such that $f_\beta(\vec{y}) \geq \gamma$. So, let Γ be the (finite) set of labels of the terminal nodes of α. For each level $\gamma \in \Gamma$, thanks to Propositions A.11 and A.8, we compute in linear time an $\texttt{MDD}$ formula $\alpha_{=\gamma}$ representing $CUT^{\sim\gamma}(f_\alpha)$ (its 1-leaf is the leaf of α labeled with γ and its 0-leaf is obtained by merging all the leaves of α not labeled with γ) and an $\texttt{MDD}$ formula $\beta_{\geq\gamma}$ representing $CUT^{\succeq\gamma}(f_\beta)$ (its 1-leaf is obtained by merging all the leaves of β labeled with a $\lambda \geq \gamma$ and its 0-leaf is obtained by merging all the leaves of α not labeled with a $\lambda < \gamma$). Checking whether every model of $\alpha_{=\gamma}$ is a model of $\beta_{\geq\gamma}$ can be done in polynomial time since $\mathbf{SE}$ is satisfied by $\texttt{MDD}$.

The procedure repeats this operation for each element of Γ, thus it runs in polynomial time. If it is the case that for each $\gamma \in \Gamma$, every model of $\alpha_{=\gamma}$ is a model of $\beta_{\geq\gamma}$, then it means that $\forall \vec{x}, f_\alpha(\vec{x}) \leq f_\beta(\vec{x})$, so the procedure outputs 1; if it is not the case, it outputs 0. □

Proposition A.35. $\texttt{SLDD}_\times$ *satisfies* $\mathbf{SE}$.

Proof. First, note that $\forall \vec{x}, f(\vec{x}) \leq g(\vec{x})$ holds if and only if (i) $\forall \vec{x}, g(\vec{x}) = 0 \implies f(\vec{x}) = 0$, and (ii) $\forall \vec{x}, f \times$

$g'(\vec{x}) \leq 1$, where $g'(\vec{x}) = 1/g(\vec{x})$ if $g(\vec{x}) > 0$ and 0 otherwise; this is not hard to check in both directions. Then, testing whether $\forall \vec{x}, f_\alpha(\vec{x}) \leq f_\beta(\vec{x})$ amounts to verify that both conditions hold.

(i) To verify that $\forall \vec{x}, f_\beta(\vec{x}) = 0 \implies f_\alpha(\vec{x}) = 0$, we only have to compute the minimal value taken by f_β (this is polynomial, since $\texttt{SLDD}_\times$ satisfies $\mathbf{OPT}_{\min}$, see Proposition A.13), and if it is 0, to compute MDD formulæ α' and β' representing the min-cuts of α and β, respectively (this is polynomial, since by Proposition A.14, $\texttt{SLDD}_\times$ satisfies $\mathbf{CUT}_{\min}$, and by Proposition A.8, an $\texttt{SLDD}_\times$ formula on $\{0,1\}$ can be transformed into an equivalent MDD in polynomial time), and to check whether $\beta' \models \alpha'$, again in polynomial time as MDD satisfies **SE** (Amilhastre et al. 2012).

(ii) To verify that $\forall \vec{x}, f \times g'(\vec{x}) \leq 1$, we first compute an $\texttt{SLDD}_\times$ representation δ of g' by inverting the offset and the label of every arc in f_β (that is, V becomes $1/V$), except for 0-labels that remain unchanged (any path in δ corresponds to the value $\prod_i 1/\varphi(a_i) = 1/\prod_i \varphi(a_i)$, except when it contains a 0-arc, so $f_\delta = g'$). It is then easy to check whether $\forall \vec{x}, f_\alpha(\vec{x}) \times g'(\vec{x}) \leq 1$: just compute the $\times$-combination of α and δ (this is polynomial because $\texttt{SLDD}_\times$ satisfies $\times\mathbf{BC}$, see Proposition A.33), compute the maximal value v^* taken by the resulting formula (polynomial because $\texttt{SLDD}_\times$ satisfies $\mathbf{OPT}_{\max}$, see Proposition A.13) and check whether $v^* \leq 1$. □

Proposition A.36. $\texttt{SLDD}_+$ *satisfies* **SE**.

