Functional Decision Diagrams: A Unifying Data Structure For Binary Decision Diagrams

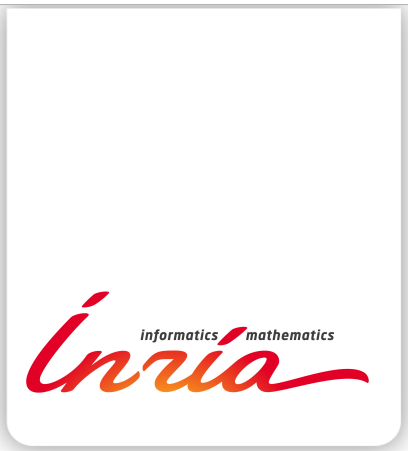
Joan Thibault, Khalil Ghorbal

Functional Decision Diagrams: A Unifying Data Structure For Binary Decision Diagrams

Joan Thibault, Khalil Ghorbal

RESEARCH REPORT
N° 9306
November 2019
Project-Teams HYCOMES

ISSN 0249-6399 ISRN INRIA/RR--9306--FR+ENG



Functional Decision Diagrams: A Unifying Data Structure For Binary Decision Diagrams

Joan Thibault, Khalil Ghorbal

Project-Teams HYCOMES

Research Report n° 9306 — November 2019 — 29 pages

Abstract: Zero-suppressed binary Decision Diagram (ZDD) is a notable alternative data structure of Reduced Ordered Binary Decision Diagram (ROBDD) that achieves a better size compression rate for Boolean functions that evaluate to zero almost everywhere. Deciding *a priori* which variant is more suitable to represent a given Boolean function is as hard as constructing the diagrams themselves. Moreover, converting a ZDD to a ROBDD (or vice versa) often has a prohibitive cost. This observation could be in fact stated about almost all existing BDD variants as it essentially stems from the non-compatibility of the reduction rules used to build such diagrams. Indeed, they are neither interchangeable nor composable. In this work, we investigate a novel functional framework, termed λDD, that ambitions to classify the already existing variants as implementations of special λDD models while suggesting, in a principled way, new models that exploit application-dependant properties to further reduce the diagram's size. We show how the reduction rules we use locally capture the global impact of each variable on the output of the entire function. Such knowledge suggests a variable ordering that sharply contrasts with the static fixed global ordering in the already existing variants as well as the dynamic reordering techniques commonly used.

Key-words: Binary Decision Diagrams, Function Abstraction, Order Abstraction

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Diagramme de Décision Fonctionel: Une Structure de Donnée Unifiante pour les Diagrammes de Décision Binaires

Résumé : Zero-suppressed binary Decision Diagram (ZDD) est le variant le plus connu des Reduced Ordered Binary Decision Diagram (ROBDD) permettant d'atteindre une représentation plus compacte en mémoire des fonctions booléennes creuses, i.e., les fonctions valant zéro presque partout. Décider á priori quel variant est plus adapté pour représenter une fonction donnée est aussi dur que construire les diagrammes eux-mêmes. De plus, convertir un ZDD en ROBDD (ou vice versa) a usuellement un cout trop élevé. Cette même observation peut être faite pour presque tous les variants de BDD dans la mesure où les règles de réductions pour calculer les diagrammes sont pour l'essentiel incompatibles. En particulier, elles ne peuvent ni commuter ni composer. Dans cet article, nous introduisons une interprétation fonctionnelle des BDD, dénommée λDD, qui a pour ambition de classifier les variants existants comme des implémentations particulières de modèles de λDD tout en suggérant, de manière systématique, de nouveaux modèles exploitants des propriétés spécifiques aux applications souhaitées. De manière à d'avantage réduire la taille des diagrammes, nous montrons comment les règles de réduction capturent, dans une certaine mesure, l'impacte global de chaque variable sur le résultat des fonctions considérées. Cette connaissance suggère un ordre local des variables: ceci différencie fortement notre approche de l'ordre statique global des variants déjà existants ou des techniques de réordonnancement dynamique. Nous appelons ces variants uniformes dans la mesure où les règles de réduction sont appliquées de manière identique à chaque variable et non spécifiquement à la première dans l'ordre choisi.

Mots-clés : Diagramme de Décision Binaire, Abstraction Fonctionelle, Abstraction d'Ordre

Introduction

A Binary Decision Diagram (BDD) is a versatile graph-based data structure, well suited in particular to effectively represent and manipulate Boolean functions. The introduction of Reduced Ordered BDD (ROBDD) by Bryant [1] in 1986 widely contributed to their adoption. Indeed, enforcing a total ordering over the variables makes the structure canonical and limits combinatorial explosion. Even though a ROBDD has an exponential worst case size, many typical applications yield a more concise representation thanks to the elimination of nodes representing useless variables (i.e., variables which do not influence the value of the output). The last thirty years have seen many BDD variants designed to capture certain application-dependant properties [2, 3, 4]. Several papers were subsequently devoted to either generalizing [5] or unifying [6] BDD variants.

In this work, we introduce a new functional framework, together with its related data structure, we term λDD. The functional models come in two flavors, *ordered* (Section 2) and *uniform* (Section 3). The ordered models serve to gradually explain the functional abstraction we perform starting from known variants. The main idea is to capture special variables, like useless variables, using functors on Boolean functions. In addition to useless variables, the so-called *canalizing* (or *grounded*) variables exploited in ZDD can be similarly captured using appropriate functors. A variable is (b,t)-canalizing (where $b,t \in \mathbb{B}$) if and only if whenever it is set to b, the entire function reduces to the constant t regardless of the valuation of the other variables. We will see how this approach allows to build new models by capturing properties of interest as special functors. Interestingly, our framework is suitable to unify, and hence classify, a significant part of already existing variants as implementations of ordered models of the same abstract data structure (see Section 4).

An important issue with almost all current variants of BDD is their sensitivity to variable ordering. Indeed, the size of the data structure may range from linear to exponential depending on the considered prefixed ordering. The uniform models suggest a novel way to tackle this problem by treating all variables equally or uniformly, in the sense that some special variables can be inserted at any position and not necessarily stacked on top of the already introduced ones as is custom. This idea turns out to be very powerful as it exposes important information on variables early on in the data structure, allowing to postpone branching as much as possible, which directly impacts the size of the diagram. We highlight the fact that these order considerations are conceptually quite different from the dynamic variable reordering used in some BDD implementations to overcome, whenever possible, the size issues (indeed, finding the optimal variable ordering is an NP-complete problem [7]).

Proving the reduction uniqueness of rich uniform λDD involves with several subtle corner cases to handle. To ensure the correctness of our models, we formalized and proved the reduction uniqueness of a rich uniform model in the proof assistant Coq [8]. As a corollary, this yields the correctness of all the simpler λDD models, both ordered and uniform.

The next section gives a functional semantics for Boolean functions based on the so-called Shannon operator.

1 Background on Boolean Functions

A Boolean function of arity $n \in \mathbb{N}$ is a form (or functional) from $\mathbb{B}^n$ to $\mathbb{B}$. It operates on an ordered tuple of Booleans of dimension n, $\{v_0, \ldots, v_{n-1}\}$, by assigning a Boolean to each of the 2^n valuations of its tuple. The set of Boolean functions of arity n, denoted by $\mathbb{B}^{n\to 1}$, is thus finite and contains 2^{2^n} elements. In particular, $\mathbb{B}^{0\to 1}$ has two elements and is isomorphic to $\mathbb{B}$ (only the types differ: functions on the one hand, and co-domain elements, or Booleans, on the

other hand). To avoid confusion, we *use a different font for* constant functions: *0* will denote the constant function of arity zero returning 0, and *1* will denote the constant function of arity zero returning 1.

This work is concerned with designing data structures to concisely encode and effectively manipulate Boolean functions. Several widely used graph-based representations rely heavily on a binary non-commutative operator, sometimes referred to as the Shannon operator in the literature, defined as follows.

Definition 1 (Star or Shannon operator)**.**

$$\begin{aligned} \star : \mathbb{B}^{n\to 1} \times \mathbb{B}^{n\to 1} &\to \mathbb{B}^{n+1\to 1} \\ (f, g) &\mapsto f \star g \end{aligned}$$

where

$$\begin{aligned} f \star g : \mathbb{B}^{n+1} &\to \mathbb{B} \\ (v_0, v_1, \ldots, v_n) &\mapsto \begin{cases} f(v_1, \ldots, v_n) & \text{if } v_0 = 0 \\ g(v_1, \ldots, v_n) & \text{if } v_0 = 1 \end{cases} \end{aligned}$$

The interest of the star operator resides in the fact that it can be used to represent any Boolean function by induction over its arity with respect to a prefixed ordering over the variables. To make this apparent in the expressions, we will annotate the operator with the name of the variable it introduces.

Whenever needed, the arity of a Boolean function will be explicitly denoted as an integer subscript between paranthesis. For instance, $f_{(n)}$ will denote a Boolean function f of arity n.

For instance, consider function $f := (x, y) \mapsto x \wedge \neg y$. It can be represented in two different ways depending on which decision variable one considers first.

$$\begin{array}{l} (0 \star_x 1) \star_y (0 \star_x 0) \\ (0 \star_y 0) \star_x (1 \star_y 0) \end{array} \tag{1}$$

The star-based representation suggests a natural encoding of Boolean functions as trees, known as Shannon Decision Trees, that are in one-to-one correspondence with Boolean functions (up to a fixed ordering of the variables). A Shannon decision tree can be efficiently compacted into a directed acyclic graph by merging all its isomorphic subgraphs. The so obtained structure is known as Shannon Decision Diagram (SDD). SDD are also in one-to-one correspondence with Boolean functions.

However, in practice, the severe drawback of these naive representation if that their size may be exponential in the number of input variables, even for simple functions. It is therefore of paramount importance to avoid branching as much as possible. In the sequel, we briefly revisit two notable BDD variants, namely ROBDD by Bryant [1] and ZDD [9, 10] by Minato, as a support to motivate and introduce our approach. Bryant's original work [1] suggests the removal of nodes with identical subgraphs since the associated variable is *useless* in the sense that its valuation does not influence the output of the function. This idea can be captured as the action of a unary operator on Boolean functions defined as follows:

$$\delta_{\mathsf{u}} : f \mapsto f \star f \ .$$

For clarity, we will also annotate such an operator with the variable's name whenever necessary. Observe, however, that by doing so, we implicitly assume that the new name introduced by the operator is not used in operand f. We stress the fact that this is only a convenience for

$\wedge$-canalizing	$\vee$-canalizing
$\delta_{\mathsf{c}_{00}} : f \mapsto 0 \star f$	$\delta_{\mathsf{c}_{01}} : f \mapsto 1 \star f$
$\delta_{\mathsf{c}_{1\partial}} : f \mapsto f \star 0$	$\delta_{\mathsf{c}_{11}} : f \mapsto f \star 1$

Table 1: Canalizing variables.

manipulating operators. Functionally speaking, the definition of the star operator, and therefore δ_{u}, is unambiguous as long as a prefixed ordering over all variables is respected.

Minato's work [9, 10] exploits the so-called *canalizing* variables, a somehow dual concept to useless variables. A variable is said to be canalizing if its sole valuation to a given value suffices to determine the value of the function regardless of the other inputs. In particular, in ZDD, a branching node is hidden (from the graph) if the valuation of its associated variable to 1 sends the output of the entire function to zero. Semantically, this idea can be also captured by the following unary operator:

$$\delta_{\mathsf{c}_{1\partial}} : f \mapsto f \star 0,$$

where the arity of the constant function 0 is adjusted to the arity of f for the operator $\star$ to apply. For example, function $g := (\delta_{\mathsf{c}_{1\partial}} f) : (x, y) \mapsto (\neg y) \wedge f(x)$ has y as canalizing. Other canalizing variables could be captured similarly. Table 1 enumerates the canalizing variables with respect to the three binary operators $\wedge$, $\vee$, and $\oplus$.

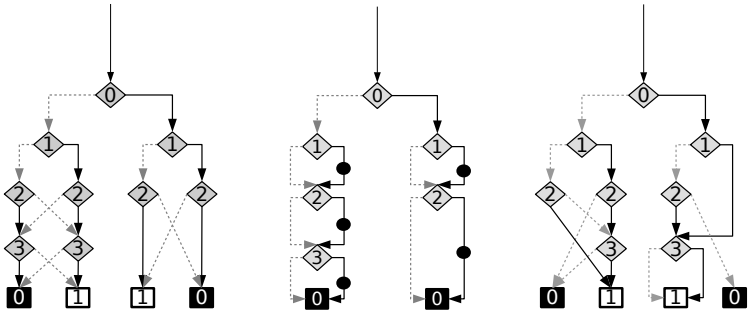
Naturally, which variant to use highly depends on the Boolean function itself. Typically, ZDD is, by design, well adapted for certain sparse functions. If one knows *a priori* what properties (e.g., sparsity) the function is likely to satisfy, one could adjust the underlying data structure. Although it is reasonable to assume such an a priori, application-related, knowledge, it is less obvious to predict which variant is suitable for some computations (e.g., transitive closure) where both sparsity and density need to be taken into consideration. What is challenging is how to translate a given property into appropriate reduction rules, and how to combine those, possibly distinct, sets of rules to benefit from their sweet spots while avoiding interference and ensuring canonicity. The functional framework we will be describing next lay down a principled way to tackle both problems whenever the property of interest can be captured as an operator that acts on Boolean functions in a specific way (like δ_{u} and $\delta_{\mathsf{c}_{1\partial}}$ for useless and canalizing variables respectively). To appreciate the differences and similarities with the data structures we will be introducing next, we end this section with three diagrams (see Figure 1) representing the function

$$(x_0, x_1, x_2, x_3) \mapsto x_0 \oplus x_3 \oplus (\neg x_1 \wedge x_2) \ ,$$

which will be our running example. Variant ROBDD+N adds a special annotation on the edges to encode negation. We will cover such aspects in more details in the following sections.

