# CONTEXT DECISION DIAGRAMS

CALEB KOCH

ABSTRACT. We present a decision-diagram inspired data structure called context decision diagrams (CDDs). CDDs are like forwarding decision diagrams but with the property that they are canonical, i.e. two reduced CDDs give rise to the same function interpretation iff they are structural the same. This result is analogous to the canonicity result for reduced order binary decision diagrams. Our data structure also has the added property that the identity and constant functions are represented by CDDs of size $O(1)$. Finally we present an efficient recursive procedure for building CDDs which is analogous to the procedure for BDDs.

## 1 Background

NetKAT is a network programming language, originally introduced in [AFG$^+$14]. Forwarding decision diagrams (FDDs) were introduced in [SEFG15] as an intermediate representation of NetKAT programs. They serve as a generalization of binary decision diagrams (BDDs) originally introduced in [Bry86]. Like BDDs, operations on FDDs are efficient, and FDDs can represent large programs compactly. They also support direct, efficient translation into forwarding tables. However, unlike BDDs, FDDs are not canonical. We seek to resolve this issue while maintaining the other desirable properties of FDDs.

## 2 Definitions

### 2.1 Notational Conventions and Arithmetic

Fix $n > 0$. Let $[n] = \{1, ..., n\}$ and let $\mathbb{B} = \{0, 1\}$ be constant bits and $\mathbb{T} = \mathbb{B} \cup \{\star\} = \{0, 1, \star\}$ be ternary bits. We denote the set of length $n$ vectors with entries in $\mathbb{B}$ as $\mathbb{B}^n$ and likewise for $\mathbb{T}^n$. Throughout we use the following notation conventions

- $a, b, c$ typically refer to vectors in $\mathbb{B}^n$
- $u, v, w$ typically refer to vectors in $\mathbb{T}^n$
- $x, y, z$ typically refer to bits (i.e. elements of $\mathbb{B}$ or $\mathbb{T}$).
- $\mathbf{0}$ is the zero vector.
- $\vec{\star}$ is the vector $\begin{bmatrix} \star \\ \vdots \\ \star \end{bmatrix}$.

We start by defining arithmetic on $\mathbb{T}^n$. Our strategy is to define operations on single bits then extend pointwise to vectors.

We define multiplication, $\cdot : \mathbb{T} \times \mathbb{T} \to \mathbb{T}$ on $\mathbb{T}$ as follows: let $x, y \in \mathbb{T}$ then

$$x \cdot y = \begin{cases} x & \text{if } y = \star \\ y & \text{otherwise.} \end{cases}$$

We will simply write $xy$ for $x \cdot y$. We define $x + y = x \vee y$ where $\vee$ is the join with respect to the ordering $\star < 0 < 1$. As noted we lift these operations to be defined on $\mathbb{T}^n \times \mathbb{T}^n$ pointwise. Scalar multiplication is defined as an operation of the form $\cdot : \mathbb{B} \times \mathbb{T}^n \to \mathbb{T}^n$, for $x \in \mathbb{B}$ and $u \in \mathbb{T}^n$

$$xu = \begin{cases} \mathbf{0} & \text{if } x = 0 \\ u & \text{otherwise.} \end{cases}$$

The following lemma will be useful later on.

**Lemma 2.1.** *If $u, v \in \mathbb{T}^n$ and $bu = bv$ for all $b \in \mathbb{B}^n$ then $u = v$.*

*Proof.* Let $i \in [n]$ be arbitrary. Note that $y(u)_i = y(v)_i$ for any $y \in \mathbb{B}$ since $bu = bv$. We'll show that $(u)_i = (v)_i$. Suppose $(u)_i \neq (v)_i$. Then at least one of $(u)_i$ or $(v)_i$ is a constant (i.e. non$-\star$) bit. Without loss of generality suppose $(u)_i = x$, a constant bit. We have

$$0(v)_i = 0(u)_i = x = 1(u)_i = 1(v)_i$$

and since $x = 0(v)_i = 1(v)_i$ we must have $(v)_i = x = (u)_i$, a contradiction. □

## 2.2 Context Decision Diagrams

We wish to define a tree based representation of boolean functions $f : \mathbb{B}^n \to \mathbb{B}^n$. Our representation is similar to that of FDDs or BDDs where we think of the tree as "filtering" input vectors into categories which represent different ways the input is transformed by $f$ to achieve the output vector. The key idea is to keep a context at each node in the tree which intuitively represents at that point in the tree the vectors that haven't yet been "filtered" away into different categories.

Formally a **context** is simply a vector $\Gamma \in \mathbb{T}^n$. Ternary bits are useful here since we can think of $\star$ as a "wildcard" representing either 0 or 1 and thus we can define a subset of $\mathbb{B}^n$ "generated" by $\Gamma \in \mathbb{T}^n$ by expanding entrywise the $\star$ bits in $\Gamma$. Formally we denote $\langle\Gamma\rangle$ as the subset of $\mathbb{B}^n$ generated by $\Gamma \in \mathbb{T}^n$ and define it as

$$\langle\Gamma\rangle = \{b \in \mathbb{B}^n : (b)_i = (\Gamma)_i \text{ if } (\Gamma)_i \neq \star\}$$

where $(v)_i$ is notation for the $i$th entry of a vector $v$.

Our definitions follow a three part process, much in the spirit of the way one might define decision diagram, then binary decision diagram, then reduced ordered binary decision diagram. In our case, we define context tree, then context decision diagram, then reduced context decision diagram.