Proof. First, note that $\forall \vec{x}, f(\vec{x}) \leq g(\vec{x})$ holds if and only if $\forall \vec{x}, K \leq g(\vec{x}) + K - f(\vec{x})$, where K is some constant. Thus, in order to check whether $\forall \vec{x}, f_\alpha(\vec{x}) \leq f_\beta(\vec{x})$, we are going to build an $\texttt{SLDD}_+$ formula α' such that $f_{\alpha'} = K - f_\alpha$, with K large enough for $f_{\alpha'}$ to range over $\mathbb{R}^+$. We compute v^*, the maximal value taken by f_α (this can be done in polynomial time, since $\texttt{SLDD}_+$ satisfies $\mathbf{OPT}_{\max}$, see Proposition A.13), and then choose some $K \geq v^*$. Applying Lemma A.25, we get an $\texttt{SLDD}_+$ representation α' of the function $K - f_\alpha$. We then only have to build the $+$-combination of α' and β (polynomial since $\texttt{SLDD}_+$ satisfies $+\mathbf{BC}$, see Proposition A.33) and check that the minimal value taken by the resulting formula is larger than K (polynomial since $\texttt{SLDD}_+$ satisfies $\mathbf{OPT}_{\min}$, see Proposition A.13). □

Variable Elimination

Lemma A.37. *For each language* L *among* $\{\texttt{ADD}, \texttt{SLDD}_+, \texttt{SLDD}_+, \texttt{AADD}\}$, *and each operator* $\odot \in \{\max, \min, +, \times\}$,

- L *satisfies* $\mathbf{S}\odot\mathbf{Elim}$ *if and only if it satisfies* $\odot\mathbf{C}$*;*
- L *satisfies* $\mathbf{SB}\odot\mathbf{Elim}$ *if and only if it satisfies* $\odot\mathbf{BC}$*;*

Proof. We first prove the two sufficient conditions, then the two necessary conditions.

($\Rightarrow$) If the $\odot$-elimination of a single variable in an L formula Σ can be achieved in polynomial time, it is possible to compute in polynomial time the $\odot$-combination of any set $\{\alpha_1, \ldots, \alpha_n\}$ of L formulæ. Indeed, from $\{\alpha_1, \ldots, \alpha_n\}$, we can generate in linear time the following L formula Σ: its root N_0 is labeled with a (new) variable v not occurring in any α_i, with $D_v = \{1, \ldots, n\}$; N_0 has n outgoing arcs a_i (with $i \in \{1, \ldots, n\}$)), where each a_i corresponds to $v = i$ and points to the root of α_i, and the leaves of $\alpha_1, \ldots, \alpha_n$ are merged when they have the same label; finally, when L $=$ AADD (resp. $\texttt{SLDD}_+$, resp. $\texttt{SLDD}_\times$), the label of each a_i is set to $\langle 0, 1\rangle$ (resp. 0, resp. 1). By definition, the $\odot$-elimination of v in Σ is $f^{\texttt{L}}_{\Sigma,\langle v,1\rangle} \odot \cdots \odot f^{\texttt{L}}_{\Sigma,\langle v,n\rangle}$, which by construction is equal to $f^{\texttt{L}}_{\alpha_1} \odot \cdots \odot f^{\texttt{L}}_{\alpha_n}$. Accordingly, if we can obtain in polynomial time an L representation of the $\odot$-elimination of v in Σ, we can also obtain in polynomial time an L representation of the $\odot$-combination of $\{\alpha_1, \ldots, \alpha_n\}$.

This proves that the satisfaction of $\mathbf{S}\odot\mathbf{Elim}$ implies that of $\odot\mathbf{C}$; now, note that the exact same construction also works in the bounded case. Indeed, when $n = 2$, the $\odot$-elimination of v in Σ, which can be obtain in time polynomial in $|\Sigma|^2$ if L satisfies $\mathbf{SB}\odot\mathbf{Elim}$, is $f^{\texttt{L}}_{\Sigma,\langle v,1\rangle} \odot f^{\texttt{L}}_{\Sigma,\langle v,2\rangle}$, which by construction is equal to $f^{\texttt{L}}_{\alpha_1} \odot f^{\texttt{L}}_{\alpha_2}$. This proves that the satisfaction of $\mathbf{SB}\odot\mathbf{Elim}$ implies that of $\odot\mathbf{BC}$.