2 Ordered Models

We introduce a new data structure, akin to Binary Decision Diagrams, that we call Functional Decision Diagram or λDD. As we shall see next, the functional point of view is flexible and powerful enough to capture many interesting aspects of the already existing BDD variants and to go beyond their current capabilities. We will start by presenting the semantics of the *ordered* version of λDD before presenting the data structure itself.



(a) ROBDD [1] (b) ROBDD+N [11] (c) ZDD [9]

Figure 1: Decision diagrams for $(x_0, x_1, x_2, x_3) \mapsto x_0 \oplus x_3 \oplus (\neg x_1 \wedge x_2)$.

2.1 Semantics

Let $\mathbb{B}^{n \Rightarrow m}$ denote the set of functors from $\mathbb{B}^{n \to 1}$ to $\mathbb{B}^{m \to 1}$. A functor $\delta \in \mathbb{B}^{n \Rightarrow m}$ is an (unary) operator that transforms a Boolean function of arity n into a Boolean function of arity m.

$$\delta : \operatorname{Dom} \delta \to \operatorname{Im} \delta,$$

where the sets $\operatorname{Dom} \delta$ and $\operatorname{Im} \delta$ will respectively denote the the domain and co-domain (or image set) of the functor δ. As we are concerned with constructing Boolean functions bottom-up from trivial constant functions, thus all functors will be such that $m \geq n$. The functors with $m = n+1$, that is those increasing the arity by exactly 1, will be of particular interest, and will be referred to as *elementary* in the remainder if this article.

This work will focus on sets of functors that are freely generated from a subset of elementary functors. Let Δ denote a set of injective elementary functors that act on Boolean functions. We term such a set a *model*, or a Δ-model in case we want to be more explicit. For instance, the $\Delta_{\mathsf{u},\mathsf{c}_{1\ell}}$-model contains two elementary functors, namely δ_{u} and $\delta_{\mathsf{c}_{1\ell}}$. Let Δ^* denote the set generated by an arbitrary number of composition of functions in Δ and Δ^+ the same set deprived from the identity functor. In other words, Δ^+ is the free semigroup of functors generated by Δ (with respect to the composition operator) and Δ^* is the monoid generated by Δ.

Definition 2 (Irreducibility)**.** *A Boolean function f is* irreducible *with respect to Δ^+ if and only if f does not belong to $\operatorname{Im} \delta$ for any $\delta \in \Delta^+$. Otherwise, it is called* reducible *(or δ-reducible if we want to emphasize the operator).*

For instance, the identity function $\imath : 0 \star 1$ is irreducible with respect to $\Delta^+_{\mathsf{u},\mathsf{c}_{1\ell}}$ since the only involved variable is neither useless nor $\mathsf{c}_{1\ell}$-canalizing, whereas the function $g \star 0$, for any Boolean function g, is $\delta_{\mathsf{c}_{1\ell}}$-reducible as it can be reduced to $\delta_{\mathsf{c}_{1\ell}} g$.

To formalize this reduction process, we construct the set Δ^- of converse relations (or transposes) of the elements in Δ, that is for each $\delta \in \Delta$, we populate Δ^- with an element δ^- defined partially over Boolean functions as follows:

$$\begin{aligned} \delta^- : \operatorname{Im} \delta &\to \operatorname{Dom} \delta \\ \delta g &\mapsto g \end{aligned}$$

Although δ^- looks very similar to an inverse element, it is not. Indeed, the composition $\delta \circ \delta^-$ differs from the identity since the latter is total, whereas the former is only defined over a subset of Boolean functions (namely $\operatorname{Im}\delta$). One can naturally extend the converse relation to arbitrary elements in Δ^+ by induction over their lengths:

$$\forall \delta_1, \delta_2 \in \Delta^+, \quad (\delta_1 \delta_2)^- := \delta_2^- \delta_1^- \ .$$

Since the set of functions of a given arity is finite, the membership $f \in \operatorname{Im}\delta$ for any function f and $\delta \in \Delta^+$ is decidable.

Therefore, one can effectively perform successive reductions

$$f \xrightarrow{\delta_1^-} f_1 \xrightarrow{\delta_2^-} \cdots \xrightarrow{\delta_k^-} f_k \quad \text{(where } f_k \text{ is irreducible)}.$$

Termination of the above process is ensured as the arity decreases during successive iterations and, by definition, constant functions of arity zero are irreducible for any Δ^+. However, depending on the model, there may exist functions with several distinct reductions, e.g., $0 \star 1 = \delta_{\mathsf{x}} 0 = \delta_{\mathsf{c}_{00}} 1 = \delta_{\mathsf{c}_{11}} 0$.

Besides, if f is non uniquely reducible, then δf is in turn necessarily non uniquely reducible.

Hence, in order to later define a well-founded canonical structure to represent Boolean functions, we motivate the following definition as a way of isolating those functions with more than one reduction.

Definition 3 (Leaf Function)**.** *Given a set Δ of unary operators, the set of* leaf functions*, L_Δ, is defined inductively as follows:*

- *the two constant functions of arity zero are (trivial) leaf functions*
- *non uniquely reducible functions are leaf functions*
- *$\delta\ell$ is a leaf function for any leaf function ℓ and any $\delta \in \Delta^+$*

The sets of irreducible and leaf functions are incomparable (as sets), they are not disjoint, and one is not included in the other. A leaf function can be either reducible or irreducible, and an irreducible function is not necessarily a leaf. We will not attempt to characterize such functions in general, but we shall see that in many relevant models, the only irreducible leaves are constant Boolean functions of arity 0, hence, the set of leafs L_Δ boils down to $\Delta^*\{0, 1\}$.[1] Also notice that in our framework, models where all Boolean functions are leaf functions have no interest since we are not able extract any useful structure from Boolean functions. The uniqueness of reduction stated below follows from the very definition of leaf functions.

Proposition 1 (Reduction uniqueness)**.** *Let Δ denote a set of elementary functors. A Boolean function f is either a leaf function or it can be uniquely reduced to $\delta\ r$, where $\delta \in \Delta^*$, and r is an irreducible function that is not a leaf.*

The idea is then to reduce a (non-leaf) function to an irreducible function that is necessarily not a leaf, and then perform the reduction on the operands of the so obtained irreducible function all the way until leaf functions are reached.

The whole purpose of the following sections is to introduce and explain the elements necessary to perform this reduction process syntactically. We will start by introducing a new data structure, akin to BDD, that matches the semantics of Boolean functions we have just seen.

[1] One may understand the fact that $L_\Delta = \Delta^*\{0, 1\}$, as an indication that Δ is expressive enough to contain its own corner cases, thus not relying on Shannon operator to deal with them. In practice, it allows to have simpler normalizing rules, hence, simplifying both the proof of canonicity and the implementation.

2.2 Ordered Functional Decision Diagrams (λDD-O)

We are given a finite set of letters Δ. We purposely overload the symbol Δ as the semantics of those letters will be the elementary unary operators introduced in Section 2.1. A n-ary λDD is a dependent inductive structure defined as follows:

$$(\phi, n) ::= (\ell, n) \mid \delta_{(n-k)}\ (\phi, k) \mid (\phi, n-1) \star (\phi, n-1),$$

where (ℓ, n) is called a *leaf* of arity n, $\star$ is a binary operator in an infix form, and $\delta_{(n-k)}$ is a *word* of length $n - k \geq 0$ made of letters from Δ. The star operator is only defined for operands with the same arity $n-1$ and returns a λDD of arity n.

An n-ary λDD can be represented as a directed acyclic graph constructed inductively. Nodes are of two types: on the one hand, a set of terminal nodes, and on the other hand a unique star node with two distinct child nodes (left and right). A fundamental difference with BDD is that the star nodes in λDD are not statically labeled. All edges are labeled with (possibly empty) words from Δ^*.

Definition 4 (λDD-O)**.** *An n-ary λDD is a graph constructed inductively* bottom-up *as follows:*

(ℓ, n) *an edge labelled with a word of length n, pointing to a terminal node*

(ϕ, n) *an edge labelled with a word $\delta_{(n-k)}$ of length $n-k$ pointing to an λDD of arity k; or a graph formed by a star node with two $(n-1)$-ary λDD child nodes.*

The composition of two words $\delta_{(k)}\delta_{(m)}$ is simply their concatenation. A λDD-O graph can be given, by induction on the graph structure, a semantics over Boolean functions.

- Terminal nodes with an edge labelled by an empty word are constant functions of arity zero
- Terminal nodes with an edge labelled by a word of length n are leaf functions of arity n (Definition 3)
- Star nodes encode the Shannon operator $\star$ over Boolean functions of the same arity
- A word encodes a functor in Δ^+ (see Section 2.1). The empty word corresponds to the identity functor.

For the sake of clarity, we will showcase the rules for a particular ordered model while running them on a concrete example. The construction can be readily generalized to other models as well. Consider the alphabet $\Delta = \{\mathtt{u}, \mathtt{c}_{10}\}$. Where $\mathtt{u}$ corresponds to the functor $\delta_{\mathtt{u}}$ encoding useless variables, and $\mathtt{c}_{10}$ encodes the functor $\delta_{\mathtt{c}_{10}}$ adding a particular canalizing variable.

We will use the standard semantics bracket $[\![(\phi, n)]\!]$ to denote the Boolean function that graph (ϕ, n) represents. Dually, each Boolean function f of arity n can be represented by an λDD (ϕ, n), such that $[\![(\phi, n)]\!] = f$.

This fact follows from the fact that any Boolean function can be encoded as an SDD that can be mapped to an instance of λDD-S (the λDD model of SDD, which is induced by the empty set of functor), which is also an instance of any λDD model (as for any set Δ of functors, $\emptyset \subseteq \Delta$). Thus, any Boolean function can be encoded as instance of any λDD model.

We next introduce the necessary reduction rules to go from an SDD to a (ϕ, n) with respect to a given Δ. For the ordered models, only the introduction and normalization rules are required. The two introduction rules for the two considered elementary functors are as follows:

$$\texttt{intro-o-u}\frac{(\phi, n) \star (\phi, n)}{\mathtt{u}\ (\phi, n)} \qquad \texttt{intro-o-c}_{10}\frac{(\phi, n) \star (0, n)}{\mathtt{c}_{10}\ (\phi, n)}$$

The resulting model is called λDD-O-UC$_{10}$ in the remainder of this article.

2.3 Normalizing Output Negation

Elementary operators, and therefore the letters, are not allowed to commute in ordered models as this would mean changing the prefixed variable ordering. Nevertheless, one could be interested in enriching the model with an arity-preserving functor, namely, the negation. The interplay between the negation and the elementary functors in ordered models then has to be studied.

Negation is introduced by normalizing the representation of the constant functions *1* which can be either represented by $(1,0)$ or $\neg(0,0)$. The rule removes the former to explicitly expose negation. The first (normalization) rule exploits the fact that $\neg$ is an involution, that is $\neg\circ\neg = \imath$:

$$\texttt{norm-n0}1\,\frac{(1,n)}{\neg\ (0,n)} \qquad \texttt{norm-nn}\,\frac{\neg\ (\neg\ (\phi,k))}{(\phi,k)}$$

We can easily prove that $\neg$ distributes over $\star$ leading to the following rule where, by convention, all annotations by $\neg$ are normalized on the right-hand side of $\star$ (similar to other BDD variants normalizing negated edges).

$$\texttt{norm-nsha}\,\frac{(\neg\ (\phi,k))\star(\phi,k)}{\neg\ ((\phi,k)\star(\neg\ (\phi,k)))}$$

Negation does not necessarily commute with all elementary functors, as shown in the following counter-example

$$\begin{array}{ll}(\delta_{\mathsf{c}_{1\mathit{0}}}\circ\neg)\mathit{0} = \mathit{1}\star\mathit{0} = \neg\imath & (x\mapsto\neg x)\\ (\neg\circ\delta_{\mathsf{c}_{1\mathit{0}}})\mathit{0} = \neg(\mathit{0}\star\mathit{0}) = \mathit{1} & (x\mapsto 1)\end{array}$$

However it quasi-commutes, that is, for each elementary functor δ, there exists an elementary functor δ' such that $\delta\circ\neg = \neg\circ\delta'$.

The following normalization rule exploits quasi-commutativity to send negation upward:

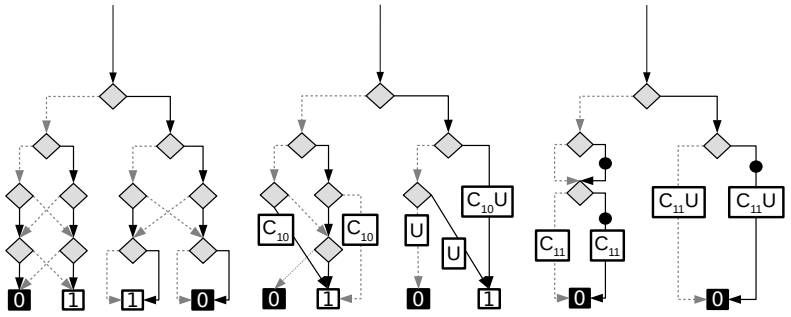
$$\texttt{norm-ncom}\,\frac{\delta_{\mathsf{e}}\circ\neg\ (\phi,k)}{\neg\circ\overline{\delta_{\mathsf{e}}}\ (\phi,k)}\ .$$

Where $\delta_{\mathsf{e}}\in\Delta$ and $\overline{\delta_{\mathsf{e}}}$ is defined by $\overline{\mathsf{u}} := \mathsf{u}$, $\overline{\mathsf{x}} := \mathsf{x}$, and, $\overline{\mathsf{c}_{b\mathsf{t}}} := \mathsf{c}_{b(\neg\mathsf{t})}$. One can formally define $\overline{\delta}$ as $\neg\delta\neg$, then show $\overline{\Delta}\subseteq\Delta$. This fact explains the difficulty for adding (and normalizing) negation in some BDD variants like ZDD. Indeed, in order to have the stability of the representation under output negation, one must have a similar property on the functor of the considered model.

