# Symbolic Simulation, Model Checking and Abstraction with Partially Ordered Boolean Functional Vectors

Amit Goel[1] and Randal E. Bryant[2]

[1] Dept. of ECE, Carnegie Mellon University. agoel@ece.cmu.edu
[2] Computer Science Dept., Carnegie Mellon University. Randy.Bryant@cs.cmu.edu

**Abstract.** Boolean Functional Vectors (BFVs) are a symbolic representation for sets of bit-vectors that can be exponentially more compact than the corresponding characteristic functions with BDDs. Additionally, BFVs are the natural representation of bit-vector sets for Symbolic Simulation. Recently, we developed set manipulation algorithms for canonical BFVs by interpreting them as totally ordered selections. In this paper we generalize BFVs by defining them with respect to a partial order. We show that partially ordered BFVs can serve as abstractions for bit-vector sets and can be used to compute over-approximations in reachability analysis. In the special case when the underlying graph of the partial order is a forest, we can efficiently compute an abstract interpretation in a symbolic simulation framework. We present circuit examples where we leverage the exponential gap in the representations and inherent structure in the state-space to demonstrate the usefulness of Partially Ordered Boolean Functional Vectors.

## 1 Introduction

Symbolic Model Checking and related state-space exploration techniques usually represent sets of states with a characteristic function, often using BDDs to encode the characteristic function. A characteristic function is essentially a test for membership, returning a one if and only if the input is in the set. Alternatively, bit-vector sets can be represented symbolically by Boolean Functional Vectors (BFVs) [4] which map bit-vectors to bit-vectors; the set represented is the range of this mapping. Hence, characteristic functions serve as set acceptors while Boolean Functional Vectors are set generators.

We are interested in the Boolean Functional Vector representation for two main reasons:

- They can be exponentially more compact than the corresponding characteristic function when using BDDs as the underlying representation.
- Boolean Functional Vectors are the natural representation for Symbolic Simulation.

However, there are two drawbacks that have prevented the representation from being used more often. Firstly, the representation is not canonical, *per se*. Secondly, until recently there were no set manipulation algorithms for this representation.

The representation can be made canonical by placing certain restrictions on the representation [4, 10]. Recently, we presented algorithms for set union, intersection and projection based on our interpretation of the canonical BFV as an ordered selection [5], thus enabling Symbolic Simulation based Model Checking.

In this paper, we generalize the framework by defining Boolean Functional Vectors with respect to a *partial order*. We show that Partially Ordered Boolean Functional Vectors serve as abstractions for bit-vector sets. If the underlying graph of the partial order is a forest, the partially ordered BFVs form a lattice and the abstraction defines a Galois connection between the BFV space and the concrete space of bit-vector sets.

The partial order allows us to selectively constrain some variables with respect to each other while ignoring constraints between unrelated variables. This is in contrast to most other approaches to abstraction where some of the variables are discarded and all constraints between the remaining variables are retained.

We present algorithms for (abstract) set manipulation and show how to use these algorithms for image computation. When the underlying graph is a forest, our method is a complete abstract interpretation. We then present two examples where we leverage the exponential gap between BFVs and characteristic functions as well as inherent structure in the state-space to enable efficient verification.

### 1.1 Related Work

Boolean Functional Vectors were originally used in [4] for image computation where symbolic simulation was used for next state computation but all other set manipulation operations were performed by first converting to characteristic functions, performing the necessary operations and converting back to Boolean Functional Vectors for the next iteration. The canonical BFVs were described in [4, 10]. An efficient algorithm to obtain a BFV from a characteristic function (parameterization) was presented in [1].

The most prominent use of Boolean Functional Vectors is in Symbolic Trajectory Evaluation (STE) [2] which is a symbolic simulation based bounded model checking approach. In [3], Chou developed a framework for symbolic ternary simulation based abstract interpretation. STE has recently been extended to Generalized Symbolic Trajectory Evaluation (GSTE) [11] to enable model checking of all $\omega$-regular properties. This was made possible by using a *reparameterization* algorithm to convert a given BFV into a canonical BFV. This algorithm, presented in [12], can be seen as a special case of the projection algorithm presented here. GSTE also uses a *ternary abstraction* in which each state element is classified as concrete or ternary. A choice for a concrete element is represented by a state variable whereas a choice for a ternary element is represented by the

ternary value $X$. In this scheme, all relations between concrete values are captured and the ternary variables can only be constrained by the concrete values. No relations between ternary values are captured. The abstraction by partial order BFVs presented here subsumes ternary abstraction. We can model the ternary abstraction by a partial order in which the concrete variables are related to each other, all concrete variables precede all ternary variables, and the ternary variables are unrelated to each other.