($\Leftarrow$) Consider an L formula α and a variable $x \in \mathrm{Var}(\alpha)$ with a domain of size d. Since L satisfies **CD** (Proposition A.10),[6] we can build in time polynomial an L representation of $f^{\texttt{L}}_{\alpha,\vec{x}}$ for each of the d possible $\vec{x}$; we denote them as $\alpha_1, \ldots, \alpha_d$.

Now, if L satisfies $\odot\mathbf{C}$, we can obtain an L representation β of $\bigodot_{\vec{x}} f^{\texttt{L}}_{\alpha,\vec{x}}$ in time polynomial in $\sum_{i=1}^{d} |\alpha_i|$, which is polynomial in $|\alpha|$ (each α_i is of size polynomial in $|\alpha|$, and $d \leq |\alpha|$ since there is at least one x-node in α, that has by definition d outgoing arcs). By definition, β is an L representation of the $\odot$-elimination of x in α, so L satisfies $\odot\mathbf{C}$.

If L only satisfies $\odot\mathbf{BC}$, then we can also build β, but we have to do it incrementally with $d - 1$ binary $\odot$-combinations, so the bound is looser: $\alpha_1 \odot \cdots \odot \alpha_d$ is obtained in time bounded by a polynomial of $|\alpha|^d$. However, this is enough for L to satisfy $\mathbf{SB}\odot\mathbf{Elim}$, by definition. □

Proposition A.38. ADD, $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, *and* AADD *do not satisfy* $\mathbf{S}\odot\mathbf{Elim}$ *or* $\odot\mathbf{Elim}$ *for any* $\odot \in \{\max, \min, +, \times\}$.

Proof. For any $\odot \in \{\max, \min, +, \times\}$, the satisfaction of $\odot\mathbf{Elim}$ implies that of $\mathbf{S}\odot\mathbf{Elim}$, and the satisfaction of $\mathbf{S}\odot\mathbf{Elim}$ implies that of $\odot\mathbf{C}$ (Lemma A.37); yet, we have shown that ADD, $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$, and AADD do not satisfy $\odot\mathbf{C}$ when $\odot \in \{\max, \min, +, \times\}$ (Proposition A.30). □

Proposition A.39. *The following results hold:*

- ADD *satisfies* $\mathbf{SB}\odot\mathbf{Elim}$ *for* $\odot \in \{\times, +, \min, \max\}$*;*
- $\texttt{SLDD}_+$ *satisfies* $\mathbf{SB}{+}\mathbf{Elim}$*;*
- $\texttt{SLDD}_+$ *and* AADD *do not satisfy* $\mathbf{SB}{\times}\mathbf{Elim}$*;*
- $\texttt{SLDD}_\times$ *satisfies* $\mathbf{SB}{\times}\mathbf{Elim}$*;*
- $\texttt{SLDD}_\times$ *and* AADD *do not satisfy* $\mathbf{SB}{+}\mathbf{Elim}$*;*

[6]Note that conditioning can actually be achieved in linear time, without ever increasing the size of the formula. But the proof would work even if it were not the case.

- SLDD$_+$, SLDD$_\times$, *and* AADD *do not satisfy* $\mathbf{SB}\odot\mathbf{Elim}$ *for* $\odot \in \{\max, \min\}$.

Proof. These are direct consequences of Lemma A.37, since

- ADD satisfies $\odot\mathbf{BC}$ for $\odot \in \{\times, +, \min, \max\}$ (Proposition A.33);
- SLDD$_+$ satisfies $+\mathbf{BC}$ (Proposition A.33);
- SLDD$_+$ and AADD do not satisfy $\times\mathbf{BC}$ (Proposition A.32);
- SLDD$_\times$ satisfies $\times\mathbf{BC}$ (Proposition A.33);
- SLDD$_\times$ and AADD do not satisfy $+\mathbf{BC}$ (Proposition A.32);
- SLDD$_+$, SLDD$_\times$, and AADD do not satisfy $\max\mathbf{BC}$ or $\min\mathbf{BC}$ (Proposition A.31). □

Marginalization

The marginalization proofs use the fact that the $\odot$-elimination of every variable in any L formula α (i.e., the "full" variable elimination) can be done in polynomial time. We denote as $\mathrm{Elim}_\odot(\alpha)$ the value resulting from such a "full" variable elimination, that is, the value $\bigodot_{\vec{x} \in D_{\mathrm{Var}_{\mathrm{L}}(\alpha)}} f^{\mathrm{L}}_{\alpha,\vec{x}}$.