To appreciate and contrast the λDD diagrams with respect to some known variants, we give in Figure 2 the λDD diagrams of the same running example, where the edges are annotated with words. The model resulting from the of functors $\{\neg\}\bigcup\{\delta_{\mathsf{u}}\}\bigcup\{\delta_{\mathsf{c}_{b\mathsf{t}}}\}_{b,\mathsf{t}\in\mathbb{B}}$, is called λDD-O-NUC in the sequel. Observe that nodes are no longer labeled with the name of the variable on which the decision is made, since this information can be retrieved from the arity of the functors and the nesting (or depth) of Shannon nodes.

3 Uniform Models

As mentioned in the previous section, factoring out (or distributing) unary operators over the generic star operator, or equivalently commuting the unary operators (whenever possible), is not allowed in the ordered models by design. This would indeed, violate the global fixed ordering over the variables. This section exploits the functional abstraction to reduce further the size of λDD precisely by changing the variable ordering in a specific way guided by the commutativity properties of the underlying elementary functors. The so obtained models will be referred as *uniform* for reasons that will become clear shortly.



(a) λDD-S (b) λDD-O-UC$_{1\partial}$ (c) λDD-O-NUC

Figure 2: Ordered λDD for $(x_0, x_1, x_2, x_3) \mapsto x_1 \oplus x_2 \oplus (\neg x_0 \wedge x_3)$.

3.1 Semantics

Swapping the star operators is always possible provided some side conditions on variables.

Proposition 2 (Swapping)**.** *For all Boolean functions f_1, f_2, g_1, g_2 with the appropriate arities,*

$$(f_1 \star_x f_2) \star_y (g_1 \star_x g_2) = (f_1 \star_y g_1) \star_x (f_2 \star_y g_2),$$

where we assume that "x" and "y" are fresh variable names not occurring in any operand.

Allowing factorization of unary operators has a potentially huge impact on the size of graph-based data structures. Indeed, avoiding or delaying branching can be formally captured as factoring out common divisors of the two operands of the star. Consider the following expressions (their usual functional notations are given on the right for clarity)

$$\begin{aligned} f = &\quad \delta_{\mathsf{c}_{1\partial},x_1}\delta_{\mathsf{u},x_0}\,1 && (f : (x_1, x_0) \mapsto \neg x_1 \wedge 1) \\ g = &\quad \delta_{\mathsf{c}_{1\partial},x_1}\delta_{\mathsf{x},x_0}\,1 && (g : (x_1, x_0) \mapsto \neg x_1 \wedge (x_0 \oplus 1)) \end{aligned}$$

In the ordered model, with respect to ordering $\{x_0, x_1, y\}$, $f \star_y g$ would remain as is. If, however, we factor out $\delta_{\mathsf{c}_{1\partial},x_1}$, we get

$$f \star_y g = (\delta_{\mathsf{c}_{1\partial},x_1}\delta_{\mathsf{u},x_0}\,1) \star_y (\delta_{\mathsf{c}_{1\partial},x_1}\delta_{\mathsf{x},x_0}\,1) = \delta_{\mathsf{c}_{1\partial},x_1}(\delta_{\mathsf{u},x_0}\,1 \star_y \delta_{\mathsf{x},x_0}\,1)$$

obtaining an expression with respect to the ordering $\{x_0, y, x_1\}$. More generally, if the functions f, g, defined over $(x_0, \ldots, x_{n-1})$ share a common divisor $\delta \in \Delta^+$ that introduces the subset of variables $\{x_{i_1}, \ldots, x_{i_s}\}$, then

$$f \star_y g = (\delta_{x_{i_1},\ldots,x_{i_s}} f') \star_y (\delta_{x_{i_1},\ldots,x_{i_s}} g') = \delta_{x_{i_1},\ldots,x_{i_s}}(f' \star_y g') \ .$$

When transposed into a graph-based representation, such factorization saves at most s branching, and therefore 2^s nodes by introducing y first (via the star operator), then $\{x_{i_1}, \ldots, x_{i_s}\}$ via δ. The non-existence of common factors motivates the following definition.

Definition 5 (Coprime Functions)**.** *Two functions f and g are said to be* Δ-coprime *(or simply coprime whenever Δ is clear from the context) if and only if there is no $\delta \in \Delta^+$ such that $f, g \in \operatorname{Im}\delta$.*

Coprimality allows us in turn to formalize the intuitive idea of extracting all functors common to two given Boolean functions.

Definition 6 (Greatest Common Divisor)**.** *Let f and g be two Boolean functions. The functor δ is a* greatest common divisor*, or* gcd*, for f and g if and only if there exist two coprime Boolean functions f' and g' such that $f = \delta f'$ and $g = \delta g'$.*

Accordingly, when two functions are coprime, their unique gcd is the identity functor. The uniqueness of the gcd also depends on the concept of leaf functions (see Definition 3) and can be stated very much like we stated reduction uniqueness (Proposition 1). Functor δ in Definition 6 is not necessarily elementary. It should be considered as a whole regardless of its possible underlying representations into elementary functors. For instance, introducing two fresh xor-canalizing variables x and y will lead to the same function regardless of which variable is introduced first. Obviously, not all elementary functors commute as shown in the following counter-example:

$$\begin{array}{ll} \delta_{\mathsf{c}_{00}}\delta_{\mathsf{c}_{11}}0 & ((x_1,x_2)\mapsto x_1\wedge x_2) \\ \delta_{\mathsf{c}_{11}}\delta_{\mathsf{c}_{00}}0 & ((x_1,x_2)\mapsto x_1) \end{array}$$

In addition, arity-preserving functors, like negation, do not necessarily commute with elementary functors. One might therefore consider a weaker notion, like *quasi-commutativity*, where two elementary functors are required. For instance, the functors $\mathsf{c}_{10}, \mathsf{c}_{11}$ quasi commute with the negation since for any function f the following holds:

$$(\neg \circ \delta_{\mathsf{c}_{10}})\ f = (\delta_{\mathsf{c}_{11}} \circ \neg)\ f\ .$$

The importance of those considerations will become clearer when one attempts to normalize the functors in order to extract common factors (see Section 3.3)

The concepts of coprimality and common factors suggest a reduction process for Boolean functions with respect to a given set of unary operators Δ. Let f denote a Boolean function of arity $n > 0$. The process starts by decomposing it into a functor and an irreducible function r (of arity less than n). Since r is irreducible, it is either is a constant function of arity zero (in which case the reduction process ends), or it is decomposable into two coprime Boolean functions (using the star operator). By iterating this process over each operand, one gets a unique reduction of f with respect to Δ all the way down to leaf functions.

Theorem 1 (General Reduction)**.** *Let Δ be a set of elementary functors. Assuming canonical representations for Δ^+ and the set of leaf functions, every Boolean function f can be uniquely reduced to leaf functions using the functors in Δ^+ and the star operator (applied to coprime functions).*

Recall that leaf functions were defined precisely to avoid dealing with non-uniqueness corner cases and to postpone such issues to finding a canonical representation for leaf functions and their interactions with other non-leaf functions. This is the topic of Section 3.3 in which we detail a canonical representation for the set of functors and leaf functions for a rich uniform model containing all canalizing variables (see Table 1) together with the negation operator.

Relative Variable Naming

In practice, we have to to guarantee the normalization of variable names, while avoiding name collision. To avoid checking cumbersome side conditions requiring that the newly introduced names should be distinct from those already introduced in the operands, it is much more convenient, from a functional point of view, to internalize this condition in the very definition of the star operator (see Definition 1):

Definition 7 ($\star_i$)**.** *Let* $i \in \mathbb{N}$.

$$\begin{aligned} \star_i : \mathbb{B}^{n\to 1} \times \mathbb{B}^{n\to 1} &\to \mathbb{B}^{n+1\to 1} \\ (f, g) &\mapsto f \star_i g \end{aligned}$$

where $f \star_i g$ *is only defined when* $0 \leq i \leq n$ *by:*

$$\begin{aligned} f \star_i g : \mathbb{B}^{n+1} &\to \mathbb{B} \\ (v_0, \ldots, v_n) &\mapsto \begin{cases} f(v_0, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n) & \text{if } v_i = 0 \\ g(v_0, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n) & \text{if } v_i = 1 \end{cases} \end{aligned}$$

This reformulation leads to introduce relative variable naming. For sake of clarity, we will keep using absolute variable names later on as index manipulations in relative naming is error-prone for the human reader.

Note that $\star_0$ is exactly the $\star$ operator of definition 1.

Swapping the star operators can now be defined as follows.

Proposition 3 (Swapping)**.** *Let* $f_1, f_2, g_1, g_2 \in \mathbb{B}^{n\to 1}$.
If $0 \leq i < j \leq n$ *then*

$$(f_1 \star_i f_2) \star_j (g_1 \star_i g_2) = (f_1 \star_{j-1} g_1) \star_i (f_2 \star_{j-1} g_2) \ .$$

Otherwise, $0 \leq j \leq i \leq n$ *and*

$$(f_1 \star_i f_2) \star_j (g_1 \star_i g_2) = (f_1 \star_j g_1) \star_{i+1} (f_2 \star_j g_2) \ .$$

One may notice that because of relative naming, this property holds even if $i = j$ as it is now possible to insert twice a variable at position zero, as in a more absolute naming, we actually introduced variables at position zero and one. Similarly, the unary operators we have introduced so far for useless and canalizing variables can now be extended with relative naming, for instance:

$$\begin{aligned} \delta_{\mathsf{u},i} &: f \mapsto f \star_i f && \text{(inserting a useless variable at the } i\text{th position)} \\ \delta_{\mathsf{x},i} &: f \mapsto f \star_i \neg f && \text{(inserting a xor-variable at the } i\text{th position)} \end{aligned}$$

Factoring out unary operators and/or commuting them (whenever allowed) follows naturally from Proposition 3, for example, if $i < j$ then:

$$(\delta_{\mathsf{u},i} f) \star_j (\delta_{\mathsf{u},i} g) = \delta_{\mathsf{u},i}(f \star_{j-1} g)$$

$$\delta_{\mathsf{x},j}(\delta_{\mathsf{u},i} f) = (\delta_{\mathsf{u},i} f) \star_j (\delta_{\mathsf{u},i} \neg f) = \delta_{\mathsf{u},i}(f \star_{j-1} \neg f) = \delta_{\mathsf{u},i}(\delta_{\mathsf{x},j-1} f)$$

The next section introduces the λDD data structure that will then be instantiated to a concrete model. As mentioned above, in the remainder of this article we will keep using absolute naming for sake of clarity. However, one must keep in mind, that all our statements can be reformulated using variable naming, modulo the introduction of case distinction.

3.2 Uniform Functional Decision Diagrams (λDD-U)

In the uniform models, the variable at position zero does not play a particular role, in fact all variables are treated equally, or *uniformly* (hence the name). In particular, one is allowed to shift variables in order to leverage more reductions. For instance, in the ordered model, $\mathsf{u}\ ((0,0) \star (1,0))$ (corresponding to the function $(y, x) \mapsto x$) and $(\mathsf{u}\ (0,0)) \star (\mathsf{u}\ (1,0))$ (corresponding

to $(y, x) \mapsto y$) are two distinct reduced representations although both essentially represent a projection. The uniform model can capture and leverage such considerations. Compared to the ordered model, the alphabet is now enriched with the letter 's' for *support* (or Shannon) variables, that is, those variables generically typed and introduced by the Shannon operator with no corresponding unary functor. This new letter 's' will essentially act as a placeholder for the already introduced support variables.

The semantics of words correspond to unary functors. For instance, the word 'ussu' corresponds to adding a useless variable at positions 0 and 3 [2] which amounts to applying the functor $\delta_{\mathsf{u},3}\delta_{\mathsf{u},0}$ For the projections, the representations are as follows

$$\begin{aligned} &\mathsf{us}\ ((0,0) \star (1,0)) && ((y,x) \mapsto x) \\ &\mathsf{su}\ ((0,0) \star (1,0)) && ((y,x) \mapsto y) \end{aligned}$$

where the word 'us' corresponds to $\delta_{\mathsf{u},0}$ and the word 'su' to $\delta_{\mathsf{u},1}$

We will denote words by overloading the symbol δ used for their semantics interpretation. The composition of two words $\delta_1 \circ \delta_2$ is defined whenever the length of the right word δ_2 matches the number of letters 's' in the left word. The composition is computed by substituting the 's' positions in δ_1 by the elements of δ_2. For example:

$$(a\mathsf{ss}b\mathsf{s})(\alpha_1\alpha_2\alpha_3) = (a\alpha_1\alpha_2 b\alpha_3),$$

where 'a' and 'b' denote two letters from alphabet Δ distinct from 's' (α_i could be any letters from the same alphabet).

The introduction and factorization rules are sketched below for xor-canalizing variables. They could be readily generalized to useless variables and other canalizing variables (see Appendix A for an exhaustive list of all rules). The introduction rule is very similar to the one we have already seen in the ordered model, except for using as many placeholders 's' as necessary for encoding the already introduced variables:

$$\texttt{intro-u-x}\frac{(\phi, k) \star \neg(\phi, k)}{\mathsf{xs}^k\ (\phi, k)}$$

The word 'xs^k' corresponds exactly to $\delta_{\mathsf{x},0}$ which inserts a new xor-canalizing variable at position 0. The generic case for inserting a xor-canalizing variable at the ith position can be done similarly by padding with the 's' letter on both the left and right sides to account for the already introduced variables, as shown below for the generic factorization rule (again for xor-canalizing variables):

$$\texttt{facto-u-x}\frac{\mathsf{x}_{i,k}\ (\phi, k) \star \mathsf{x}_{i,k}\ (\phi', k)}{\mathsf{x}_{i+1,k+1}\ ((\phi, k)) \star (\phi', k))}$$

where, for any $0 \leq i < k$, $\mathsf{x}_{i,k} := \mathsf{s}^i\mathsf{x}\mathsf{s}^{k-i}$ is the word of length $k+1$, composed of k 's' and one 'x' at the ith position (counting from the left and starting with index zero).