The following will be useful in defining our main structure

**Definition 2.2** (Well-formedness)**.** A pair of vectors $(u, v) \in \mathbb{T}^n \times \mathbb{T}^n$ is said to be well formed if the following condition is met for $i \in [n]$:

$$(u)_i = (v)_i \Rightarrow (u)_i = \star.$$

**Definition 2.3** (Context tree)**.** A context tree on $\mathbb{T}^n$ is a binary tree defined inductively:

$$\text{A leaf } \ell ::= (\Gamma, u) \in \mathbb{T}^n \times \mathbb{T}^n, \text{ where } (\Gamma, u) \text{ is a well formed pair}$$
$$\text{A branch } n ::= (i, \Gamma, T_0, T_1),\ i \in [n] \text{ called the test index}, \Gamma \in \mathbb{T}^n, \text{ and } T_0, T_1 \text{ are trees}$$
$$\text{A context tree } T ::= \ell \mid n \text{ where } \ell \text{ is a leaf, } n \text{ is a branch.}$$

For a context tree $T$ we use $C(T)$ to denote the context of $T$ (i.e. $C(\Gamma, u) = \Gamma$ and $C(i, \Gamma, T_0, T_1) = \Gamma$).

**Definition 2.4** (Function interpretation)**.** Let $T$ be a context tree on $\mathbb{T}^n$. The action of $T$, denoted $(\!|T|\!)$ is a function $\mathbb{B}^n \to \mathbb{T}^n$ defined inductively for $b \in \mathbb{B}^n$:

$$(\!|(\Gamma, u)|\!)(b) ::= u$$
$$(\!|(i, \Gamma, T_0, T_1)|\!)(b) ::= [(b)_i = 0](\!|T_0|\!)(b) + [(b)_i = 1](\!|T_1|\!)(b)$$

where $[\cdot]$ is the Iverson bracket. We can simultaneously view $(\!|T|\!)$ as a function from $\mathbb{B}^n$ to $[\mathbb{B}^n \to \mathbb{B}^n]$, the set of functions from $\mathbb{B}^n$ to $\mathbb{B}^n$, by the multiplication map. That is, $(\!|T|\!)$ gives rise naturally to the map $b' \mapsto b'(\!|T|\!)(b)$ (multiplication by $b'$). This observation leads to the function interpretation $[\![T]\!] : \mathbb{B}^n \to \mathbb{B}^n$ defined by $[\![T]\!](b) = (\!|T|\!)(b)(b) = b(\!|T|\!)(b)$.

Note that $[\![T]\!]$ is independent of contexts. This fact will come up later.

In the next two definitions we use the following notation: $\Gamma[x/i]$ to refer to the substitution of the bit $x \in \mathbb{T}$ for whatever bit is in the $i$th position of $\Gamma \in \mathbb{T}^n$.

**Definition 2.5** (Context Decision Diagram)**.** A context decision diagram (CDD) on $\mathbb{T}^n$ is a finite context tree $T$ on $\mathbb{T}^n$ satisfying the following for all branches $n = (i, \Gamma, T_0, T_1)$ in $T$

1. [Well formed context] For all $j \in \{i, ..., n\}$ we have $(\Gamma)_j = \star$.

2. [Proper succession] For $T_k$, $k \in \{0,1\}$ we have $\Gamma[k/i] = C(T_k)$.
3. [Order] The test index of $n$ is strictly less than the test indices of the children of $n$.

We say that a CDD $T$ is **initial** if $C(T) = \vec{\star}$, the vector whose entries are all $\star$.

**Definition 2.6** (Reduced CDD)**.** A reduced CDD is a CDD $T$ such that for every branch $n = (i, \Gamma, T_0, T_1)$ in $T$, there is an associated *certificate of reducedness* which is a pair of vectors $(b_0, b_1)$, $b_0, b_1 \in \mathbb{B}^n$ such that $b_0 = b_1[0/i]$ and $b_1 = b_0[1/i]$ and

$$[\![T_0]\!](b_0) \neq [\![T_1]\!](b_0)$$
$$[\![T_0]\!](b_1) \neq [\![T_1]\!](b_1).$$

We note that the reducedness condition is necessary for obtaining canonicity. One might suspect that a simpler condition such as no isomorphic subgraphs or children (as with BDDs) would be sufficient; however, in the case of CDDs, this is not the case. Consider for example, the following CDD.

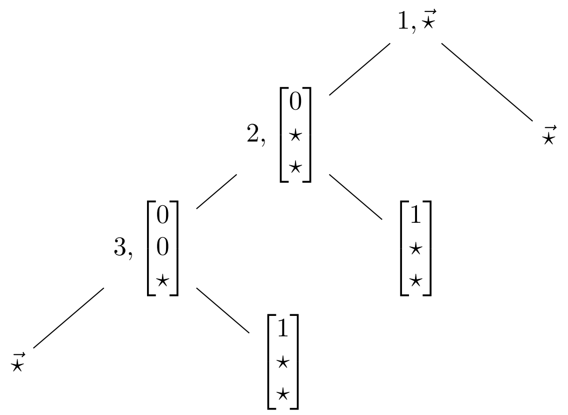

$1, \vec{\star}$
$2, \begin{bmatrix}0\\\star\\\star\end{bmatrix}$
$\vec{\star}$
$3, \begin{bmatrix}0\\0\\\star\end{bmatrix}$
$\begin{bmatrix}1\\\star\\\star\end{bmatrix}$
$\vec{\star}$
$\begin{bmatrix}1\\\star\\\star\end{bmatrix}$

FIGURE 1. A CDD that is not reduced

**Example 2.7** (Necessity of reducedness condition for canonicity)**.** Note that in this diagrammatic representation, we exclude the contexts on the leaf nodes since they can be inferred from the parent's context. This tree is not reduced since if we write the CDD as $(1, \star, T_0, T_1)$ then it is straightforward to check that $[\![(1, \star, T_0, T_1)]\!] = [\![T_0]\!]$ and so no such certificate could exist as in the reducedness condition. Notice though that $T_0$ is in fact a reduced CDD. Namely, Figure 2 is the initial reduced CDD equivalent to $[\![(1, \star, T_0, T_1)]\!]$ (in interpretation).