Lemma A.40. *For any language* $\mathrm{L} \in \{\mathrm{ADD}, \mathrm{SLDD}_+, \mathrm{SLDD}_\times, \mathrm{AADD}\}$ *and any operator* $\odot \in \{\max, \min, +, \times\}$*, if there exists a polynomial algorithm mapping any* L *formula* α *to the value* $\mathrm{Elim}_\odot(\alpha)$*, then* L *satisfies* $\odot\mathbf{Marg}$.

Proof. Consider an L formula α and a variable x, denoting denoting $D_x = \{d_1, \dots, d_k\}$. The following ADD formula β is a representation of the $\odot$-marginalization of f^{L}_α on x: β has one root labeled with x, with k outgoing arcs $a_1, \dots, a_k$, such that for each $i \in \{1, \dots, k\}$, $v(a_i) = d_i$ and a_i points to a leaf L_i with value $\varphi(L_i) = \bigodot_{\vec{y} \in D_{\mathrm{Var}(\alpha)\setminus\{x\}}} f^{\mathrm{L}}_{\alpha,\vec{y}}(\langle x, d_i\rangle)$, that is, $\varphi(L_i) = \mathrm{Elim}_\odot(f^{\mathrm{L}}_{\alpha,\langle x,d_i\rangle})$. Accordingly, if for any value d_i, the valuation $\varphi(L_i)$ can be computed in time polynomial in the size of α (i.e., if "full" variable elimination can be computed in polynomial time from α once conditioned by $x = d_i$), then β can be computed in time polynomial in the size of α.

Since all four languages satisfy **CD** (Proposition A.10), this implies that if there exists a polynomial algorithm for "full" variable elimination, then there exists a polynomial algorithm building an ADD representation β of the marginalization of any L formula on x. This shows the result when L is ADD; now when L is SLDD$_+$ (resp. SLDD$_\times$, resp. AADD), β can be turned in linear time into an equivalent SLDD$_+$ (resp. SLDD$_\times$, resp. AADD) formula (Proposition A.7). □

Proposition A.41. ADD, SLDD$_+$, SLDD$_\times$, *and* AADD *satisfy* $\odot\mathbf{Marg}$ *for* $\odot \in \{\max, \min\}$.

Proof. When $\odot$ is max (resp. min), $\mathrm{Elim}_\odot(\alpha)$ is simply the maximal (resp. minimal) value taken by f^{L}_α. Since from Proposition A.13, each language in $\{\mathrm{ADD}, \mathrm{SLDD}_+, \mathrm{SLDD}_\times, \mathrm{AADD}\}$ satisfies $\mathbf{OPT}_{\max}$ (resp. $\mathbf{OPT}_{\min}$), "full" variable max-elimination (resp. min-elimination) can be done in polynomial time; hence from Lemma A.40 we get that these languages all satisfy $\max\mathbf{Marg}$ (resp. $\min\mathbf{Marg}$). □

Proposition A.42. *There exists a polynomial-time algorithm mapping any* AADD *formula* α *to* $\mathrm{Elim}_+(\alpha)$.

Proof. We show that "full" variable $+$-elimination on AADD is polynomial because we can iteratively eliminate the last variable in linear time.

Let $X = \{x_1, \dots, x_n\} \subseteq \mathcal{X}$ and $y \in \mathcal{X}$; let α be an AADD formula over $X \cup \{y\}$, ordered in such a way that $x_1 \lhd \cdots \lhd x_n \lhd y$. To simplify the proof, we suppose that every variable in $X \cup \{y\}$ is mentioned in every path of α; this is harmless, since we never need to add more than $(n+1)d$ arcs (with φ-value $\langle 0, 1\rangle$) per node in α (with d the cardinal of the largest variable domain).

We have the following:

$$\begin{aligned}\mathrm{Elim}_+(\alpha) &= \sum_{\vec{z} \in D_X \times D_y} f^{\mathrm{AADD}}_\alpha(\vec{z}) \\ &= \sum_{\vec{x} \in D_X} \sum_{\vec{y} \in D_y} f^{\mathrm{AADD}}_\alpha(\vec{x} \cdot \vec{y})\end{aligned}$$