Proposition 4 (Correctness of `facto-u-x`)**.** *For any Boolean functions $[\![(\phi, k)]\!]$ and $[\![(\phi', k)]\!]$ of arity n, the Boolean functions of the premise and conclusion of the reduction rule* `facto-u-x` *are semantically equivalent, that is:*

$$[\![\mathsf{x}_{i,k}\ (\phi, k)) \star \mathsf{x}_{i,k}\ (\phi', k))]\!] = [\![\mathsf{x}_{i+1,k+1}\ ((\phi, k)) \star (\phi', k))]\!],$$

[2] Using relative naming, one may either introduce a useless variable, first at position 0 then at position 3, or first at position 2, then at position 0, both ways leading to the same results.

Similar introduction and factorization rules can be defined for the four other canalizing variables using "letters" $\mathsf{c}_{00}, \mathsf{c}_{10}$ and $\mathsf{c}_{01}, \mathsf{c}_{11}$ (cf. Appendix A).

The last class of reduction rules concerns normalization rules, for both words (as a syntactic representation of functors) and terminal nodes (as a syntactic representation of leaf functions). As already stated, all words compose but do not commute in general. This requires isolating those functors that commute from those which do not.

Apart from elementary functors, the arity-preserving negation operator either commutes or quasi-commutes with the elementary functors. For instance, if one wants to normalize the use of the negation with the canalizing variables, one has to account for the two letters $\mathsf{c}_{b\mathsf{t}}$ and $\mathsf{c}_{b(\neg\mathsf{t})}$ and use the following equivalence:

$$\neg \circ \left(\mathsf{s}^j \mathsf{c}_{b\mathsf{t}} \mathsf{s}^i\right) = \left(\mathsf{s}^j \mathsf{c}_{b(\neg\mathsf{t})} \mathsf{s}^i\right) \circ \neg \ .$$

Before detailing our canonical representation for leaf functions (Section 3.3.2), we want to give an intuition about why several representations are possible for the exact same function. When $(\phi, k) = (\mathsf{t}, k)$ with $\mathsf{t} \in \{0, 1\}$, at least two introduction rules apply because of the following equivalences:

$$\mathsf{c}_{b\mathsf{t}}\mathsf{u}^k\ (\mathsf{t}, k) = \mathsf{u}^{k+1}\ (\mathsf{t}, k) \tag{2}$$

$$\mathsf{c}_{b\mathsf{t}}\mathsf{u}^k\ (\neg\mathsf{t}, k) = \mathsf{c}_{(\neg b)(\neg\mathsf{t})}\mathsf{u}^k\ (\mathsf{t}, k) \tag{3}$$

One thus has to decide whether the new variable is useless or canalizing. When such leaf functions are considered separately, we use the following normalization rules, stating that a variable shall be canalizing only if it is not useless.

$$\texttt{norm-u-cbtt}\frac{\mathsf{c}_{b\mathsf{t}}\mathsf{u}^k\ (\mathsf{t}, k)}{\mathsf{u}^{k+1}\ (\mathsf{t}, k)} \ .$$

Lastly, when at least one operand of the star operator is a leaf function, fixing one particular canonical representation may be inconvenient as they could potentially hide common factors. We present next how we solve these issues.

3.3 Syntactic Normalization

In this section, we detail the normalization rules for all models combining useless and canalizing variables together with the (output) negation. We term this model λ`DD-U-NUCX`. We will manipulate words built using the following four sets of letters

$$\Delta_{\mathsf{u}} := \{\mathsf{u}\}, \quad \Delta_{\mathsf{x}} := \{\mathsf{x}\}, \quad \Delta_{\mathsf{c}_0} := \{\mathsf{c}_{0\mathit{0}}, \mathsf{c}_{1\mathit{0}}\}, \quad \Delta_{\mathsf{c}_1} := \{\mathsf{c}_{0\mathit{1}}, \mathsf{c}_{1\mathit{1}}\}$$

in addition to the letter 's' that will be used as a placeholder, as explained in the previous section. The letters in each set correspond to their respective elementary functors, namely $\delta_{\mathsf{u},i}$, $\delta_{\mathsf{x},i}$, $\{\delta_{\mathsf{c}_{0\mathit{0}},i}, \delta_{\mathsf{c}_{1\mathit{0}},i}\}$, and $\{\delta_{\mathsf{c}_{0\mathit{1}},i}, \delta_{\mathsf{c}_{1\mathit{1}},i}\}$. This separation is motivated by the following facts: (1) the elementary functors in each set alone commute or quasi-commute among themselves, (2) they also either commute (for Δ_{u} and Δ_{x}) or quasi-commute (for Δ_{c_0} and Δ_{c_1}) with the negation, and (3) the elementary functors (distinct from δ_{u}) across sets do not commute.

3.3.1 Normalizing Words

We remind the reader, that we use absolute naming of variables for convenience. Consider the following functor that takes a Boolean function of arity 2 (having, say, the names x_2 and x_4 for its inputs) and returns a Boolean function of arity 7:

$$\delta_1 := \delta_{\mathsf{x},x_3}\delta_{\mathsf{x},x_2}\delta_{\mathsf{c}_{0\mathit{0}},x_0}\delta_{\mathsf{c}_{1\mathit{0}},x_6}\delta_{\mathsf{c}_{0\mathit{1}},x_5}$$

The functor inserts 5 canalizing variables of different types. It can be represented as:

$$\delta_1 = [\![(\mathsf{ssxxsss})(\mathsf{c}_{0\mathit{0}}\mathsf{sssc}_{1\mathit{0}})(\mathsf{ssc}_{0\mathit{1}})]\!]$$

where the action of the functor is staged into three *homogeneous* functors with respect to the sets we described earlier: first a functor ($\mathsf{ssc}_{0\mathit{1}}$) in $\Delta^+_{\mathsf{c}_{\mathit{1}}}$, then a functor ($\mathsf{c}_{0\mathit{0}}\mathsf{sssc}_{1\mathit{0}}$) in $\Delta^+_{\mathsf{c}_{\mathit{0}}}$, and finally a functor ($\mathsf{ssxxsss}$) in Δ^+_{x}. To emphasize the staging aspect of words in the uniform models, we will represent them in a tableau. The word corresponding to δ_1 is represented as follows:

$$\delta_1 := [\![\begin{pmatrix} \mathsf{c}_{0\mathit{0}} & & \mathsf{x}\ \mathsf{x} & \\ & \mathsf{s} & & \mathsf{s}\ \mathsf{c}_{0\mathit{1}}\ \mathsf{c}_{1\mathit{0}} \end{pmatrix}]\!]$$

having three rows (one for each homogeneous component) and seven columns (one for each variable). The letter 's' is omitted from all rows except the last one for clarity. The word $\delta_2 := \delta_{\mathsf{u},x_8}\delta_{\mathsf{u},x_7}\delta_1$ is represented by adding a row at the bottom of the tableau having 'u' in the 7th and 8th columns. Since $\delta_{\mathsf{u},i}$ commutes with all other elementary functors defined so far, moving the letter 'u' upward (or downward) does not change the semantics of the word. The first normalization rule will therefore consist in propagating the letters 'u' upward all the way up to the first row. Notice that a row with only 's' letters is semantically equivalent to the identity. Thus, such rows are removed from the tableau. We therefore obtain the following tableau for δ_2:

$$\delta_2 := [\![\begin{pmatrix} \mathsf{c}_{0\mathit{0}} & & \mathsf{x}\ \mathsf{x} & & \mathsf{u}\ \mathsf{u} \\ & \mathsf{s} & & \mathsf{s}\ \mathsf{c}_{0\mathit{1}}\ \mathsf{c}_{1\mathit{0}} & \end{pmatrix}]\!]$$

Following the same reasoning, adjacent rows containing commuting elementary functors will also be moved up till reaching a row with a non-commuting functor. For example, consider $\delta_3 := \delta_1(\mathsf{uc}_{1\mathit{1}})$, first we add a row that corresponds to the homogeneous word '$\mathsf{uc}_{1\mathit{1}}$':

$$\begin{pmatrix} \mathsf{c}_{0\mathit{0}} & & \mathsf{x}\ \mathsf{x} & \\ & \mathsf{s} & & \mathsf{s}\ \mathsf{c}_{0\mathit{1}}\ \mathsf{c}_{1\mathit{0}} \end{pmatrix} \begin{pmatrix} \mathsf{uc}_{1\mathit{1}} \end{pmatrix} \longrightarrow \begin{pmatrix} \mathsf{c}_{0\mathit{0}} & & \mathsf{x}\ \mathsf{x} & & & \mathsf{c}_{1\mathit{0}} \\ & (\mathsf{s}) & & (\mathsf{s}) & \mathsf{c}_{0\mathit{1}} & \\ & \mathsf{u} & & \mathsf{c}_{1\mathit{1}} & & \end{pmatrix}$$

Then, we move the useless variable all the way up to the first row

$$\begin{pmatrix} \mathsf{c}_{0\mathit{0}} & & \mathsf{x}\ \mathsf{x} & & \mathsf{c}_{1\mathit{0}} \\ & \mathsf{u} & & \mathsf{c}_{0\mathit{1}} & \\ & & \mathsf{c}_{1\mathit{1}} & & \end{pmatrix} \longrightarrow \begin{pmatrix} \mathsf{c}_{0\mathit{0}} & \mathsf{u}\ \mathsf{x}\ \mathsf{x} & & & \mathsf{c}_{1\mathit{0}} \\ & & & \mathsf{c}_{0\mathit{1}} & \\ & & \mathsf{c}_{1\mathit{1}} & & \end{pmatrix}$$

Next, we notice that $\mathsf{c}_{0\mathit{1}}$ and $\mathsf{c}_{1\mathit{1}}$ both belong to $\Delta_{\mathsf{c}_{\mathit{1}}}$ thus they commute and we can merge the last two rows together:

$$\begin{pmatrix} \mathsf{c}_{0\mathit{0}} & \mathsf{u}\ \mathsf{x}\ \mathsf{x} & & & \mathsf{c}_{1\mathit{0}} \\ & & & \mathsf{c}_{0\mathit{1}} & \\ & & \mathsf{c}_{1\mathit{1}} & & \end{pmatrix} \longrightarrow \begin{pmatrix} \mathsf{c}_{0\mathit{0}} & \mathsf{u}\ \mathsf{x}\ \mathsf{x} & & & \mathsf{c}_{1\mathit{0}} \\ & & & \mathsf{c}_{1\mathit{1}}\ \mathsf{c}_{0\mathit{1}} & \end{pmatrix}$$

In general, composition of words $\delta_1 \circ \delta_2$ corresponds to a special concatenation operation on tableaus and is valid if and only if the number of 's' in the last row of δ_1 matches the number of columns in δ_2: the columns of δ_2 are spread right under the 's' positions in the last row of δ_1, the rest is always padded with the s letters.

We move on to explain how the negation is encoded in our representation and detail its interplay with the other functors. Just like useless variables were propagated upward to avoid branching as much as possible and to expose information on the structure early one, we wish to move the negation upward in the representation to allow negation to be performed in constant time, which in turn permits to efficiently check for the potential introduction of xor-canalizing variables (recall that δ_{x} or $\delta_{\mathsf{x},i}$ are defined using the negation).

We add a Boolean supscript to the upper-left corner of the tableau to explicit the presence of the negation at the beginning of the word: 1 represents the presence of a negation, 0 its absence. For instance:

$$\neg \circ \delta_1 = \neg \; {}^{0}\begin{pmatrix} \mathsf{c}_{0\emptyset} & \times & \times \\ \mathsf{s} & & \mathsf{s} \\ & \mathsf{c}_{01} & \mathsf{c}_{1\emptyset} \end{pmatrix} \longrightarrow {}^{1}\begin{pmatrix} \mathsf{c}_{0\emptyset} & \times & \times \\ \mathsf{s} & & \mathsf{s} \\ & \mathsf{c}_{01} & \mathsf{c}_{1\emptyset} \end{pmatrix}$$

With respect to this convention, left-hand composition of our representation with the negation is immediate: it flips the supscript bit. Right-hand composition is more involved as the negation has to travel through the tableau. Given a tableau δ, we denote $\overline{\delta}$ the tableau where all symbols $\mathsf{c}_{b\mathsf{t}}$ have been replaced with $\mathsf{c}_{b(\neg\mathsf{t})}$, while x and the implicit s are left unchanged. These considerations reflect the fact that the negation commutes with δ_{u} and δ_{x} and quasi-commutes with the other canalizing variables. Therefore, the normalization of the right-hand composition with negation will be defined as

$$\delta \circ \neg = \neg \circ \overline{\delta} \ .$$

For example:

$$\delta_1 \circ \neg = {}^{0}\begin{pmatrix} \mathsf{c}_{0\emptyset} & \times & \times \\ \mathsf{s} & & \mathsf{s} \\ & \mathsf{c}_{01} & \mathsf{c}_{1\emptyset} \end{pmatrix} \circ \neg \longrightarrow {}^{1}\overline{\begin{pmatrix} \mathsf{c}_{0\emptyset} & \times & \times \\ \mathsf{s} & & \mathsf{s} \\ & \mathsf{c}_{01} & \mathsf{c}_{1\emptyset} \end{pmatrix}} = {}^{1}\begin{pmatrix} \mathsf{c}_{01} & \times & \times \\ \mathsf{s} & & \mathsf{s} \\ & \mathsf{c}_{0\emptyset} & \mathsf{c}_{11} \end{pmatrix}$$

We finally need to adapt our definition of the concatenation over tableaus to account for the negation encoding. We extend the xor operator $\oplus$ as follow:

$$b \oplus \delta := \begin{cases} \delta & \text{if } b = 0 \\ \overline{\delta} & \text{if } b = 1 \end{cases}$$

The composition of ${}^{b_1}(\,\delta_1\,)$ with ${}^{b_2}(\,\delta_2\,)$ is given by:

$${}^{b_1}(\,\delta_1\,)\,{}^{b_2}(\,\delta_2\,) := {}^{b_1 \oplus b_2}(\,(b_2 \oplus \delta_1)\delta_2\,)$$

We exhaustively summarize the normalization rules on words as well as the properties they guarantee.