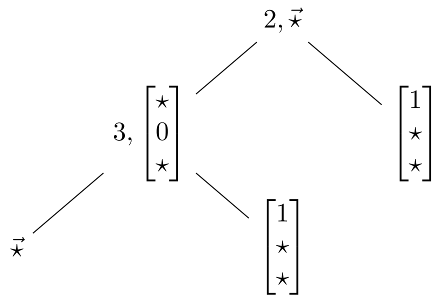

$2, \vec{\star}$
$3, \begin{bmatrix}\star\\0\\\star\end{bmatrix}$
$\begin{bmatrix}1\\\star\\\star\end{bmatrix}$
$\vec{\star}$
$\begin{bmatrix}1\\\star\\\star\end{bmatrix}$

FIGURE 2. Equivalent reduced initial CDD

One certificate for the reduced CDD in Figure 2 is the pair

$$\left(\begin{bmatrix}0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}\right).$$

One might try to adjust the reducedness condition to something simpler, say perhaps that $[\![T_0]\!]|_{\langle\Gamma\rangle} \neq [\![T_1]\!]|_{\langle\Gamma\rangle}$. This condition again fails to provide canonicity. Consider the following CDD

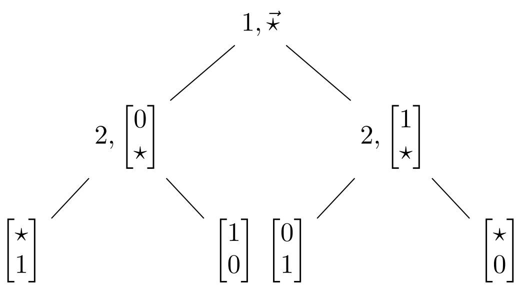

$1, \vec{\star}$
$2, \begin{bmatrix}0\\\star\end{bmatrix}$
$2, \begin{bmatrix}1\\\star\end{bmatrix}$
$\begin{bmatrix}\star\\1\end{bmatrix}$
$\begin{bmatrix}1\\0\end{bmatrix}$
$\begin{bmatrix}0\\1\end{bmatrix}$
$\begin{bmatrix}\star\\0\end{bmatrix}$

FIGURE 3. Another CDD that is not reduced

Let $T = (1, \vec{\star}, T_0, T_1)$ be the CDD in Figure 3. Then we have

$$[\![T_0]\!]\left(\begin{bmatrix}0\\1\end{bmatrix}\right) = \begin{bmatrix}1\\0\end{bmatrix} \neq \begin{bmatrix}0\\0\end{bmatrix} = [\![T_1]\!]\left(\begin{bmatrix}0\\1\end{bmatrix}\right)$$

and so $[\![T_0]\!]|_{\langle\Gamma\rangle} \neq [\![T_1]\!]|_{\langle\Gamma\rangle}$. However, $[\![T]\!]$ is equivalent to the interpretation of the following CDD.

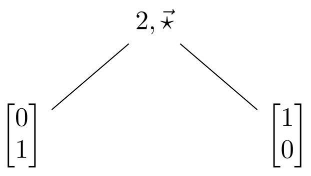

$2, \vec{\star}$
$\begin{bmatrix}0\\1\end{bmatrix}$
$\begin{bmatrix}1\\0\end{bmatrix}$

FIGURE 4. An equivalent initial CDD.

The problem is that in particular the first CDD does not satisfy the reducedness of Definition 2.6.

## 3 Properties

*Remark* 3.1. If $T = (i, \Gamma, T_0, T_1)$ is a reduced CDD then $T_0$ and $T_1$ are also reduced CDDs.

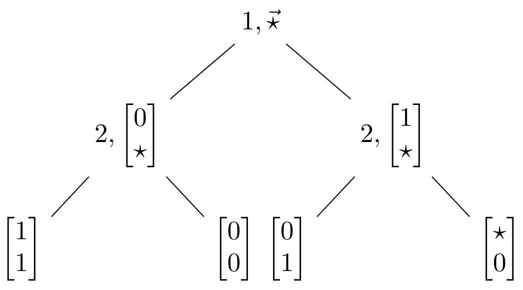
*Remark* 3.2. Let $T = (i, \Gamma, T_0, T_1)$ be a context tree over $\mathbb{T}^n$ and let $B_0 = \mathbb{B}^n[0/i]$ (all binary vectors with the $i$th bit equal to 0) and $B_1 = \mathbb{B}^n[1/i]$ then $[\![T]\!]|_{B_0} = [\![T_0]\!]|_{B_0}$ and $[\![T]\!]|_{B_1} = [\![T_1]\!]|_{B_1}$.

*Remark* 3.3. If $T = (i, \Gamma, T_0, T_1)$ is a reduced CDD over $\mathbb{T}^n$ then by no redundant branches there is some vector $b_1 \in \mathbb{B}^n$ such that $[\![T]\!](b_1) \neq [\![T_0]\!](b_1)$. By Remark 3.2 we have $b_1 \notin \mathbb{B}^n[0/i]$ and thus the $i$th bit is 1. A symmetric statement also applies to $T$ and $T_1$.

*Remark* 3.4. If $T = (i, \Gamma, T_0, T_1)$ is a CDD on $\mathbb{T}^n$, then any leaf $(\Gamma_0, w)$ in $T_0$ satisfies $(\Gamma_0)_i = 0$ and thus $(w)_i \neq 0$ and likewise for $(\Gamma_1, v)$ in $T_1$ we have $(\Gamma_1)_i = 1$ and thus $(v)_i \neq 1$.