For a given assignment $\vec{x} \cdot \vec{y}$, let us consider the path p in α corresponding to $\vec{x} \cdot \vec{y}$. The path p contains $n+1$ arcs, that we denote $a_1, \dots, a_n, a_{n+1}$. For each $i \in \{1, \dots, n+1\}$, we denote $\varphi(a_i) = \langle q_i, f_i\rangle$; finally, the offset is as usual $\langle q_0, f_0\rangle$. By definition of the interpretation function of AADD,

$$\begin{aligned}f^{\mathrm{AADD}}_\alpha(\vec{x} \cdot \vec{y}) &= q_0 + f_0(q_1 + f_1(q_2 + \cdots \\ &\qquad \cdots + f_n(q_{n+1} + f_{n+1} \times 0) \cdots)) \\ &= q_0 + f_0 q_1 + f_0 f_1 q_2 + \cdots \\ &\qquad \cdots + f_0 \cdots f_n q_{n+1} \\ &= \sum_{i=0}^{n+1} \left(q_i \prod_{j=0}^{i-1} f_j \right) \\ &= \sum_{i=0}^{n} \left(q_i \prod_{j=0}^{i-1} f_j \right) + q_{n+1} \cdot \prod_{j=0}^{n} f_j.\end{aligned}$$

There are three elements in this formula:

- the valuation q_{n+1} of the y-arc, which is the last arc along the path, and which we denote as $\varphi_{\vec{x}\cdot\vec{y}}$;
- the "additive offset" given by the first n arcs, $\sum_{i=0}^{n} \left(q_i \prod_{j=0}^{i-1} f_j \right)$, which we denote as $Q_{\vec{x}}$;
- the "multiplicative offset" given by the first n arcs, $\prod_{j=0}^{n} f_j$, that we denote as $F_{\vec{x}}$.

The key point is that neither $Q_{\vec{x}}$ nor $F_{\vec{x}}$ depends on $\vec{y}$, so we can write

$$\begin{aligned}\mathrm{Elim}_+(\alpha) &= \sum_{\vec{x} \in D_X} \sum_{\vec{y} \in D_y} f^{\mathrm{AADD}}_\alpha(\vec{x} \cdot \vec{y}) \\ &= \sum_{\vec{x} \in D_X} \sum_{\vec{y} \in D_y} (Q_{\vec{x}} + \varphi_{\vec{x}\cdot\vec{y}} \cdot F_{\vec{x}}) \\ &= \sum_{\vec{x} \in D_X} \left(|D_y| \cdot Q_{\vec{x}} + F_{\vec{x}} \cdot \sum_{\vec{y} \in D_y} \varphi_{\vec{x}\cdot\vec{y}} \right). \end{aligned} \tag{1}$$

It is thus possible to transform α into another AADD formula α' that does not mention y and verifies $\mathrm{Elim}_+(\alpha') = \mathrm{Elim}_+(\alpha)$ as follows. For each y-labeled node N, we compute the value $\varphi_N = \sum_{a \in \mathrm{Out}(N)} q_a$. Then we update the φ-label of each arc $a_{\mathrm{In}} \in \mathrm{In}(N)$ so that it becomes $\langle q_{a_{\mathrm{In}}} + \frac{f_{a_{\mathrm{In}}}}{|D_y|}\varphi_N, 0\rangle$. We also modify the offset, which becomes $\langle q_0 \cdot |D_y|, f_0 \cdot |D_y|\rangle$. Finally, we merge all the y-nodes into a new leaf.

We show that $\mathrm{Elim}_+(\alpha') = \mathrm{Elim}_+(\alpha)$; let us denote $\langle q', f'\rangle$ the φ-label of an arc in α' when the corresponding arc in α has label $\langle q, f\rangle$. We can rewrite the left part of the sum in Equation 1:

$$\begin{aligned}
|D_y| \cdot Q_{\vec{x}} &= |D_y| \cdot \sum_{i=0}^{n} \left(q_i \prod_{j=0}^{i-1} f_j \right) \\
&= |D_y| \cdot q_0 + \sum_{i=1}^{n} \left(q_i \cdot |D_y| \cdot f_0 \prod_{j=1}^{i-1} f_j \right) \\
&= q'_0 + \sum_{i=1}^{n} \left(q_i f'_0 \prod_{j=1}^{i-1} f_j \right) \\
&= q'_0 + \sum_{i=1}^{n} \left(q_i \prod_{j=0}^{i-1} f'_j \right),
\end{aligned}$$