1. u letters (if any) appear only on the first row. Otherwise, move them upward until they all reach the first row.
2. all rows contain at least one non-s letter. Rows with only s letters are removed.
3. any two consecutive rows belong to distinct alphabets Δ_{x}, $\Delta_{\mathsf{c}_\emptyset}$ or Δ_{c_1}. Otherwise, merge any two consecutive rows which use the same alphabet.

One may show that reduced words are canonical representations for functor in

$$\{\neg\} \cup \{\delta_{\mathsf{u},i}, \delta_{\mathsf{x},i}, \delta_{\mathsf{c}_{0\emptyset},i}, \delta_{\mathsf{c}_{1\emptyset},i}, \delta_{\mathsf{c}_{01},i}, \delta_{\mathsf{c}_{01},i}\}^{+}$$

The above reduction rules can be adapted (in fact simplified) to give a canonical representations for simpler, yet interesting, models (see Table 3 in Section 4). The next section details the normalization rules for leaf functions.

3.3.2 Normalizing Leaf Functions

In addition to the trivial leaf functions, there are exactly four reducible leaf functions (cf. Definition 3) in model λ`DD-U-NUCX`, namely

$0 \star_i 0$	Reducible with $\delta_{\mathsf{u},i}$, or $\delta_{\mathsf{c}_{1\emptyset},i}$, or $\delta_{\mathsf{c}_{0\emptyset},i}$
$1 \star_i 1$	Reducible with $\delta_{\mathsf{u},i}$, or $\delta_{\mathsf{c}_{11},i}$, or $\delta_{\mathsf{c}_{01},i}$
$0 \star_i 1$	Reducible with $\delta_{\mathsf{x},i}$, or $\delta_{\mathsf{c}_{11},i}$, or $\delta_{\mathsf{c}_{0\emptyset},i}$
$1 \star_i 0$	Reducible with $\delta_{\mathsf{x},i}$, or $\delta_{\mathsf{c}_{1\emptyset},i}$, or $\delta_{\mathsf{c}_{01},i}$

where *0* and *1* are constant functions of any arity.[3] We therefore have two types of leaf functions:

1. $\mathsf{t}_{(n)}$: constant functions of arity n ($\mathsf{t} \in \{0, 1\}$), and
2. $\pi_{b,i,n}$: projection functions of arity n on the ith variables (resp. negation of projection) functions of arity n on the i-th variable if the Boolean $b = 0$ (resp. $b = 1$).

We adopt the following representation for leaf functions: ${}^{b}(\delta)\,[T]$, i.e., a pair made of a functor representation ${}^{b}(\delta)$ and a terminal representation $[T]$ where $T \in \{\mathsf{t}_{(n)}, \pi_{b,i,n}\}$. A terminal $[T]$ is said to be reduced if $T \in \{0_{(0)}, \pi_{0,0,1}\}$. Terminal representations are normalized using the two following rules:

- $[\mathsf{t}_{(n)}] \longrightarrow {}^{\mathsf{t}}(\,\mathsf{u}^n\,)\,[0_{(0)}]$
- $[\pi_{b',i,n}] \longrightarrow {}^{b'}(\,\mathsf{u}^i\mathsf{s}\mathsf{u}^{n-i-1}\,)\,[\pi_{0,0,1}]$

Those normalization rules are still not sufficient, as one also has to account for the potential interactions of leaf functions with functors. One has first to fix a preference over functors when applied to the first type of reduced terminals since $\delta_{\mathsf{c}_{b0}} 0 = \delta_{\mathsf{u}} 0$ for any Boolean b. We solve this issue by prioritizing δ_{u}, as this functor commutes with all other functors. The rule applies whenever the last row of the tableau is in Δ_{c_0}. For example:

$$^{1}\left(\begin{array}{ccc}\mathsf{c}_{01} & & \mathsf{x}\,\mathsf{x}\,\mathsf{u}\\ & \mathsf{c}_{10} & \\ & & \mathsf{c}_{00}\end{array}\;\mathsf{c}_{11}\right)[0_{(0)}] \longrightarrow {}^{1}(\,\mathsf{c}_{01}\;\mathsf{u}\,\mathsf{x}\,\mathsf{x}\,\mathsf{u}\,\mathsf{u}\;\mathsf{c}_{11}\,)\,[0_{(0)}]$$

The two letters c_{10} and c_{00} in the last row got transformed into u, then moved up to normalize the functor (as seen in the previous section). The associated generic rule is as follows (where ω denotes a row concatenated to tableau δ):

$${}^{b}(\,\delta\omega\,)\,[0_{(0)}] \quad (\text{with } \omega \in \Delta^{+}_{\mathsf{uc}_0}) \longrightarrow {}^{b}(\,\delta\mathsf{u}^{|\omega|}\,)\,[0_{(0)}]$$

Second, we also need to fix a preference over reduced terminals as they are sometimes interchangeable. For instance, $\delta_{\mathsf{c}_{00},0}\pi_{0,0,1} = \delta_{\mathsf{c}_{00}}\delta_{\mathsf{c}_{00}} 1$, and $\pi_{0,0,1} = \delta_{\mathsf{x}} 0$. We solve these issues by prioritizing constants over projections. For example:

$$^{1}\left(\begin{array}{ccc}\mathsf{c}_{01} & & \mathsf{x}\,\mathsf{x}\,\mathsf{u}\\ & \mathsf{s} & \\ & & \mathsf{c}_{00}\end{array}\;\mathsf{c}_{11}\right)[\pi_{0,0,1}] \longrightarrow {}^{1}\left(\begin{array}{ccc}\mathsf{c}_{01} & & \mathsf{x}\,\mathsf{x}\,\mathsf{u}\\ & \mathsf{c}_{00} & \\ & & \mathsf{c}_{00}\end{array}\;\mathsf{c}_{11}\right)[1_{(0)}] \longrightarrow {}^{0}\left(\begin{array}{ccc}\mathsf{c}_{00} & & \mathsf{x}\,\mathsf{x}\,\mathsf{u}\\ & \mathsf{c}_{01} & \\ & & \mathsf{c}_{01}\end{array}\;\mathsf{c}_{10}\right)[0_{(0)}]$$

The first reduction transformed the projection into the constant *1*, which in turn got transformed into $\neg 0$ before normalizing the functor. The associated generic rule is as follows:

$${}^{b}(\,\delta\omega\,)\,[\pi_{0,0,1}] \quad (\text{with } \omega \in \Delta^{+}_{\mathsf{uc}_\mathsf{t}}) \longrightarrow {}^{b}(\,\delta\omega'\,)\,[(\neg\mathsf{t})_0] \quad (\text{with } \omega' := \omega(\mathsf{c}_{\mathsf{tt}}) \in \Delta^{*}_{\mathsf{uc}_\mathsf{t}})$$

Likewise, we transform projections using xor-canalizing letter to exhibit a constant terminal, as in the following example:

$$^{1}\left(\begin{array}{ccc}\mathsf{c}_{01} & & \mathsf{x}\,\mathsf{x}\,\mathsf{u}\\ & \mathsf{s} & \\ & & \mathsf{x}\end{array}\;\mathsf{c}_{11}\right)[\pi_{0,0,1}] \longrightarrow {}^{1}\left(\begin{array}{ccc}\mathsf{c}_{01} & & \mathsf{x}\,\mathsf{x}\,\mathsf{u}\\ & \mathsf{x} & \\ & & \mathsf{x}\end{array}\;\mathsf{c}_{11}\right)[0_{(0)}]$$

The associated generic rule is as follows:

$${}^{b}(\,\delta\omega\,)\,[\pi_{0,0,1}] \quad (\text{with } \omega \in \Delta^{+}_{\mathsf{ux}}) \longrightarrow {}^{b}(\,\delta\omega'\,)\,[0_0] \quad (\text{with } \omega' := \omega(\mathsf{x}) \in \Delta^{+}_{\mathsf{ux}})$$

The next section details the necessary rules for factoring out common functors of the operands of the star operator.

[3] Observe that in this case the set of leaf functions L_Δ is exactly $\Delta^*\{0, 1\}$.

3.3.3 Interactions of the Shannon operator with Normalization Expressions

We start by defining the normalization rule for negation (a similar propagation rule is used in variants like ROBDD augmented with negation)

$$\left({}^{1}\left(\,\delta\,\right)\left(\phi_1, n_1\right)\right) \star \left(\phi_2, n_2\right) \longrightarrow \neg\left({}^{0}\left(\,\delta\,\right)\left(\phi_1, n_1\right) \star \neg\left(\phi_2, n_2\right)\right)$$

Then, we define the factorization problem for functor representation, starting with their tableau components. Given two tableaus δ_1 and δ_2, we want to compute tableaus δ (the gcd of δ_1 and δ_2, see Definition 6), δ_1' (denoted $\delta_1 \div \delta$) and δ_2' (denoted $\delta_2 \div \delta$) such that $\delta_1 = \delta\delta_1'$ and $\delta_2 = \delta\delta_2'$ and the pair $\delta_1'(\phi_1, n_1)$ and $\delta_2'(\phi_2, n_2)$ is coprime (see Definition 5). Let $(\mathsf{s} \cdot \delta)$ denote the operation of adding a 's' column to tableau δ, that is adding a new support variable. We introduce the following factorization rule:

$$\left(\delta_1\ f_1\right) \star \left(\delta_2\ f_2\right) \longrightarrow \left(\mathsf{s} \cdot \delta\right)\left(\delta_1'\ f_1 \star \delta_2'\ f_2\right) \quad .$$

We can express gcd recursively, with $\gcd(\omega_1\alpha_i\omega_2,\, \omega_1'\alpha_i\omega_2') = (\mathsf{s}^{|\omega_i|}\alpha_i\mathsf{s}^{|\omega_2|}) \cdot \gcd(\omega_1\omega_2,\, \omega_1'\omega_2')$. and $\gcd(\omega,\, \omega') = \mathsf{s}^{|\omega|}$ if $\forall i, \omega_i \neq \omega_i' \vee \omega_i = \mathsf{s}$. Let us see this factorization on an example. First we compute the gcd of both tableaus.

$$\gcd\left(\left(\begin{array}{ccc} \mathsf{c}_{0\emptyset} & & \\ & \mathsf{s} & \mathsf{u} \\ & & \mathsf{c}_{01} \end{array} \times \times \begin{array}{c} \\ \\ \mathsf{c}_{1\emptyset} \end{array}\right),\ \left(\begin{array}{ccc} \mathsf{c}_{0\emptyset} & & \\ & \mathsf{u} & \\ \mathsf{s} & & \mathsf{c}_{0\emptyset} \end{array} \times \times \begin{array}{c} \\ \\ \mathsf{u} \end{array}\right)\right) = \left(\begin{array}{ccc} \mathsf{c}_{0\emptyset} & & \\ & \mathsf{s} & \\ & & \mathsf{s} \end{array} \times \times \begin{array}{c} \\ \\ \mathsf{u} \end{array}\right)$$

We extend the greatest common divisor with an 's' column to encode the fact that a new variable is added at the first position.

$$\left(\mathsf{s} \cdot \left(\begin{array}{ccc} \mathsf{c}_{0\emptyset} & & \\ & \mathsf{s} & \\ & & \mathsf{s} \end{array} \times \times \begin{array}{c} \\ \\ \mathsf{u} \end{array}\right)\right) = \left(\begin{array}{ccc} \mathsf{s} & & \\ & \mathsf{c}_{0\emptyset} & \\ & & \mathsf{s} \end{array} \times \times \begin{array}{c} \\ \\ \mathsf{u} \end{array}\right)$$

We compute both remainders.

$$\left(\begin{array}{ccc} \mathsf{c}_{0\emptyset} & & \\ & \mathsf{s} & \mathsf{u} \\ & & \mathsf{c}_{01} \end{array} \times \times \begin{array}{c} \\ \\ \mathsf{c}_{1\emptyset} \end{array}\right) \div \left(\begin{array}{ccc} \mathsf{c}_{0\emptyset} & & \\ & \mathsf{s} & \\ & & \mathsf{s} \end{array} \times \times \begin{array}{c} \\ \\ \mathsf{u} \end{array}\right) = \left(\begin{array}{ccc} \mathsf{s} & & \\ & \mathsf{c}_{01} & \\ & & \mathsf{c}_{1\emptyset} \end{array}\right)$$

$$\left(\begin{array}{ccc} \mathsf{c}_{0\emptyset} & & \\ & \mathsf{u} & \\ \mathsf{s} & & \mathsf{c}_{0\emptyset} \end{array} \times \times \begin{array}{c} \\ \\ \mathsf{u} \end{array}\right) \div \left(\begin{array}{ccc} \mathsf{c}_{0\emptyset} & & \\ & \mathsf{s} & \\ & & \mathsf{s} \end{array} \times \times \begin{array}{c} \\ \\ \mathsf{u} \end{array}\right) = \left(\begin{array}{ccc} \mathsf{u} & & \\ & \mathsf{s} & \\ & & \mathsf{c}_{0\emptyset} \end{array}\right)$$

The gcd and division computations can be extended to functor representations with the negation supscript as follows:

- $\gcd({}^{b_1}\left(\,\delta_1\,\right), {}^{b_2}\left(\,\delta_2\,\right)) := {}^{0}\left(\,\gcd(b_1 \oplus \delta_1, b_2 \oplus \delta_2)\,\right)$
- ${}^{b_1}\left(\,\delta_1\,\right) \div {}^{b_2}\left(\,\delta_2\,\right) := {}^{b_1}\left(\,\delta_1 \div ((b_1 \oplus b_2) \oplus \delta_2)\,\right)$

leading to the following generic factorization rule:

$${}^{b_1}\left(\,\delta_1\,\right)\left(\phi_1, n_1\right) \star {}^{b_2}\left(\,\delta_2\,\right)\left(\phi_2, n_2\right) \longrightarrow {}^{0}\left(\,\mathsf{s}\cdot\delta\,\right)\left({}^{b_1}\left(\,\delta_1'\,\right)\left(\phi_1, n_1\right) \star {}^{b_2}\left(\,\delta_2'\,\right)\left(\phi_2, n_2\right)\right)$$

with:

- ${}^{0}\left(\,\delta\,\right) = \gcd\left({}^{b_1}\left(\,\delta_1\,\right), {}^{b_2}\left(\,\delta_2\,\right)\right)$
- ${}^{b_1}\left(\,\delta_1'\,\right) = {}^{b_1}\left(\,\delta_1\,\right) \div {}^{0}\left(\,\delta\,\right)$
- ${}^{b_2}\left(\,\delta_2'\,\right) = {}^{b_2}\left(\,\delta_2\,\right) \div {}^{0}\left(\,\delta\,\right)$

Below are two concrete examples:

$$^0\left(\ \mathsf{c}_{0\emptyset}\ {}_{\mathsf{s}}^{\times\times\mathsf{u}}\ \mathsf{c}_{1\emptyset}\ \right)(\phi_1,n_1)\quad\star\quad{}^0\left(\ \mathsf{c}_{0\emptyset}\ {}^{\mathsf{u}\times\times\mathsf{u}}\ \mathsf{s}\,\mathsf{c}_{0\emptyset}\ \right)(\phi_2,n_2)\longrightarrow$$
$$^0\left(\ \mathsf{s}\,\mathsf{c}_{0\emptyset}\ \mathsf{s}^{\times\times\mathsf{u}}\ \mathsf{s}\,\mathsf{s}\ \right)\quad\left({}^0\left(\ \mathsf{s}\,\mathsf{s}\,\mathsf{c}_{0\mathit{1}}\ {}^{\mathsf{c}_{1\emptyset}}\ \right)(\phi_1,n_1)\quad\star\quad{}^0\left(\ \mathsf{usc}_{0\emptyset}\ \right)(\phi_2,n_2)\right)$$

$$^0\left(\ \mathsf{c}_{0\emptyset}\ {}_{\mathsf{s}}^{\times\times\mathsf{u}}\ \mathsf{c}_{1\emptyset}\ \right)(\phi_1,n_1)\quad\star\quad{}^1\left(\ \mathsf{c}_{0\mathit{1}}\ {}^{\mathsf{u}\times\times\mathsf{u}}\ \mathsf{s}\,\mathsf{c}_{0\mathit{1}}\ \right)(\phi_2,n_2)\longrightarrow$$
$$^0\left(\ \mathsf{s}\,\mathsf{c}_{0\emptyset}\ \mathsf{s}^{\times\times\mathsf{u}}\ \mathsf{s}\,\mathsf{s}\ \right)\quad\left({}^0\left(\ \mathsf{s}\,\mathsf{s}\,\mathsf{c}_{0\mathit{1}}\ {}^{\mathsf{c}_{1\emptyset}}\ \right)(\phi_1,n_1)\quad\star\quad{}^1\left(\ \mathsf{usc}_{0\mathit{1}}\ \right)(\phi_2,n_2)\right)$$

The last normalization rule concerns factoring projections. As shown in the following example

$$[\pi_{1,2,7}]\ \star\ {}^1\left(\ \mathsf{c}_{0\mathit{1}}\ {}_{\mathsf{s}}^{\times\times\mathsf{u}}\ \mathsf{c}_{1\mathit{1}}\ \right)(\phi_1,n_1)\ \longrightarrow\ {}^1\left(\begin{array}{ccc}\mathsf{c}_{0\emptyset} & {}^{\times}\ {}^{\mathsf{u}} & \\ & \mathsf{c}_{0\mathit{1}}\ {}_{\mathsf{s}}^{\times} & \\ & & \mathsf{c}_{1\mathit{1}}\end{array}\right)(\phi_1,n_1)$$

projections need to be expanded in a specific way in order to exhibit common factors. We give the generic expansion when a projection appears on the left-hand side of the star operator and when the first row of the tableau of the right-hand side operand is in $\Delta_{\mathsf{u},\mathsf{x}}$. Let $\mathsf{x}_{i,n} := \left(\mathsf{s}^i\mathsf{x}\mathsf{s}^{n-i-1}\right) \in \Delta^*_{\mathsf{ux}}$ and $\mathsf{c}_{b,\mathsf{t},i,n} := \left(\mathsf{s}^i\mathsf{c}_{b\mathsf{t}}\mathsf{s}^{n-i-1}\right) \in \Delta^*_{\mathsf{ux}}$. Assuming the first row is a word ω in Δ_{ux} and $\omega_i = \mathsf{x}$, let ω' denote the word ω without its i-th element. Then the rule is as follows:

$$\underbrace{{}^{b_1}\left(\ \mathsf{u}^i\mathsf{s}\mathsf{u}^{n_1-i-1}\ \right)[\pi_{0,0,1}]}_{=\pi_{b_1,i,n_1}}\star\,{}^{b_2}\left(\ \omega\delta_2\ \right)(\phi_2,n_2)\longrightarrow{}^0\left(\ \mathsf{x}_{i+1,n_1+1}\ \right){}^0\left(\ \mathsf{c}_{0,b_1,0,n_1-1}\ \right){}^{b_2}\left(\ \omega'\delta_2\ \right)(\phi_2,n_2)$$

Similar rules can be formulated when the projection appears in the right-hand side of the star operator, and when the first row is a word in $\Delta_{\mathsf{uc_t}}$. An exhaustive list of reduction rules can be found in Appendix A.

When projections appear on both sides of the star, the factorization and normalization are taken care of by the already defined reduction rules, which yields:

$${}^{b_1}[\pi_{0,i_1,n}]\star{}^{b_2}[\pi_{0,i_2,n}]\quad(\text{with } i_1<i_2)\longrightarrow(\mathsf{u}^{i_1}\mathsf{s}\mathsf{u}^{i_2-i_1-1}\mathsf{s}\mathsf{u}^{n-i_2-1})\left({}^{b_1}[\pi_{0,0,n}]\star{}^{b_2}[\pi_{0,1,n}]\right)$$

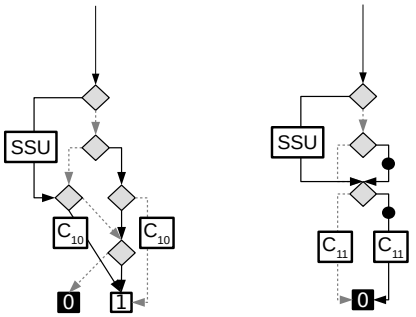
We end this section by the following important characterization of Boolean functions in the uniform model combining useless and all canalizing variables.

Theorem 2 (Characterization of Boolean Functions)**.** *Let f be a Boolean function of arity n, then*

- *if f is a projection, then, there exists a Boolean $b \in \mathbb{B}$ and an integer $i \in [\![0, n-1]\!]$ such that $f = \pi_{b,i,n}$;*
- *otherwise, f can be uniquely reduced to g using a functor in $\Delta^*_{\mathsf{uxc_t}}$ having the following form*

$$(\delta_{\mathsf{u},I_\mathsf{u}} \circ \delta_{p_k,I_k} \circ \cdots \circ \delta_{p_1,I_1})$$

  - *where $\forall i \in [\![1,k]\!]$, $p_i \in \{\mathsf{x}, \mathsf{c}_0, \mathsf{c}_1\}$,*
  - *g is $\Delta_{\mathsf{uxc_t}}$-irreducible,*
  - *$\forall i \in [\![1,k-1]\!]$, $p_i \neq p_{i+1}$,*
  - *if $f = \mathsf{t}_{(n_0)}$ then $n_0 = 0$, $p_1 \neq \mathsf{c_t}$ and $\delta_{p_1} \in \mathbb{B}^{0\Rightarrow n_1}$ with $|I_1| \geq 2$ (i.e., $n_1 \geq 2$).*



(a) λDD-U-UC$_{10}$ (b) λDD-U-NUC

Figure 3: Uniform λDD for $(x_0, x_1, x_2, x_3) \mapsto x_0 \oplus x_3 \oplus (\neg x_1 \wedge x_2)$

We proved this theorem in the proof assistant Coq to make sure that no corner cases were missed. [4] Indeed, as models become more elaborate, the combinatorics of the potential interactions between elementary functors on the one hand, and the arity-preserving operators on the other hand grows. This motivated the formalization of our models in a proof assistant where we formally proved the canonicity of the reduction. Notice that we get, as a corollary, similar statements for other simpler models (ordered and uniform).

Figure 3 shows the representations of the same running examples as in Figures 1 and 2 with respect to two uniform models. In this example, tableaus are simple rows. Notice in particular the linear structure of λDD-U-NUC, which is the most effective for this example.

4 λDD as a Unification Framework

Variant	**Model**	Δ	L_Δ
SDD	λDD-S	$\emptyset$	$\{0_{(0)}, 1_{(0)}\}$
SDD+N	λDD-NS	$\{\neg\}$	$\{0_{(0)}, 1_{(0)}\}$
ROBDD [1]	λDD-O-U	$\{\delta_{\mathsf{u}}\}$	$\{0_{(n)}, 1_{(n)}\}_{n\in\mathbb{N}}$
ROBDD+N [12]	λDD-O-NU	$\{\neg\} \cup \{\delta_{\mathsf{u}}\}$	$\{0_{(n)}, 1_{(n)}\}_{n\in\mathbb{N}}$
ZDD [9]	λDD-O-C$_{10}$	$\{\delta_{\mathsf{c}_{10}}\}$	$\{0_{(n)}, \delta^n_{\mathsf{c}_{10}} 1_{(0)}\}_{n\in\mathbb{N}}$
Chain-BDD [13]	λDD-O-UC$_{10}$	$\{\delta_{\mathsf{u}}, \delta_{\mathsf{c}_{10}}\}$	$\left\{0_{(n)}, \{\delta_{\mathsf{u}}, \delta_{\mathsf{c}_{10}}\}^n 1_{(0)}\right\}_{n\in\mathbb{N}}$
Chain-ZDD [13]	λDD-O-UC$_{10}$	$\{\delta_{\mathsf{u}}, \delta_{\mathsf{c}_{10}}\}$	$\left\{0_{(n)}, \{\delta_{\mathsf{u}}, \delta_{\mathsf{c}_{10}}\}^n 1_{(0)}\right\}_{n\in\mathbb{N}}$
ESR-L$_0$ [14]	λDD-O-UC$_{10}$	$\{\delta_{\mathsf{u}}, \delta_{\mathsf{c}_{10}}\}$	$\left\{0_{(n)}, \{\delta_{\mathsf{u}}, \delta_{\mathsf{c}_{10}}\}^n 1_{(0)}\right\}_{n\in\mathbb{N}}$
ESR [14]	λDD-O-UC$_0$	$\{\delta_{\mathsf{u}}, \delta_{\mathsf{c}_{00}}, \delta_{\mathsf{c}_{10}}\}$	$\left\{0_{(n)}, \{\delta_{\mathsf{u}}, \delta_{\mathsf{c}_{00}}, \delta_{\mathsf{c}_{10}}\}^n 1_{(0)}\right\}_{n\in\mathbb{N}}$

Table 2: Existing BDD variants with their corresponding λDD ordered models.

The functional point of view suggests a natural hierarchy to classify and enumerate all models, ordered and uniform, generated by elementary functors. As we shall detail in this section, it

[4] cf. supplemental material uploaded with the submission.

turns out that a large body of already existing BDD variants can be seen, through the lenses of Functional Decision Diagrams, as special ordered models, summarized in Table 2. One can see, at a glance, simply by comparing the elementary functors, the differences and similarities between those different variants. It is clear from the table that the recent attempts have been mainly driven by combining several existing variants to benefit from their sweet spots. For a set of elementary functor to be easily extended with negation, it has to be stable by negation in the first place, which is the case for λDD-O-U but not for λDD-O-C_{10} and other variants based on it. For example λDD-O-UC (the model induced by $\{\delta_{\mathsf{u}}\} \bigcup \{\delta_{\mathsf{c}_{bt}}\}_{b,t\in\mathbb{B}}$) containing all four canalizing variables and useless variables can be simply extended with negation into λDD-O-NUC. Section 3 details how to make these elementary functors compatible.

4.1 Common Variants as λDD Models

The most basic model consists in an empty Δ with two leaves corresponding to the constant Boolean functions *0* and *1* with arity zero. The so obtained structure is isomorphic to Shannon Decision Diagrams. We call this model λDD-S. We next consider arity-preserving functors of the form

$$(\delta_p f) : (x_0, \ldots, x_{n-1}) \mapsto p(f(x_0, \ldots, x_{n-1})),$$

where p is a Boolean function in $\mathbb{B}^{1\to 1}$. Essentially δ_p is a parametrized functor that transforms the output of the function it operates on (as is) using its parameter p, which is a Boolean function of arity 1. There are 4 possibilities for p, two of which, namely the constant functions *0* and *1*, are non-injective and therefore of little interest when it comes to canonical representations. The two candidates left for p are the identity function $\imath$ and its negation $\neg$. When p is the identity, one recovers λDD-S. Finally, the last case, we term λDD-NS, enriches λDD-S with the negation operator.

In a similar vein, let parameter p be a Boolean function of arity 2, combining the output of f with a fresh new variable. The general form would therefore be

$$(\delta_p f) : (x_0, \ldots, x_{n-1}) \mapsto p(x_0, f(x_1, \ldots, x_{n-1})) \ .$$

One can enumerate the 16 possible choices for p by combining two functions of arity 1, using the Shannon operator $\star$:

$\star$	$0_{(1)}$	$1_{(1)}$	$\imath_{(1)}$	$\neg\imath_{(1)}$
$0_{(1)}$	$0_{(2)}$	$(x,y) \mapsto x$	$(x,y) \mapsto x \wedge y$	$(x,y) \mapsto \neg(\neg x \vee y)$
$1_{(1)}$	$(x,y) \mapsto \neg x$	$1_{(2)}$	$(x,y) \mapsto \neg x \vee y$	$(x,y) \mapsto \neg(x \wedge y)$
$\imath_{(1)}$	$(x,y) \mapsto \neg x \wedge y$	$(x,y) \mapsto x \vee y$	$(x,y) \mapsto y$	$(x,y) \mapsto x \oplus y$
$\neg\imath_{(1)}$	$(x,y) \mapsto \neg(x \vee y)$	$(x,y) \mapsto \neg(\neg x \wedge y)$	$(x,y) \mapsto \neg(x \oplus y)$	$(x,y) \mapsto \neg y$

Constant functions (2 cases) as well as projections on the first (newly introduced) variable (2 cases) are non-injective and are denoted by $\perp$ in the table below. All remaining cases can be rewritten using the 6 operators we have already seen (encoding useless and canalizing variables), together with output negation as summarized below:

$\star$	$0_{(1)}$	$1_{(1)}$	$\imath_{(1)}$	$\neg\imath_{(1)}$
$0_{(1)}$	$\perp$	$\perp$	c_{00}	$\neg\mathsf{c}_{01}$
$1_{(1)}$	$\perp$	$\perp$	c_{01}	$\neg\mathsf{c}_{00}$
$\imath_{(1)}$	c_{10}	c_{11}	u	x
$\neg\imath_{(1)}$	$\neg\mathsf{c}_{11}$	$\neg\mathsf{c}_{10}$	$\neg\mathsf{x}$	$\neg\mathsf{u}$

Table 2 summarizes some models (and their respective variants) with names prefixed by λDD-O- (O standing for Ordered).