The conjunctive canonical decomposition [7] is a symbolic representation closely related to Boolean Functional Vectors. This correspondence was explored in our earlier work [5]. The theory we develop here can be applied, with suitable modifications, to conjunctive decompositions as well.

In [6], the authors perform abstraction using overlapping projections. These projections correspond to the chains in our partial orders. The difference is primarily this: with overlapping projections, there is a set constraint for each projection while with partially ordered BFVs, there is an evaluation function for each state element. With overlapping projections, the constraints between variables occurring together in multiple projections will be repeated in each of these projections. With partially ordered BFVs, each variable is constrained only once.

### 1.2 Preliminaries

An n-length bit-vector $\vec{X}$ is a mapping from the set of indices $\mathcal{I} = \{1, \ldots, n\}$ to the set of Boolean values $\mathcal{B} = \{0, 1\}$. The set of all n-length bit vectors is noted $[\mathcal{I} \mapsto \mathcal{B}]$. Let $\preceq$ be a partial order on $\mathcal{I}$ and $\prec$ the associated strict order. The set of ancestors for index $i$ is given by $ancestors(i) = \{j | j \prec i\}$.

Let $\prec_{min}$ be the minimum relation whose transitive, reflexive closure is $\preceq$. When the graph $(\mathcal{I}, \prec_{min})$ is a forest, we say that $i$ is a root if it has no ancestors, and we define $parent(i)$ when $i$ is not a root by $parent(i) \prec_{min} i$. The set of children of an index is then given by $children(i) = \{j | i = parent(j)\}$.

In the following, $\vec{V} = \langle v_1, \ldots, v_n \rangle$ represents a vector of Boolean variables, $\vec{X}$ and $\vec{Y}$ represent bit-vectors and $\vec{F}, \vec{G}$ and $\vec{H}$ represent Boolean Functional Vectors.

## 2 Partially Ordered Boolean Functional Vectors

**Definition 1.** *An $(\mathcal{I}, \preceq)$-BFV is a vector of Boolean functions $\vec{F} = \langle f_1, \ldots, f_n \rangle$ such that for all indices $i$:*

$$f_i(\vec{V}) = f_i^1(\vec{V}) + f_i^c(\vec{V}) \cdot v_i$$

*where $f_i^1$ and $f_i^c$ are mutually exclusive Boolean functions of the ancestors of $i$:*

$$f_i^1(\vec{X}) \cdot f_i^c(\vec{X}) = 0$$
$$(\forall j \in ancestors(i). f_j(\vec{X}) = f_j(\vec{Y})) \Rightarrow (f_i^1(\vec{X}) = f_i^1(\vec{Y})) \wedge (f_i^c(\vec{X}) = f_i^c(\vec{Y}))$$

*for all $\vec{X}, \vec{Y} \in [\mathcal{I} \mapsto \mathcal{B}]$.*

An $(\mathcal{I}, \preceq)$-BFV is to be interpreted as an ordered selection. The variable $v_i$ represents an input *choice* for the $i$-th bit while $f_i$ is the corresponding *selection*. The definition requires that the selection for the $i$-th bit is constrained only by the selections for its ancestors. We will refer to $f_i^1$ and $f_i^c$ as the *forced-to-one* and *free-choice* conditions for the $i$-th bit. Additionally, we define the *forced-to-zero* condition $f_i^0 = \neg(f_i^1 + f_i^c)$.

**Definition 2.** *We define the space $\mathcal{F}_{(\mathcal{I},\preceq)}$ to include all $(\mathcal{I}, \preceq)$-BFVs and extend it to $\mathcal{F}^{\perp}_{(\mathcal{I},\preceq)}$ by including a bottom element to represent the empty set:*

$$\mathcal{F}_{(\mathcal{I},\preceq)} = \{\vec{F} | \vec{F} \text{ is an } (\mathcal{I}, \preceq)\text{-BFV}\}$$
$$\mathcal{F}^{\perp}_{(\mathcal{I},\preceq)} = \mathcal{F}_{(\mathcal{I},\preceq)} \cup \{\perp\}$$

We now define a concretization function which maps an $(\mathcal{I}, \preceq)$-BFV to its range and the bottom element to the empty set.