Note that Remark 3.4 follows from proper succession and well formed leaves.

*Remark* 3.5. Just like reduced binary decision diagrams, reduced CDDs are not necessarily minimal. Sometimes one can get a smaller diagram by switching the order of the tests. For example, the following is a reduced CDD:


$1, \vec{\star}$
$2, \begin{bmatrix}0\\\star\end{bmatrix}$
$2, \begin{bmatrix}1\\\star\end{bmatrix}$
$\begin{bmatrix}1\\1\end{bmatrix}$
$\begin{bmatrix}0\\0\end{bmatrix}$
$\begin{bmatrix}0\\1\end{bmatrix}$
$\begin{bmatrix}\star\\0\end{bmatrix}$

and this is a functionally equavalent CDD:

$$\begin{tikzcd}
 & & 2, \vec{\star} \arrow[dl, dash] \arrow[dr, dash] & \\
 & 1, \begin{bmatrix} \star \\ 0 \end{bmatrix} \arrow[dl, dash] \arrow[dr, dash] & & \begin{bmatrix} \star \\ 0 \end{bmatrix} \\
\begin{bmatrix} 1 \\ 1 \end{bmatrix} & & \begin{bmatrix} 0 \\ 1 \end{bmatrix} &
\end{tikzcd}$$

**Lemma 3.6.** *Let $u, v, w \in \mathbb{T}^n$. Let $b \in \mathbb{B}^n$ be arbitrary and let $b_0 = b[0/i]$ and let $b_1 = b[1/i]$ for some $i \in [n]$. If*

$$b_0 w = b_0 u$$
$$b_1 v = b_1 u$$
$$b_k v \neq b_k w \text{ for } k = 0 \text{ or } 1$$

*then $b_k v$ and $b_k w$ differ only on the $i$th bit.*

*Proof.* We simply observe that

$b_1 u$ and $b_0 u$ differ by at most the $i$th bit

$b_1 v$ and $b_0 w$ differ by at most the $i$th bit because $b_1 u = b_1 v, b_0 u = b_0 w$

$b_1 v$ and $b_1 w$ differ by at most the $i$th bit

$b_0 v$ and $b_0 w$ differ by at most the $i$th bit.

By "differ by at most the $i$th bit" we mean it cannot be the case e.g. that $b_0 v, b_0 w$ differ by the $i + 1$th bit (or any other bit that is not $i$ for that matter). Thus since $b_k w \neq b_k v$ they must differ by exactly the $i$th bit. □

**Lemma 3.7.** *Let $T = (i, \Gamma, T_0, T_1)$ be a reduced CDD over $\mathbb{T}^n$. Let $u \in \mathbb{T}^n$ be arbitrary. Let $b \in \mathbb{B}^n$ be arbitrary and let $b_0 = b[0/i]$ and let $b_1 = b[1/i]$. Suppose that $k = 0$ or $k = 1$ and*

$$\llbracket T \rrbracket (b_0) = b_0 u$$
$$\llbracket T \rrbracket (b_1) = b_1 u$$
$$\llparenthesis T \rrparenthesis (b_0)(b_k) \neq \llparenthesis T \rrparenthesis (b_1)(b_k)$$

*then if $k = 0$ we have $(\llparenthesis T \rrparenthesis (b_0))_i = 1$ otherwise if $k = 1$ we have $(\llparenthesis T \rrparenthesis (b_1))_i = 0$.*

*Proof.* We'll give a direct proof. We start by noting that we can apply Lemma 3.6 to get that $\llparenthesis T \rrparenthesis (b_0)(b_k)$ and $\llparenthesis T \rrparenthesis (b_1)(b_k)$ differ by exactly the $i$th bit. Let $w = \llparenthesis T \rrparenthesis (b_0)$ and $v = \llparenthesis T \rrparenthesis (b_1)$. We note that by Remark 3.4 we must have $(w)_i \neq 0$ and $(v)_i \neq 1$. Now suppose $k = 0$. Then the $i$th bit of $b_0 v$ is a 0 since $(v)_i \neq 1$, so intuitively $(v)_i$ cannot "modify" the 0 on the $i$th bit of $b_0$. Thus in order for $b_0 v$ to differ from $b_0 w$ in the $i$th bit, $w$ *has* to "modify" the $i$th bit which means $(w)_i = 1$. Now for the $k = 1$ case, we use an analogous argument to show that $v$ must modify the $i$th bit of $b_1 v$ so that it differs from $b_1 w$ (whose $i$th bit must be 1 since $(w)_i \neq 0$). Thus $(v)_i = 0$ which completes the proof. □

**Lemma 3.8.** *Let $T = (i, \Gamma, T_0, T_1), T'$ be reduced CDDs over $\mathbb{T}^n$ such that $T'$ does not test the $i$th bit (meaning that in nowhere in $T'$ is there a node of the form $(i, \Gamma', T'_0, T'_1)$ which tests the $i$th bit), then we have $\llbracket T \rrbracket|_{\langle C(T) \rangle} \neq \llbracket T' \rrbracket|_{\langle C(T) \rangle}$.*