In particular, one may notice that Bryant's Chain-BDD, Chain-ZDD [13] and ESR−L_0 [14] are three possible implementations of the model called λDD-O-UC$_{10}$, the difference lying in their particular choices of encoding functors. In particular, it shows that while,on the one hand, the λDD framework is an efficient way of classifying existing models, combining them and finding new ones, on the other hand, it only provide a generic encoding and manipulation strategy, finding efficient algorithms and data structures remains at the user's discretion. Notice that the TBDD [15] variant uses a syntactic negation that does not correspond to the (functional) standard negation. It therefore does not fit in the current models but can be captured as a special model enriched with a functor that captures the semantics of its specific negation.

A similar enumeration could be performed for uniform models as summarized in Table 3.

Model	Δ	L_Δ
λDD-U-U	$\{\delta_{\mathsf{u},i}\}_{i\in\mathbb{N}}$	$\{0_{(n)}, 1_{(n)}\}_{n\in\mathbb{N}}$
λDD-U-NU	$\{\neg\} \cup \{\delta_{\mathsf{u},i}\}_{i\in\mathbb{N}}$	$\{0_{(n)}, 1_{(n)}\}_{n\in\mathbb{N}}$
λDD-U-NUC	$\{\neg\} \cup \{\delta_{\mathsf{u},i}\}_{i\in\mathbb{N}}$ $\cup\{\delta_{\mathsf{c}_{b\mathsf{t}},i}\}_{b,\mathsf{t}\in\mathbb{B},i\in\mathbb{N}}$	$\Delta^*\{0_{(0)}, 1_{(0)}\}$
λDD-U-NUX	$\{\neg\} \cup \{\delta_{\mathsf{u},i}, \delta_{\mathsf{x},i}\}_{i\in\mathbb{N}}$	$\{\{\delta_{\mathsf{u}}, \delta_{\mathsf{x}}\}^n\{0_{(0)}, 1_{(0)}\}\}_{b,\mathsf{t}\in\mathbb{B},i\in\mathbb{N}}$
λDD-U-NUCX	$\{\neg\} \cup \{\delta_{\mathsf{u},i}, \delta_{\mathsf{x},i}\}_{i\in\mathbb{N}}$ $\cup\{\delta_{\mathsf{c}_{b\mathsf{t}},i}\}_{b,\mathsf{t}\in\mathbb{B},i\in\mathbb{N}}$	$\Delta^*\{0_{(0)}, 1_{(0)}\}$

Table 3: Enumeration of some uniform models generated by elementary functors.

Apart from the above mentioned variants, the λDD framework can be instantiated to many other interesting variants (providing adequate reduction and normalization rules). Let us cite for instance, the Dual-Edge Based Variant [4], where one needs to add an arity preserving variant $\delta_{\mathtt{DE}}$ defined by:

$$(\delta_{\mathtt{DE}} f) : (x_0, \ldots, x_{n-1}) \mapsto \neg f(\neg x_0, \ldots, \neg x_{n-1}) \ .$$

Likewise, the Input Negation Invariant-Based Variant [2] can be captured by a λDD model with two functors parametrized by a vector of Booleans $a := (a_0, \ldots, a_{n-1}) \in \mathbb{B}^n$. The first functor, $\delta_{\oplus a}$, is arity-preserving and defined as follows:

$$(\delta_{\oplus a} f) : (x_0, \ldots, x_{n-1}) \mapsto f(x_0 \oplus a_0, \ldots, x_{n-1} \oplus a_{n-1}) \ .$$

The second functor, $\delta_{\beta,a,i}$, is elementary (it increases the arity by exactly one) and is defined as follows:

$$(\delta_{\beta,a,i} f) : \\ x \mapsto f\left(x_0 \oplus (x_i \wedge a_0), \ldots, x_{i-1} \oplus (x_i \wedge a_{i-1}), x_{i+1} \oplus (x_i \wedge a_{i+1}), \ldots, x_{n-1} \oplus (x_i \wedge a_{n-1})\right)$$

4.2 Generalizing λDD

The λDD framework can be further generalized in the three directions briefly discussed below. We inform the reader that this section serves the sole purpose of introducing interesting extensions of the current λDD framework. These extensions should mostly be considered as possible future work. In particular the notation introduced in this section will not used in the remainder of this article.

General Decision Diagrams So far, we only used one combinator, namely the Shannon operator $\star$ which is a *universal* and *elementary* combinator. It is universal since all non-constant functions have a pre-image through $\star$, and is elementary as it takes two functions of the same arity and increases the arity by 1.

λDDs are akin to Functional Decision Diagrams (FDDs) introduced by Kebschull et al. [16]. In both approaches, logic circuits are regarded (semantically) as Boolean functions. The main difference being the underlying operator (or combinator) used to build the diagrams. When FDD uses the Davio combonator, λDD uses the Shannon operator.

Becker and Drechsler [17] identified a total of 12 distinct universal and elementary combinators having the same form as Shannon's, except that ITE is substituted with an arbitrary function p. By allowing output negation, they further reduce this number to only 3, one of which is Shannon's. The other two combinators are:

- the positive Davio combinator defined by $(f \star_{\mathtt{D}^+} g)(x) = f(\mu_n(x)) \oplus (x_n \wedge g(\mu_n(x)))$, and
- the negative Davio combinator defined by $(f \star_{\mathtt{D}^-} g)(x) = f(\mu_n(x)) \oplus ((\neg x_n) \wedge g(\mu_n(x)))$.

One can go further in this direction, by introducing even more complex combinators [18, 19].

Disjoint Support Decomposition (DSD) Disjoint Support Combinators form yet another interesting class of operators that we would like to support in the near future. We say that a k-ary combinator ρ is a (p, σ)-disjoint support combinator

$$\rho(f_1, \ldots, f_k)(x) = p(f_1(\sigma_1(x)), \ldots, f_k(\sigma_k(x)))$$

where σ is a k-partition of $\{1, \ldots, n\}$, $p \in \mathbb{B}^{k \to 1}$ (called the primitive), f_i is an n_i-ary Boolean function (with $\sum_i n_i = n$), and $\sigma_i(x_1, \ldots, x_n) = (x_{\sigma_{i,1}}, \ldots, x_{\sigma_{i,n_i}}) \in \mathbb{B}^{n_i}$. Bertacco et al. [20] showed that BDD could be efficiently extended with such combinators by selecting p as the k-ary NOR operator. Furthermore, Bertacco proved [21] that any non-decomposable function can serve as a primitive of such combinator.

Multi-valued Decision Diagram (MDD)
We could extend our current framework to integrate Multi-valued decision diagrams. In particular, if the decision variables have a finite domain, we can easily extend the Shannon combinator $\star$ to take the right number of input functions. However, if the decision variables have an infinite domain, this simple extension does not work anymore, and trickier combinators have to be used in order to conserve the finiteness of the structure.

4.3 Related Work On The Classification of Boolean Functions

The λDD-based classification differs from Darwiche's work [22] which uses relative compactness and absolute worst-time complexity of standard queries to classify representations of Boolean functions. Firstly, λDD is more fine grained. With respect to Darwiche's classification many important variants like BDD [1], BDD+N [11], ZDD [9], Chain-DD [13], TBDD [15], and ESRBDD [14] fall in the same class and are therefore mostly indistinguishable from the vanilla variant SDD, The only difference being that BDD-like variants, contrary to ZDD-like variants, handle negation in constant time and linear space. We have seen how λDD is able to highlight the similarities and differences of those variants. Secondly, unlike Darwiche's classification, which is blind to potential conflicts when merging two distinct variants, the λDD framework focuses primarily on combining variants in a principled and systematic way.

Conclusion

We introduced a functional framework for Binary Decision Diagrams together with a related data structure, we called λDD. We showed that most existing variants of BDD, including recent ones, can be regarded as special ordered models of our framework. Our functional point of view allows to combine in a principled way the properties of interest one wants to capture. More importantly, through factoring common functors, the uniform models suggest, by construction, a particular variable ordering that further reduces the overall size of the structure. We detailed the necessary reduction rules (introduction, normalization and factorization) for a rich uniform model that captures both useless and canalizing variables together with negation, and formalized and proved the canonicity of the reduction in a proof assistant.

Several ordered models corresponding to existing variants have been implemented as part of the DAGaml software: λDD-O-U (ROBDD), λDD-O-NU (ROBDD + complemented edges), λDD-O-C$_{10}$ (ZDD), λDD-O-UC$_0$ (ESRBDD).

As part of our future work, we plan to investigate more models suggested by the λDD framework and their relations to existing ones. In particular, the λDD uniform models can be seen as interesting special cases of Disjoint Support Decomposition [21] where one branch is of arity 1, although an additional effort is required to better understand the link between both.

References

[1] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. 35, pp. 677–691, Aug. 1986.

[2] J. R. Burch and D. E. Long, "Efficient boolean function matching," in *Proceedings of the 1992 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '92, (Los Alamitos, CA, USA), pp. 408–411, IEEE Computer Society Press, 1992.

[3] S. Minato, N. Ishiura, and S. Yajima, "Shared binary decision diagram with attributed edges for efficient boolean function manipulation," in *27th ACM/IEEE Design Automation Conference*, pp. 52–57, Jun 1990.

[4] D. M. Miller and R. Drechsler, "Dual edge operations in reduced ordered binary decision diagrams," in *Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on*, vol. 6, pp. 159–162 vol.6, May 1998.

[5] C. Meinel and A. Slobodová, "A unifying theoretical background for some bdd-based data structures," *Form. Methods Syst. Des.*, vol. 11, pp. 223–237, Oct. 1997.

[6] P. Kerntopf, "New generalizations of shannon decomposition," 2001.

[7] B. Bollig and I. Wegener, "Improving the variable ordering of OBDDs is NP-complete," *IEEE Transactions on Computers*, vol. 45, pp. 993–1002, Sept. 1996.

[8] T. C. development team, *The Coq proof assistant reference manual*. LogiCal Project, 2004. Version 8.0.

[9] S. Minato, "Zero-suppressed bdds for set manipulation in combinatorial problems," in *Proceedings of the 30th International Design Automation Conference*, DAC '93, (New York, NY, USA), pp. 272–277, ACM, 1993.

[10] A. Mishchenko, "An introduction to zero-suppressed binary decision diagrams," tech. rep., in 'Proceedings of the 12th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, 2001.

[11] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *Proceedings of the 27th ACM/IEEE Design Automation Conference*, DAC '90, (New York, NY, USA), pp. 40–45, ACM, 1990.

[12] F. Somenzi, "Binary decision diagrams," 1999.

[13] R. E. Bryant, "Chain reduction for binary and zero-suppressed decision diagrams," *CoRR*, vol. abs/1710.06500, 2017.

[14] J. Babar, C. Jiang, G. Ciardo, and A. Miner, "Binary Decision Diagrams with Edge-Specified Reductions," in *Tools and Algorithms for the Construction and Analysis of Systems* (T. Vojnar and L. Zhang, eds.), Lecture Notes in Computer Science, pp. 303–318, Springer International Publishing, 2019.

[15] T. van Dijk, R. Wille, and R. Meolic, "Tagged bdds: Combining reduction rules from different decision diagram types," in *FMCAD*, pp. 108–115, IEEE, 2017.

[16] U. Kebschull, E. Schubert, and W. Rosenstiel, "Multilevel logic synthesis based on functional decision diagrams," in *[1992] Proceedings The European Conference on Design Automation*, pp. 43–47, Mar. 1992. ISSN: null.

[17] B. Becker and R. Drechsler, "How many decomposition types do we need? [decision diagrams]," in *EDTC*, 1995.

[18] A. Bernasconi, V. Ciriani, G. Trucco, and T. Villa, "On decomposing boolean functions via extended cofactoring," in *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, (3001 Leuven, Belgium, Belgium), pp. 1464–1469, European Design and Automation Association, 2009.

[19] L. Amarú, P. Gaillardon, and G. D. Micheli, "Biconditional bdd: A novel canonical bdd for logic synthesis targeting xor-rich circuits," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1014–1017, March 2013.

[20] V. Bertacco and M. Damiani, "Boolean function representation based on disjoint-support decompositions," in *Proceedings International Conference on Computer Design. VLSI in Computers and Processors*, pp. 27–32, Oct 1996.

[21] V. M. Bertacco, *Achieving Scalable Hardware Verification with Symbolic Simulation.* PhD thesis, Stanford, CA, USA, 2003. AAI3104197.

[22] A. Darwiche and P. Marquis, "A Knowledge Compilation Map," *1*, vol. 17, pp. 229–264, Sept. 2002.