**Definition 3.** *The concretization function $\gamma : \mathcal{F}^{\perp}_{(\mathcal{I},\preceq)} \mapsto \mathcal{P}([\mathcal{I} \mapsto \mathcal{B}])$ is given by:*

$$\gamma(\perp) = \emptyset$$
$$\gamma(\vec{F}) = \{\vec{X} \in [\mathcal{I} \mapsto \mathcal{B}] | \exists \vec{Y} \in [\mathcal{I} \mapsto \mathcal{B}].\vec{F}(\vec{Y}) = \vec{X}\} \text{ for all } \vec{F} \in \mathcal{F}_{(\mathcal{I},\preceq)}$$

We say that $\vec{F}$ abstracts a bit vector set $S$ if $\gamma(\vec{F}) \supseteq S$ and that $\vec{F}$ represents $S$ if $\gamma(\vec{F}) = S$. Not all bit-vector sets $S$ can be represented in $\mathcal{F}^{\perp}_{(\mathcal{I},\preceq)}$but our definition ensures that if there is a representation for $S$, then it is unique. Additionally, vectors in the range of an $(\mathcal{I}, \preceq)$-BFV are mapped to themselves:

**Theorem 1.** *Given $\vec{F}, \vec{G} \in \mathcal{F}_{(\mathcal{I},\preceq)}$ and $\vec{X} \in [\mathcal{I} \mapsto \mathcal{B}]$:*

$$\gamma(\vec{F}) = \gamma(\vec{G}) \Leftrightarrow \forall \vec{X} \in [\mathcal{I} \mapsto \mathcal{B}].F(\vec{X}) = G(\vec{X})$$
$$\vec{X} \in \gamma(\vec{F}) \Leftrightarrow \vec{F}(\vec{X}) = \vec{X}$$

The above theorem gives us a procedure to obtain the characteristic function $\chi_{\gamma(\vec{F})}$ for a set from its BFV $\vec{F}$. Recall that $\chi_{\gamma(\vec{F})}(\vec{X}) = 1$ if and only if $\vec{X} \in \gamma(\vec{F})$. From Theorem 1 it follows that $\chi_{\gamma(\vec{F})}(\vec{X}) = 1$ if and only if $F(\vec{X}) = \vec{X}$. Hence, we can derive:

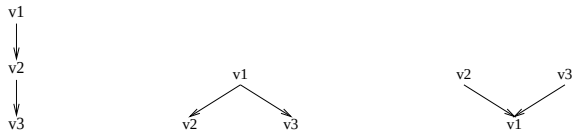
$$\chi_{\gamma(\vec{F})}(\vec{V}) = \prod_{i \in \mathcal{I}} v_i \leftrightarrow f_i(\vec{V})$$

We observe that $\langle v_1 \leftrightarrow f_i, \ldots, v_n \leftrightarrow f_n \rangle$ is a canonical conjunctive decomposition [7] for $\gamma(\vec{F})$. The theory we develop in this paper for Boolean Functional Vectors applies, with suitable modifications, to conjunctive decompositions as well.

We now define a partial ordering $\sqsubseteq$ on $\mathcal{F}^{\perp}_{(\mathcal{I},\preceq)}$ by lifting the subset ordering $\subseteq$ on bit-vector sets.

**Definition 4.** *The partial ordering $\sqsubseteq$ is defined on $\mathcal{F}^{\perp}_{(\mathcal{I},\preceq)}$ by:*

$$\vec{F} \sqsubseteq \vec{G} \Leftrightarrow \gamma(\vec{F}) \subseteq \gamma(\vec{G})$$




(a) $\preceq_1$ (b) $\preceq_2$ (c) $\preceq_3$

**Fig. 1.** Partial Orders used in Example 1

*Example 1.* Given $S = \{000, 100, 111\}$, $\mathcal{I} = 1, 2, 3$ and $\vec{V} = \langle v_1, v_2, v_3 \rangle$, let $\preceq_1$, $\preceq_2$ and $\preceq_3$ be the reflexive transitive closures of the partial orders depicted in Figures 1(a),1(b) and 1(c) respectively.

The $(\mathcal{I}, \preceq_1)$-BFV $\vec{F}_1 = \langle v_1, v_1 \cdot v_2, v_1 \cdot v_2 \rangle$ represents $S$, i.e., $\gamma(\vec{F}_1) = S$.

The $(\mathcal{I}, \preceq_2)$-BFV $\vec{F}_2 = \langle v_1, v_1 \cdot v_2, v_1 \cdot v_3 \rangle$ abstracts $S$. We have $\gamma(\vec{F}_2) = \{000, 100, 101, 110, 111\}$ which is a superset of $S$. Moreover, $\vec{F}_2$ is the minimum abstraction for $S$ in $\mathcal{F}^{\perp}_{(\mathcal{I}, \preceq)}$.