since for $i \in \{1, \ldots, n-1\}$, $f'_i = f_i$. Isolating q_n, we get

$$\begin{aligned}
|D_y| \cdot Q_{\vec{x}} &= q'_0 + \sum_{i=1}^{n-1} \left(q_i \prod_{j=0}^{i-1} f'_j \right) + q_n \prod_{j=0}^{n-1} f'_j \\
&= \sum_{i=0}^{n-1} \left(q'_i \prod_{j=0}^{i-1} f'_j \right) + q_n \prod_{j=0}^{n-1} f'_j,
\end{aligned}$$

since for $i \in \{1, \ldots, n-1\}$, $q'_i = q_i$. As for the right part of the sum in Equation 1:

$$\begin{aligned}
F_{\vec{x}} \cdot \sum_{\vec{y} \in D_y} \varphi_{\vec{x}\cdot\vec{y}} &= f_0 \left(\prod_{j=1}^{n-1} f_j \right) f_n \sum_{\vec{y} \in D_y} \varphi_{\vec{x}\cdot\vec{y}} \\
&= \left(\prod_{j=0}^{n-1} f'_j \right) \frac{f_n}{|D_y|} \sum_{\vec{y} \in D_y} \varphi_{\vec{x}\cdot\vec{y}},
\end{aligned}$$

since $f'_0 = f_0 \cdot |D_y|$ and for $i \in \{1, \ldots, n-1\}$, $f'_i = f_i$. Now, since $q'_n = q_n + \frac{f_n}{|D_y|} \sum_{\vec{y} \in D_y} \varphi_{\vec{x}\cdot\vec{y}}$, by summing the two parts, we get

$$\begin{aligned}
&|D_y| \cdot Q_{\vec{x}} + F_{\vec{x}} \cdot \sum_{\vec{y} \in D_y} \varphi_{\vec{x}\cdot\vec{y}} \\
&\quad = \sum_{i=0}^{n-1} \left(q'_i \prod_{j=0}^{i-1} f'_j \right) + \left(\prod_{j=0}^{n-1} f'_j \right) q'_n \\
&\quad = \sum_{i=0}^{n} \left(q'_i \prod_{j=0}^{i-1} f'_j \right) = f_{\alpha'}^{\mathtt{AADD}}(\vec{x}),
\end{aligned}$$

by definition of the interpretation function of AADD. Equation 1 can hence be rewritten as

$$\mathrm{Elim}_+(\alpha) = \sum_{\vec{x} \in D_X} f_{\alpha'}^{\mathtt{AADD}}(\vec{x}) = \mathrm{Elim}_+(\alpha').$$

The procedure thus eliminates the last variable without changing the value of the "full" variable elimination; moreover, it runs in linear time, and the resulting formula is always smaller than the original one. Hence, iteratively repeating the procedure for each variable in a bottom-up way, and stopping when the resulting formula only contains a leaf and an offset, we obtain $\mathrm{Elim}_+(\alpha)$ in time polynomial in the size of α. $\square$

Corollary A.43. *ADD, $\mathtt{SLDD}_+$, $\mathtt{SLDD}_\times$, and AADD satisfy* $+\mathbf{Marg}$.

Proof. Full variable $+$-elimination is polynomial on AADD. Since any ADD (resp. $\mathtt{SLDD}_+$, $\mathtt{SLDD}_\times$) formula can be turned into an equivalent AADD formula in linear time (Proposition A.7), full variable $+$-elimination is also polynomial on ADD (resp. $\mathtt{SLDD}_+$, $\mathtt{SLDD}_\times$). Hence, from Lemma A.40, we get that all four languages satisfy $+\mathbf{Marg}$. $\square$

Proposition A.44. *There exists a polynomial-time algorithm mapping any* $\mathtt{SLDD}_\times$ *formula* α *to* $\mathrm{Elim}_\times(\alpha)$.

Proof. The proof is similar to that of full $+$-elimination in AADD: we show that we can iteratively eliminate the last variable in linear time. Let $X = \{x_1, \ldots, x_n\} \subseteq \mathcal{X}$ and $y \in \mathcal{X}$; let α be an $\mathtt{SLDD}_\times$ formula over $X \cup \{y\}$, ordered in such a way that $x_1 \triangleleft \cdots \triangleleft x_n \triangleleft y$. To simplify the proof, we suppose that every variable in $X \cup \{y\}$ is mentioned in every path of α. Lemma A.24 shows that this can be done in polynomial time.