*Proof.* Let $b \in \mathbb{B}^n$ and $b_0 = b[0/i]$ and $b_1 = b[1/i]$ be such that $\llbracket T_0 \rrbracket (b_0) \neq \llbracket T_1 \rrbracket (b_0)$ and $\llbracket T_0 \rrbracket (b_1) \neq \llbracket T_1 \rrbracket (b_1)$. Let $u = \llparenthesis T' \rrparenthesis (b_0)$. We observe that $u = \llparenthesis T' \rrparenthesis (b_1)$ as well since $T'$ doesn't test the $i$th bit. Assume for contradiction that $\llbracket T \rrbracket|_{\langle C(T) \rangle} = \llbracket T' \rrbracket|_{\langle C(T) \rangle}$, then $\llbracket T \rrbracket (b_0) = b_0 u$ and $\llbracket T \rrbracket (b_1) = b_1 u$. Let $w = \llparenthesis T \rrparenthesis (b_0)$ and $v = \llparenthesis T \rrparenthesis (b_1)$. We thus have

$$b_0 w \neq b_0 v$$
$$b_1 w \neq b_1 v$$

which allows us to apply Lemma 3.7 to conclude that $(w)_i = 1$ and $(v)_i = 0$. Now we derive

$$b_0 w = b_0 u \Rightarrow (u)_i = 1 \text{ since } (w)_i = 1$$
$$b_1 v = b_1 u \Rightarrow (u)_i = 0 \text{ since } (v)_i = 0$$

which is a contradiction.

□

**Lemma 3.9.** *Let $T$ and $T'$ be reduced CDDs over $\mathbb{T}^n$ such that $C(T) = C(T')$ and $[\![T]\!]|_{\langle C(T)\rangle} = [\![T']\!]|_{\langle C(T)\rangle}$ then $T, T'$ are either both leaves or both branches.*

*Proof.* Our proof is by contradiction. Suppose without loss of generality that $T = (i, \Gamma, T_0, T_1)$ and $T' = (\Gamma', u)$ and $[\![T]\!]|_{\langle C(T)\rangle} = [\![T']\!]|_{\langle C(T)\rangle}$. Then $T'$ does not test the $i$th bit so by Lemma 3.8 we have

$$[\![T]\!]|_{\langle\Gamma\rangle} \neq [\![T']\!]|_{\langle\Gamma\rangle}$$
$$[\![T]\!]|_{\langle C(T)\rangle} \neq [\![T']\!]|_{\langle C(T)\rangle}$$

which is a contradiction.

□

**Lemma 3.10.** *Given two reduced CDDs on $\mathbb{T}^n$, $T = (i, \Gamma, T_0, T_1)$ and $T' = (j, \Gamma, T_0', T_1')$, if $[\![T]\!]|_{\langle\Gamma\rangle} = [\![T']\!]|_{\langle\Gamma\rangle}$ then $i = j$.*

*Proof.* For contradiction, suppose that the antecedent holds but $i \neq j$ and wlog $i < j$. By ordering, the $i$th bit is never tested in $T'$, so we can apply Lemma 3.8 and conclude $[\![T]\!]|_{\langle\Gamma\rangle} \neq [\![T']\!]|_{\langle\Gamma\rangle}$, a contradiction.

□

**Corollary 3.11** (Lemmas 3.9 and 3.10)**.** *If $T$ and $T'$ are reduced CDDs over $\mathbb{T}^n$ such that $[\![T]\!]|_{\langle C(T)\rangle} = [\![T']\!]|_{\langle C(T')\rangle}$ then $|T| = |T'|$ (where $|\cdot|$ is the size of the tree).*

**Lemma 3.12.** *The operator $\langle\cdot\rangle : \mathbb{T}^n \to 2^{\mathbb{B}^n}$ is injective.*

*Proof.* The map

$$x \in \mathbb{T} \mapsto \begin{cases} \{x\} & \text{if } x \neq \star \\ \{0,1\} & \text{otherwise} \end{cases}$$

is injective and characterizes the pointwise map of $\langle\cdot\rangle$. It follows that if $\Gamma, \Gamma' \in \mathbb{T}^n$ such that $B = \langle\Gamma\rangle = \langle\Gamma'\rangle$, then $(\Gamma)_i = (\Gamma')_i$ for all $i \in [n]$ and so $\Gamma = \Gamma'$.

□

**Theorem 3.13** (Canonicity)**.** *Given two reduced CDDs on $\mathbb{T}^n$, $T$ and $T'$, if $[\![T]\!]|_{\langle C(T)\rangle} = [\![T']\!]|_{\langle C(T')\rangle}$ then $T = T'$.*

*Proof.* Given Corollary 3.11 we can simultaneously induct on the structure of $T$ and $T'$. To start suppose they are both leaves where $T = (\Gamma, u), T' = (\Gamma', v)$. We have

$$[\![T]\!]|_{\langle C(T)\rangle} = [\![T']\!]|_{\langle C(T')\rangle}$$
$$\langle C(T)\rangle = \langle C(T')\rangle$$
$$\langle\Gamma\rangle = \langle\Gamma'\rangle$$
$$\Gamma = \Gamma'$$

which follows from Lemma 3.12. We also have for all $b \in \mathbb{B}^n$, $[\![T]\!](b) = [\![T']\!](b)$ and so $bu = bv$. Thus by Lemma 2.1 we have $u = v$ and thus $T = (\Gamma, u) = (\Gamma, v) = T'$.