A Appendix: Reduction Rules

We present here an exhaustive list of elementary reduction rules for uniform models. We define three main alphabets $\Delta_{\mathsf{sux}} := \{\mathsf{s}, \mathsf{u}, \mathsf{x}\}$, $\Delta_{\mathsf{suc}_0} := \{\mathsf{s}, \mathsf{u}, \mathsf{c}_{00}, \mathsf{c}_{10}\}$ and $\Delta_{\mathsf{suc}_1} := \{\mathsf{s}, \mathsf{u}, \mathsf{c}_{01}, \mathsf{c}_{11}\}$. A word over one of these alphabet is denoted by (ω); if one wants to specify that some symbol α appears at a specific position i, we denote it $(\omega_1 \alpha_i \omega_2)$. If one wants to make explicit which symbols are allowed, they are put as an index for the whole word, e.g., $(\omega)_{\mathsf{su}}$ denotes a word in alphabet $\{\mathsf{s}, \mathsf{u}\}^*$. As a reminder, we denote the Shannon operator by $\star$ and a formula of arity n by (ϕ, n).

`intro-u`	$(\phi, n) \star (\phi, n)$	$\longrightarrow$	$(\mathsf{u}\mathsf{s}^n)\,(\phi, n)$
`intro-x`	$(\phi, n) \star \neg(\phi, n)$	$\longrightarrow$	$(\mathsf{x}\mathsf{s}^n)\,(\phi, n)$
`intro-c0t`	$[\mathsf{t}_{(n)}] \star (\phi, n)$	$\longrightarrow$	$(\mathsf{c}_{0\mathsf{t}}\mathsf{s}^n)\,(\phi, n)$
`intro-c1t`	$(\phi, n) \star [\mathsf{t}_{(n)}]$	$\longrightarrow$	$(\mathsf{c}_{1\mathsf{t}}\mathsf{s}^n)\,(\phi, n)$

Table 4: Introduction Rules

`norm-neg-sha`	$\neg(\phi_1, n) \star (\phi_2, n)$	$\longrightarrow$	$\neg\left((\phi_1, n) \star \neg(\phi_2, n)\right)$
`norm-neg-up`	$(\omega)\neg$	$\longrightarrow$	$\neg(\overline{\omega})$
`norm-cst`	$[\mathsf{t}_{(n)}]$	$\longrightarrow$	$(\mathsf{u}^n)[\mathsf{t}_{(0)}]$
`norm-pi-neg`	$[\pi_{1,i,n}]$	$\longrightarrow$	$\neg[\pi_{0,i,n}]$
`norm-pi-u`	$[\pi_{b,i,n}]$	$\longrightarrow$	$\neg(\mathsf{u}^i\mathsf{s}\mathsf{u}^{n-i-1})[\pi_{b,0,1}]$
`norm-comp`	$(\omega_1)_\Delta(\omega_2)_\Delta$	$\longrightarrow$	$(\omega_1 \circ \omega_2)_\Delta$
`norm-cbtt`	$(\omega)_{\mathsf{suc}_{b\mathsf{t}}}(\mathsf{t}, n)$	$\longrightarrow$	$(\mathsf{u}^{\lvert\omega\rvert})[\mathsf{t}_{(0)}]$
`norm-cbntt`	$(\mathsf{u}^i\mathsf{c}_{b(\neg\mathsf{t})}\mathsf{u}^j)[\mathsf{t}_{(0)}]$	$\longrightarrow$	$(\mathsf{u}^i\mathsf{s}\mathsf{u}^j)[\pi_{b\oplus\mathsf{t},0,1}]$
`norm-xt`	$(\mathsf{u}^i\mathsf{x}\mathsf{u}^j)[\mathsf{t}_{(0)}]$	$\longrightarrow$	$(\mathsf{u}^i\mathsf{s}\mathsf{u}^j)[\pi_{0,0,1}]$
`norm-xpi`	$(\omega)_{\mathsf{sux}}[\pi_{0,0,1}]$	$\longrightarrow$	$(\omega \circ (\mathsf{x}))[0_{(0)}]$
`norm-cpi`	$(\omega)_{\mathsf{suc}_\mathsf{t}}[\pi_{0,0,1}]$	$\longrightarrow$	$(\omega \circ (\mathsf{c}_{\mathsf{tt}}))[\neg\mathsf{t}_{(0)}]$

Table 5: Normalization Rules

We define the dual operator $\overline{\cdot}$ on symbols: $\overline{\mathsf{s}} = \mathsf{s}$, $\overline{\mathsf{u}} = \mathsf{u}$, $\overline{\mathsf{x}} = \mathsf{x}$, and $\overline{\mathsf{c}_{b\mathsf{t}}} = \mathsf{c}_{b(\neg\mathsf{t})}$. We extend the dual operator to words by applying it symbol wise. One may check the following semantic property $(\omega)(\neg f) = \neg((\overline{\omega}(f)))$. Rule `norm-neg-up` states that the negation operator always move upward in the structure. Rule `norm-comp` states that, if two consecutive words belong to the same basic alphabet, then these two words are composed (we denote by $\circ$ the explicit word composition operator). Note that, if a word belongs to $\{\mathsf{s}, \mathsf{u}\}^*$, then it belongs to the three basic alphabets, otherwise it belongs to exactly one of the three basic alphabets.

Table 6 summarizes the factorization rules, using the following notations: notations are used:

- $\oplus_b\phi := \begin{cases} \phi & \text{if } b = 0 \\ \neg\phi & \text{if } b = 1 \end{cases}$
- $\pi^{\#}_{b,i,n} := \oplus_b(\mathsf{u}^i\mathsf{s}\mathsf{u}^{n-i-1})[\pi_{0,0,1}]$

$$\text{facto-noneg}\frac{(\omega_1\alpha_i\omega_2)\,(\phi,n)\star(\omega_1'\alpha_i\omega_2')\,(\phi',n')}{(\mathsf{s}^{|\omega_1|+1}\alpha_{i+1}\mathsf{s}^{|\omega_2|})\,((\omega_1\omega_2)(\phi,n)\star(\omega_1'\omega_2')(\phi',n'))}$$

$$\text{facto-neg}\frac{(\omega_1\alpha_i\omega_2)\,(\phi,n)\star\neg(\omega_1'\overline{\alpha}_i\omega_2')\,(\phi',n')}{(\mathsf{s}^{|\omega_1|+1}\alpha_{i+1}\mathsf{s}^{|\omega_2|})\,((\omega_1\omega_2)(\phi,n)\star\neg(\omega_1'\omega_2')(\phi',n'))}$$

$$\text{facto-pi-x}\frac{\pi^{\#}_{b_1,i,n}\star\oplus_{b_2}(\omega_1\mathsf{x}_i\omega_2)(\phi_2,n_2)}{(\mathsf{x}_{i+1,n+1})\,(\mathsf{c}_{0,b_1,0,n})\,\oplus_{b_2}(\omega_1\omega_2)(\phi_2,n_2)}$$

$$\text{facto-x-pi}\frac{\oplus_{b_2}(\omega_1\mathsf{x}_i\omega_2)(\phi_2,n_2)\star\pi^{\#}_{b_1,i,n}}{(\mathsf{x}_{i+1,n+1})\,(\mathsf{c}_{1,b_1,0,n})\,\oplus_{b_2}(\omega_1\omega_2)(\phi_2,n_2)}$$

$$\text{facto-pi-c}\frac{\pi^{\#}_{b_1,i,n}\star\oplus_{b_2}(\omega_1\mathsf{c}_{b_3\mathsf{t},i}\omega_2)(\phi_2,n_2)\quad(\text{with } b_1\oplus b_2=b_3\oplus\mathsf{t})}{(\mathsf{c}_{0,(b_1\oplus b_3),i+1,n+1})\,(\mathsf{c}_{0,\neg(b_1\oplus b_3),0,n})\,\oplus_{b_2}(\omega_1\omega_2)(\phi_2,n_2)}$$

$$\text{facto-c-pi}\frac{\oplus_{b_2}(\omega_1\mathsf{c}_{b_3\mathsf{t},i}\omega_2)(\phi_2,n_2)\star\pi^{\#}_{b_1,i,n}\quad(\text{with } b_1\oplus b_2=b_3\oplus\mathsf{t})}{(\mathsf{c}_{1,(b_1\oplus b_3),i+1,n+1})\,(\mathsf{c}_{1,\neg(b_1\oplus b_3),0,n})\,\oplus_{b_2}(\omega_1\omega_2)(\phi_2,n_2)}$$

Table 6: Factorization Rules

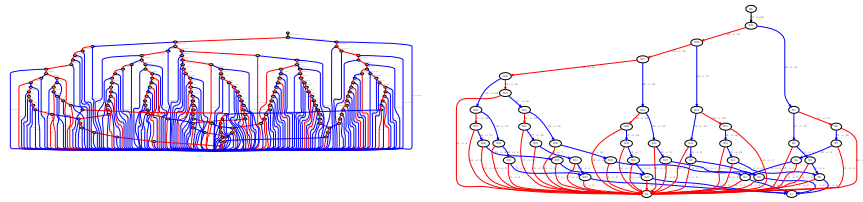
B Appendix: More Examples

B.1 N-Queens Problem

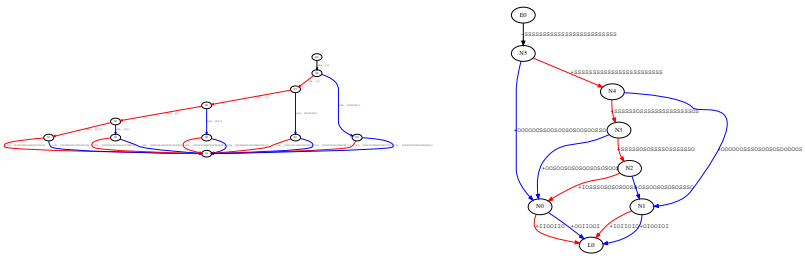
The n-queens problem is the problem of placing n chess queens on a $n \times n$ chess board, so that no queen can reach another in one step. Thus no two queens can be on the same row, column or diagonal. We represent the solution of the 5-queens problem in Figure 4 using the quadratic encoding, i.e., each cell of the chess board is encoded as a distinct variable, whose value is true if and only if there is a queen on this cell. The graphical representation have been generated using DAGaml and Graphviz. Blue edges represent `if 1` edges and red edges represent `if 0` edges. The annotation `Nxxx` in the nodes is used to represent the index of the node, and should not be confused with variable indexing in ROBDD. The annotation `L0` in the leaf (which semantic interpretation is $0_{\mathbb{B}}$) is only used to simply identify the leaf node when reading the file. One can easily see the compression from ROBDD to λ`DD-U-NUC` which allows to have a more human-readable representation of the solutions.

B.2 CNF formulas

Figure 5 compares the, namely representations of the CNF `satlib/uf20-91/uf20-01.cnf` using four different models λ`DD-O-NU` (aka. `ROBDD+N`), λ`DD-O-`C_{10} (aka. ZDD), λ`DD-O-`UC_0 (aka. `ESRBDD`) and our latest model λ`DD-U-NUC`.

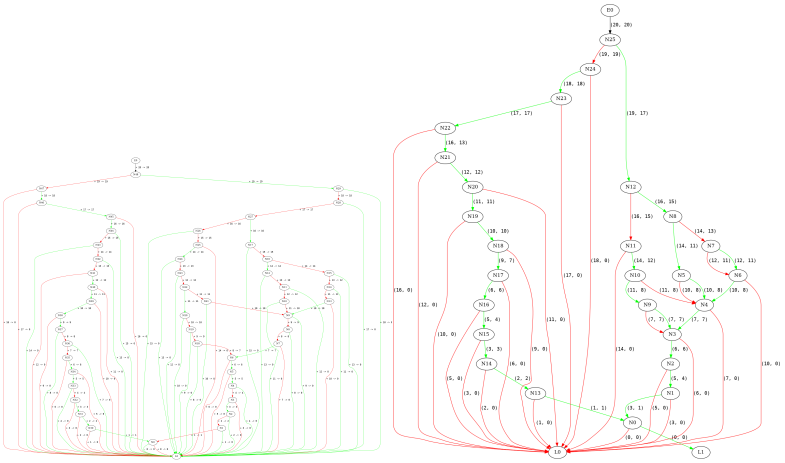


(a) λDD-U-NU (b) ZDD

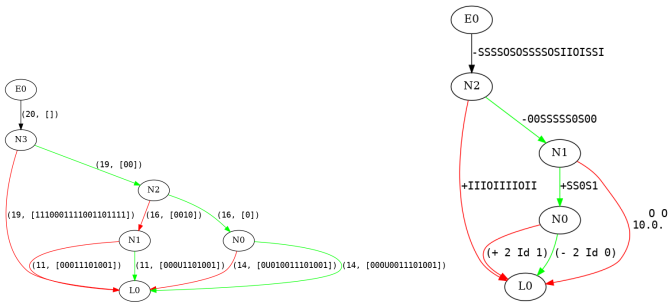


(c) λDD-O-NUC (d) λDD-U-NUC

Figure 4: Shows the representation of solutions of the 5-queens problem using either λDD-U-NU (which has the same size as ROBDD in this case) 4a, ZDD 4b, λDD-O-NUC 4c or λDD-U-NUC 4d.

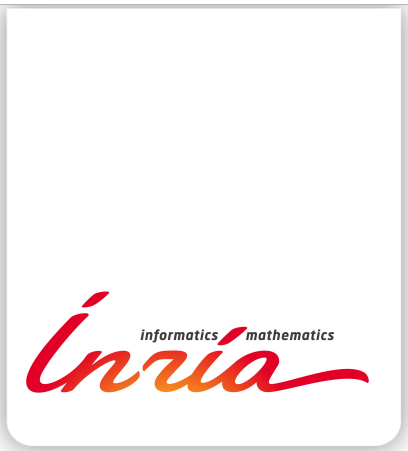


(a) λDD-O-NU (aka. ROBDD+N) (b) λDD-O-C$_{10}$ (aka. ZDD)



(c) λDD-O-UC$_0$ (aka. ESRBDD) (d) λDD-U-NUC

Figure 5: Shows the representation of solutions of the CNF `satlib/uf20-91/uf20-01.cnf` using four different models λDD-O-NU 5a (aka. ROBDD+N), λDD-O-C$_{10}$ 5b (aka. ZDD), λDD-O-UC$_0$ 5c (aka. ESRBDD) and our latest model λDD-U-NUC 5d.



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399