We have the following:

$$\begin{aligned}
\mathrm{Elim}_\times(\alpha) &= \prod_{\vec{z} \in D_X \times D_y} f_\alpha^{\mathtt{SLDD}_\times}(\vec{z}) \\
&= \prod_{\vec{x} \in D_X} \prod_{\vec{y} \in D_y} f_\alpha^{\mathtt{SLDD}_\times}(\vec{x} \cdot \vec{y})
\end{aligned}$$

For a given assignment $\vec{x} \cdot \vec{y}$, let us consider the path p in α corresponding to $\vec{x} \cdot \vec{y}$. The path p contains $n+1$ arcs: we denote as $\Phi_{\vec{x}}$ the product of the φ-labels of the first n arcs and the offset, and as $\varphi_{\vec{x}\cdot\vec{y}}$ the φ-label of the last arc. By definition of the interpretation function of $\mathtt{SLDD}_\times$,

$$f_\alpha^{\mathtt{SLDD}_\times}(\vec{x} \cdot \vec{y}) = \Phi_{\vec{x}} \times \varphi_{\vec{x}\cdot\vec{y}},$$

and since $\Phi_{\vec{x}}$ does not depend on $\vec{y}$, we get

$$\begin{aligned}
\mathrm{Elim}_\times(\alpha) &= \prod_{\vec{x} \in D_X} \prod_{\vec{y} \in D_y} \Phi_{\vec{x}} \times \varphi_{\vec{x}\cdot\vec{y}} \\
&= \prod_{\vec{x} \in D_X} \left(\Phi_{\vec{x}} \prod_{\vec{y} \in D_y} \varphi_{\vec{x}\cdot\vec{y}} \right).
\end{aligned}$$

Hence, it is possible to transform α into another $\mathtt{SLDD}_\times$ formula α' that does not mention y and verifies $\mathrm{Elim}_\times(\alpha') = \mathrm{Elim}_\times(\alpha)$ as follows. For each y-labeled

Table 3: Results about basic queries, optimization, and γ-cutting, together with the number of the proposition (or corollary) proving each claim. The symbol legend is in Table 1.

Query	ADD	SLDD$_+$	SLDD$_\times$	AADD
$\mathbf{EQ}$	√ A.9	√ A.9	√ A.9	√ A.9
$\mathbf{SE}$	√ A.34	√ A.36	√ A.35	?
$\mathbf{OPT}_{\max}, \mathbf{OPT}_{\min}$	√ A.13	√ A.13	√ A.13	√ A.13
$\mathbf{CT}_{\max}, \mathbf{CT}_{\min}$	√ A.15	√ A.15	√ A.15	√ A.15
$\mathbf{ME}_{\max}, \mathbf{ME}_{\min}$	√ A.15	√ A.15	√ A.15	√ A.15
$\mathbf{MX}_{\max}, \mathbf{MX}_{\min}$	√ A.15	√ A.15	√ A.15	√ A.15
$\mathbf{CUT}_{\max}, \mathbf{CUT}_{\min}$	√ A.14	√ A.14	√ A.14	√ A.14
$\mathbf{VA}_{\sim\gamma}$	√ A.12	√ A.17	√ A.17	√ A.17
$\mathbf{VA}_{\succeq\gamma}, \mathbf{VA}_{\preceq\gamma}$	√ A.12	√ A.16	√ A.16	√ A.16
$\mathbf{CO}_{\sim\gamma}$	√ A.12	○ A.18	○ A.18	○ A.18
$\mathbf{CO}_{\succeq\gamma}, \mathbf{CO}_{\preceq\gamma}$	√ A.12	√ A.16	√ A.16	√ A.16
$\mathbf{ME}_{\sim\gamma}$	√ A.12	○ A.21	○ A.21	○ A.21
$\mathbf{ME}_{\succeq\gamma}, \mathbf{ME}_{\preceq\gamma}$	√ A.12	√ A.23	√ A.23	√ A.23
$\mathbf{MX}_{\sim\gamma}$	√ A.12	○ A.21	○ A.21	○ A.21
$\mathbf{MX}_{\succeq\gamma}, \mathbf{MX}_{\preceq\gamma}$	√ A.12	√ A.22	√ A.22	√ A.22
$\mathbf{CUT}_{\sim\gamma}$	√ A.11	● A.27	● A.27	● A.27
$\mathbf{CUT}_{\succeq\gamma}, \mathbf{CUT}_{\preceq\gamma}$	√ A.11	● A.28	● A.29	● A.29
$\mathbf{CT}_{\sim\gamma}$	√ A.12	○ A.21	○ A.21	○ A.21
$\mathbf{CT}_{\succeq\gamma}, \mathbf{CT}_{\preceq\gamma}$	√ A.12	○ A.26	○ A.26	○ A.26