Now for the inductive step let $T = (i, \Gamma, T_0, T_1)$ and $T' = (i, \Gamma, T_0', T_1')$. Let $\Gamma_0 = \Gamma[0/i]$ and $\Gamma_1 = \Gamma[1/i]$. We observe

$$[\![T_0]\!]|_{\langle\Gamma_0\rangle} = [\![T]\!]|_{\langle\Gamma_0\rangle} = [\![T']\!]|_{\langle\Gamma_0\rangle} = [\![T_0']\!]|_{\langle\Gamma_0\rangle}$$

where $[\![T]\!]|_{\langle\Gamma_0\rangle} = [\![T']\!]|_{\langle\Gamma_0\rangle}$ holds since $\langle\Gamma_0\rangle \subseteq \langle\Gamma\rangle$. Applying the inductive hypothesis on $T_0, T_0'$ we get $T_0 = T_0'$. By an analogous argument we get $[\![T_1]\!]|_{\langle\Gamma_1\rangle} = [\![T_1']\!]|_{\langle\Gamma_1\rangle}$ and so $T_1 = T_1'$. We thus have

$$T = (i, \Gamma, T_0, T_1) = (i, \Gamma, T_0', T_1') = T'.$$

□

Note we need the restriction to $\langle C(T)\rangle$ and $\langle C(T')\rangle$ in the above theorem in order to apply the inductive hypothesis. But in the case of initial CDDs which will be of most interest, this restriction won't matter, as highlighted in the following corollary.

**Corollary 3.14.** *Let $T$ and $T'$ be initial reduced CDDs over $\mathbb{T}^n$ such that $[\![T]\!] = [\![T']\!]$, then $T = T'$.*

*Proof.* We have $\langle\Gamma\rangle \subseteq \mathbb{B}^n$ by definition and $\vec{\star} = \Gamma = C(T) = C(T')$ since they are both initial. So

$$\begin{aligned}
[\![T]\!] &= [\![T']\!] \\
[\![T]\!]|_{\mathbb{B}^n} &= [\![T']\!]|_{\mathbb{B}^n} \\
[\![T]\!]|_{\langle\Gamma\rangle} &= [\![T']\!]|_{\langle\Gamma\rangle} \\
[\![T]\!]|_{\langle C(T)\rangle} &= [\![T']\!]|_{\langle C(T')\rangle}
\end{aligned}$$

which allows us the apply Theorem 3.13 to conclude $T = T'$. □

**Claim 3.15.** If $T$ is an initial reduced CDD over $\mathbb{T}^n$ and $[\![T]\!] = \mathrm{id}$, the identity on $\mathbb{B}^n$ then $T = (\vec{\star}, \vec{\star})$.

*Proof.* We observe that $(\vec{\star}, \vec{\star})$ is an initial reduced CDD over $\mathbb{T}^n$ whose interpretation gives the identity since $b\vec{\star} = b$ for all $b \in \mathbb{B}^n$. By Corollary 3.14 we have since $[\![T]\!] = [\![(\vec{\star}, \vec{\star})]\!]$ then $T = (\vec{\star}, \vec{\star})$.

□

**Claim 3.16.** If $T$ is an initial reduced CDD over $\mathbb{T}^n$ and $[\![T]\!](b) = bv$ for all $b \in \mathbb{B}^n$, then $T = (\vec{\star}, v)$.

*Proof.* We observe that $[\![(\vec{\star}, v)]\!](b) = bv$ and that $(\vec{\star}, v)$ is an initial reduced CDD. It follows by Corollary 3.14 that since $[\![T]\!] = [\![(\vec{\star}, v)]\!]$ we have $T = (\vec{\star}, v)$.

□

## 4 How to make reduced CDDs

Our goal is to define a procedure $\texttt{make-branch}(i, T_0, T_1)$ that takes two reduced CDDs $T_0, T_1$ and a test bit $i$ which is smaller than all the test bits in $T_0, T_1$ and returns the functional equivalent of $(i, \vec{\star}, T_0, T_1)$ as a reduced CDD. The basic idea is to first check whether one of $T_0, T_1$ can be "subsumed" by the other. For example let $T$ be the reduced CDD of Figure 2, then we should have $T = \texttt{make-branch}(1, T, (\vec{\star}, \vec{\star}))$ since intuitively $(\vec{\star}, \vec{\star})$ can be subsumed into the leaves of $T$. We define two smaller procedures that will be useful in defining make-branch. The procedure $\texttt{well-formed}(u, v)$ takes two vectors $u, v \in \mathbb{T}^n$ and turns them into a well-formed pair by making $(v)_i = \star$ whenever $(u)_i = (v)_i$. We also define $\texttt{merge-leaves}(i, \Gamma, u, v)$ which takes a context $\Gamma$, forms the pairs $\texttt{well-formed}(\Gamma[0/i], u)$ and $\texttt{well-formed}(\Gamma[1/i], v)$ and merges the pairs if possible. Before giving a formal specification, we start with the following definition.

**Definition 4.1** (Mergeable leaves)**.** The CDDs $(\Gamma_0, u)$ and $(\Gamma_1, v)$ (both leaves) are mergeable over $i$ if

(1) $\Gamma_0 = \Gamma_1[0/i]$ and $\Gamma_1 = \Gamma_0[1/i]$
(2) Either $u = v$ or $u, v$ differ only by the $i$th bit for which one of $(u)_i, (v)_i$ is $\star$ and the other is a constant bit

To merge a mergeable pair $(\Gamma_0, u)$ and $(\Gamma_1, v)$ over $i$ means to form the leaf $(\Gamma_0[\star/i], u+v)$. We now define formally

$$\texttt{merge-leaves}(i, \Gamma, u, v) = \begin{cases} (\Gamma, w) & : \text{ if } \texttt{well-formed}(\Gamma[0/i], u), \texttt{well-formed}(\Gamma[1/i], v) \text{ are mergeable} \\ & \quad \text{and } (\Gamma, w) \text{ is their merge} \\ (i, \Gamma, T_0, T_1) & : \text{otherwise, here } T_0 = \texttt{well-formed}(\Gamma[0/i], u), \\ & \qquad\qquad\qquad T_1 = \texttt{well-formed}(\Gamma[1/i], v) \end{cases}.$$