node N, we compute the value $\varphi_N = \times_{a \in \mathrm{Out}(N)} q_a$, then we update the φ-label of each arc $a_{\mathrm{In}} \in \mathrm{In}(N)$ so that it becomes $\varphi(a_{\mathrm{In}}) \times \varphi_N$, and finally, we merge all the y-nodes into a new leaf. The process is linear in the size of α, and it should be clear that

$$f_{\alpha'}^{\mathtt{SLDD}_\times}(\vec{x}) = \Phi_{\vec{x}} \prod_{\vec{y} \in D_y} \varphi_{\vec{x}\cdot\vec{y}},$$

and thus that

$$\mathrm{Elim}_\times(\alpha) = \prod_{\vec{x} \in D_X} f_{\alpha'}^{\mathtt{SLDD}_\times}(\vec{x}) = \mathrm{Elim}_\times(\alpha').$$

Consequently, it is possible to eliminate the last variable in an SLDD$_\times$ formula in linear time, without changing the value of the "full" variable elimination. Computing this value can thus be done by eliminating every variable iteratively in a bottom-up way, stopping when the formula is reduced to a leaf and an offset; since the size of the formula is always decreasing, the overall procedure is polynomial-time. □

Corollary A.45. ADD *and* SLDD$_\times$ *satisfy* $\times\mathbf{Marg}$.

Proof. Full variable $\times$-elimination is polynomial on SLDD$_\times$. Since any ADD formula can be turned into an equivalent SLDD$_\times$ formula in linear time (Proposition A.7), full variable $\times$-elimination is also polynomial on ADD. Hence, from Lemma A.40, we get that ADD and SLDD$_\times$ satisfy $+\mathbf{Marg}$. □

Table 4: Results about transformations, together with the number of the proposition (or corollary) proving each claim. The symbol legend is in Table 1.

Transformation	ADD	SLDD$_+$	SLDD$_\times$	AADD
$\mathbf{CD}$	√ A.10	√ A.10	√ A.10	√ A.10
$\max\mathbf{BC}, \min\mathbf{BC}$	√ A.33	● A.31	● A.31	● A.31
$+\mathbf{BC}$	√ A.33	√ A.33	● A.32	● A.32
$\times\mathbf{BC}$	√ A.33	● A.32	√ A.33	● A.32
$\max\mathbf{C}, \min\mathbf{C}$	● A.30	● A.30	● A.30	● A.30
$+\mathbf{C}$	● A.30	● A.30	● A.30	● A.30
$\times\mathbf{C}$	● A.30	● A.30	● A.30	● A.30
$\max\mathbf{Elim}, \min\mathbf{Elim}$	● A.38	● A.38	● A.38	● A.38
$+\mathbf{Elim}$	● A.38	● A.38	● A.38	● A.38
$\times\mathbf{Elim}$	● A.38	● A.38	● A.38	● A.38
$\mathbf{S}\max\mathbf{Elim}, \mathbf{S}\min\mathbf{Elim}$	● A.38	● A.38	● A.38	● A.38
$\mathbf{S}{+}\mathbf{Elim}$	● A.38	● A.38	● A.38	● A.38
$\mathbf{S}{\times}\mathbf{Elim}$	● A.38	● A.38	● A.38	● A.38
$\mathbf{SB}\max\mathbf{Elim}, \mathbf{SB}\min\mathbf{Elim}$	√ A.39	● A.39	● A.39	● A.39
$\mathbf{SB}{+}\mathbf{Elim}$	√ A.39	√ A.39	● A.39	● A.39
$\mathbf{SB}{\times}\mathbf{Elim}$	√ A.39	● A.39	√ A.39	● A.39
$\max\mathbf{Marg}, \min\mathbf{Marg}$	√ A.41	√ A.41	√ A.41	√ A.41
$+\mathbf{Marg}$	√ A.43	√ A.43	√ A.43	√ A.43
$\times\mathbf{Marg}$	√ A.45	?	√ A.45	?