We can thus define a function $\texttt{merge}(i, \Gamma, T_0, T_1)$ which tries to merge $T_0$ and $T_1$ over $i$ by matching the substructures of $T_0$ and $T_1$. So for example in the base case $T_0$ and $T_1$ are leaves $(\Gamma_0, u)$ and $(\Gamma_1, v)$ in which case $\texttt{merge}(i, \Gamma, T_0, T_1) = \texttt{merge-leaves}(i, \Gamma, u, v)$. In the general case, we check the test indices if $T_0$ and $T_1$. If $T_0$ and $T_1$ both test some index $j$, we recurse simultaneously into the right and left subtrees. That is

$$\texttt{merge}(i, \Gamma, (j, \Gamma_0, T_0', T_0''), (j, \Gamma_1, T_1', T_1'')) =$$
$$(j, \Gamma, \texttt{merge}(i, \Gamma[0/j], T_0', T_1'), \texttt{merge}(i, \Gamma[1/j], T_0'', T_1'')).$$

If $T_0$ and $T_1$ test different bits, say $k$ and $j$, then recurse into the tree with the smaller index. That is (suppose wlog $k < j$)

$$\texttt{merge}(i, \Gamma, (k, \Gamma_0, T_0', T_0''), T_1) =$$

$$(k, \Gamma, (\texttt{merge}(i, \Gamma[0/k], T_0', T_1)), (\texttt{merge}(i, \Gamma[1/k], T_0'', T_1))).$$

We use the convention that the test bit of a leaf is $\infty$. This convention ensures that the merge procedure is always making a step towards completion. The $\texttt{make-branch}(i, T_0, T_1)$ procedure then operates by attempting to merge $T_0, T_1$ over $i$ and the initial context $\Gamma = \vec{\star}$. If the result of the merge is a valid CDD (specifically if proper ordering is maintained) then simply return the merge (this intuitively means that one diagram was "subsumed" by the other); otherwise, return $(i, \vec{\star}, T_0, T_1)$, with the contexts of $T_0, T_1$ corrected so $i$ remains a fixed bit. For example, let $T_0$ be the CDD of Figure 2 and let $T_1 = \star$, then one can compute that $\texttt{merge}(1, T_0, T_1) = T_0$ and thus there is no need to test the first bit.

**Example 4.2.** Consider the following

$$\texttt{make-branch}\left(1, \vec{\star}, \begin{tikzcd} & 2,\vec{\star} \arrow[dl, dash] \arrow[dr, dash] & \\ \begin{bmatrix}\star\\1\end{bmatrix} & & \begin{bmatrix}1\\0\end{bmatrix}\end{tikzcd}, \begin{tikzcd} & 2,\vec{\star} \arrow[dl, dash] \arrow[dr, dash] & \\ \begin{bmatrix}0\\1\end{bmatrix} & & \begin{bmatrix}\star\\0\end{bmatrix}\end{tikzcd}\right).$$

The first step in the merge procedure is to recurse into the two subtrees since they both test the 2nd bit. We start with the leftmost recursive call to compute

$$\texttt{merge}\left(1, \begin{bmatrix}\star\\0\end{bmatrix}, \begin{bmatrix}\star\\1\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}\right) = \begin{bmatrix}0\\1\end{bmatrix}$$

and

$$\texttt{merge}\left(1, \begin{bmatrix}\star\\1\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}\star\\0\end{bmatrix}\right) = \begin{bmatrix}1\\0\end{bmatrix}.$$

Thus we have

$$\texttt{merge}\left(1, \vec{\star}, \begin{tikzcd} & 2,\vec{\star} \arrow[dl, dash] \arrow[dr, dash] & \\ \begin{bmatrix}\star\\1\end{bmatrix} & & \begin{bmatrix}1\\0\end{bmatrix}\end{tikzcd}, \begin{tikzcd} & 2,\vec{\star} \arrow[dl, dash] \arrow[dr, dash] & \\ \begin{bmatrix}0\\1\end{bmatrix} & & \begin{bmatrix}\star\\0\end{bmatrix}\end{tikzcd}\right) = \begin{tikzcd} & 2,\vec{\star} \arrow[dl, dash] \arrow[dr, dash] & \\ \begin{bmatrix}0\\1\end{bmatrix} & & \begin{bmatrix}1\\0\end{bmatrix}\end{tikzcd}.$$

Since the tests on this final tree are properly ordered, it will be returned as the result of the `make-branch` procedure.

## 4.1 Remarks about the `merge` procedure

In all these remarks $T_0$ and $T_1$ be two reduced CDDs and $i$ be a bit not already tested in $T_0, T_1$.

*Remark* 4.3. Let $u$ be a leaf in either $T_0$ or $T_1$ then at some point in $\texttt{merge}(i, \vec{\star}, T_0, T_1)$ a call is made to `merge-leaves` where one of the leaves is $u$.

*Remark* 4.4. Let $(\Gamma_0, u)$ be a leaf node in $T_0$ and let $(\Gamma_1, v)$ be a leaf in $T_1$. Then $\texttt{merge-leaves}(i, \Gamma, u, v)$ is not be called in $\texttt{merge}(i, \vec{\star}, T_0, T_1)$ iff $\Gamma_0, \Gamma_1$ differ by some bit $j$ such that $(\Gamma_0)_j, (\Gamma_1)_j \in \{0, 1\}$.

Remark 4.4 follows from the observation that the `merge` procedure only splits the the comparisons it makes when both subtrees test the same bit. So if $\texttt{merge-leaves}(i, \Gamma, u, v)$ is not called then at some point the merge procedure can to a test bit both subtrees tested and $u, v$ were in separate branches in the recursive subcall. Hence they must have been in opposite subtrees.

## 4.2 Proof that the `merge` procedure works

Our goal is to prove that the merge procedure fails to fully merge the two trees iff the new CDD $(i, \vec{\star}, T_0, T_1)$ is already reduced.

**Theorem 4.5.** *Let $T_0$ and $T_1$ be two reduced CDDs and $i$ be a bit not already tested in $T_0, T_1$. Then $(i, \vec{\star}, T_0, T_1)$ is a reduced CDD iff `merge`$(i, \vec{\star}, T_0, T_1)$ fails to return a valid CDD.*

*Proof.* We start with the "only if" direction. Suppose that $(i, \star, T_0, T_1)$ is a reduced CDD. For purposes of contradiction, suppose that `merge`$(i, \vec{\star}, T_0, T_1)$ succeeds in merging $T_0, T_1$. Let $(b_0, b_1)$ be the certificate of reducedness for the root node of $(i, \vec{\star}, T_0, T_1)$. We define

$$u = (\!|T_0|\!)(b_0)$$
$$v = (\!|T_1|\!)(b_1)$$

and let $\Gamma_0, \Gamma_1$ be their respective contexts in the original $T_0, T_1$. Since $b_0, b_1$ differ only on the $i$th bit, we cannot have their respective contexts differ by constant bits. Thus by Remark 4.4 we get that `merge-leaves`$(i, \Gamma, u, v)$ is called at some point in the merge. However we note that since $b_0u \neq b_0v$ and $b_1u \neq b_1v$ by the assumption that $(i, \vec{\star}, T_0, T_1)$ is a reduced CDD, we cannot merge $u, v$. Thus `merge-leaves`$(i, \Gamma, u, v)$ fails to merge the leaves which means `merge`$(i, \vec{\star}, T_0, T_1)$ would fail at merging $T_0$ and $T_1$.

Now for the other direction suppose the `merge`$(i, \vec{\star}, T_0, T_1)$ fails. Then, the basic idea is that we can produce a certificate of reducedness of $(i, \vec{\star}, T_0, T_1)$ from the failed merge. Specifically, if `merge`$(i, \vec{\star}, T_0, T_1)$ fails then at some point a recursive call to `merge-leaves`$(i, \Gamma, u, v)$ fails. We use $u$ and $v$ to construct a vector $c \in \mathbb{B}^n$ on which we base the certificate of reducedness. We construct $c$ by the following

```
1: $c \leftarrow u$
2: For $k = 1, ..., n$ :
3:     If $(u)_k \neq (v)_k$ and $\star \in \{(u)_k, (v)_k\}$ then set $(c)_k$ to the one bit in $\{0, 1, \star\} \setminus \{(u)_k, (v)_k\}$
4:     Else if $(u)_k = \star$ then set $(c)_k$ to $(\Gamma)_k$ if $(\Gamma)_k \neq \star$, otherwise set it to 0.
5: $b_0 \leftarrow c[0/i], b_1 \leftarrow c[1/i]$
```

The claim is that $(b_0, b_1)$ is a certificate of reducedness for $(i, \vec{\star}, T_0, T_1)$. We thus need to verify

$$[\![T_0]\!](b_0) \neq [\![T_1]\!](b_0)$$
$$[\![T_0]\!](b_1) \neq [\![T_1]\!](b_1).$$

We start by showing that $(\!|T_0|\!)(b_0) = u$ and $(\!|T_1|\!)(b_0) = v$ (the analogous result will automatically hold for $b_1$ since $b_0, b_1$ differ only by the $i$th bit). We note the context $\Gamma$ encodes exactly the vectors which filter into either $u$ or $v$. By our construction we have $b_0, b_1 \in \langle \Gamma \rangle$, hence $(\!|T_0|\!)(b_0) = u$ and $(\!|T_1|\!)(b_0) = v$. Hence it is sufficient to show that

$$b_0u \neq b_0v$$
$$b_1u \neq b_1v.$$

By the failure of `merge-leaves`$(i, \Gamma, u, v)$ we know that $u, v$ differ by at least 1 bit. There are two cases to consider.
<u>Case 1</u>: $u, v$ differ only by the $i$th bit.
In this case, we know that both bits must be constant since otherwise `merge-leaves` would have succeeded. We thus simply note since they are constant bits

$$(b_0u)_i = (u)_i \neq (v)_i = (b_0v)_i$$

and the same holds for $b_1$. Thus $(b_0, b_1)$ is a valid certificate.

<u>Case 2</u>: $u, v$ differ only by some non-$i$th bit.

Let $j \neq i$ be a bit on which they differ. If $(u)_j$ and $(v)_j$ are both constant bits we apply the same argument as in Case 1 to get that $b_0 u \neq b_0 v$. Otherwise suppose one of $(u)_j, (v)_j$ is $\star$. Without loss of generality suppose $(u)_j = \star$. By our selection of $b_0, b_1$ we have $(b_0)_j, (b_1)_j$ are constant bits differing from both $(u)_j$ and $(v)_j$ so

$$(b_0 v)_j = (v)_j \neq (b_0)_j = (b_0 u)_j.$$

Thus $(b_0, b_1)$ is a valid certificate. It follows then that if `merge`$(i, \vec{\star}, T_0, T_1)$ fails then $(i, \star, T_0, T_1)$ is a reduced CDD.

$\square$

## References

[AFG$^+$14] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. Netkat: Semantic foundations for networks. *SIGPLAN Not.*, 49(1):113–126, January 2014.

[Bry86] Randal E Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.

[SEFG15] Steffen Smolka, Spiridon Eliopoulos, Nate Foster, and Arjun Guha. A fast compiler for netkat. *ACM SIGPLAN Notices*, 50(9):328–341, 2015.