The $(\mathcal{I}, \preceq_3)$-BFVs $\vec{G} = \langle (v_2 \cdot v_3) + (\neg v_2 \cdot \neg v_3) \cdot v_1, v_2, v_3 \rangle$ and $\vec{H} = \langle (v_2 + v_3) + (\neg v_2 \cdot \neg v_3) \cdot v_1, v_2, v_3 \rangle$ abstract $S$, since $\gamma(\vec{G}) = \{000, 001, 010, 100, 111\}$ and $\gamma(\vec{H}) = \{000, 101, 110, 100, 111\}$. Note that $\vec{G}$ and $\vec{H}$ are unrelated minimal abstractions.

**Lemma 1.** *If $(\mathcal{I}, \prec_{min})$ is a forest then there is a minimum abstraction $\alpha(S)$ in $\mathcal{F}^{\perp}_{(\mathcal{I}, \preceq)}$ for every set $S \subseteq \mathcal{P}([\mathcal{I} \mapsto \mathcal{B}])$.*

**Theorem 2.** *If $(\mathcal{I}, \prec_{min})$ is a forest then:*

1. *$(\mathcal{F}^{\perp}_{(\mathcal{I}, \preceq)}, \sqsubseteq)$ forms a complete lattice. For $\vec{F}, \vec{G} \in \mathcal{F}^{\perp}_{(\mathcal{I}, \preceq)}$, the least upper bound and greatest lower bound are given by:*

$$\vec{F} \sqcup \vec{G} = \alpha(\gamma(\vec{F}) \cup \gamma(\vec{G}))$$
$$\vec{F} \sqcap \vec{G} = \alpha(\gamma(\vec{F}) \cap \gamma(\vec{G}))$$

2. *The pair of adjoined functions $(\alpha, \gamma)$ forms a* Galois connection *between the concrete space $(\mathcal{P}([\mathcal{I} \mapsto \mathcal{B}]), \subseteq)$ and the abstract space $(\mathcal{F}^{\perp}_{(\mathcal{I}, \preceq)}, \sqsubseteq)$. For all $S \in \mathcal{P}([\mathcal{I} \mapsto \mathcal{B}])$ and $\vec{F} \in \mathcal{F}^{\perp}_{(\mathcal{I}, \preceq)}$:*

$$S \subseteq \gamma(\alpha(S))$$
$$\vec{F} = \alpha(\gamma(\vec{F}))$$

Note that concretization does not lose any information. Furthermore, if $(\mathcal{I}, \preceq)$ is a total order, then the abstract space $\mathcal{F}^{\perp}_{(\mathcal{I}, \preceq)}$ is isomorphic to the concrete space $\mathcal{P}([\mathcal{I} \mapsto \mathcal{B}])$ and no information is lost in abstraction, either:

**Theorem 3.** *If $\mathcal{F}^{\perp}_{(\mathcal{I}, \preceq)}$ is totally ordered, then for all $S \in \mathcal{P}([\mathcal{I} \mapsto \mathcal{B}])$:*

$$S = \gamma(\alpha(S))$$

## 3 Algorithms

In this section, we assume that $(\mathcal{I}, \prec_{min})$ is a forest and present algorithms for computing the least upper bound and greatest lower bound of $(\mathcal{I}, \preceq)$-BFVs $\vec{F}$ and $\vec{G}$ to give us abstractions for set union and intersection respectively. We also present an algorithm to compute the minimum abstraction for projection.

The set union and intersection algorithms are modified from [5] to take into account that $\preceq$ is not necessarily a total order. The algorithm for projection is new and is more efficient than computing the union of the cofactors because of fewer intermediate computations.

The algorithms presented here can be modified for the case when $(\mathcal{I}, \prec_{min})$ is not a forest. The modifications are required to account for cases when there are conflicting constraints from unrelated ancestors for the selection of a bit. In such a case, we could obtain a minimal abstraction by choosing the constraint from one of these ancestors, or an upper bound of these minimal abstraction by ignoring all constraints in case of a conflict.

### 3.1 Set Union

In selecting a vector from the union of two sets, we could select the vector from either of the operands. If we select the bits one at a time (in an order consistent with $\preceq$), initially we can make a selection from either set. In this scenario, the bit being selected is forced-to-zero(one) if and only if it is forced-to-zero(one) in both sets. We can continue to select from either set, until we commit ourselves to one of the operands. This happens when the bit being selected is forced-to-zero(one) in one of the operand sets and we select the opposite value. From then on, we can exclude the operand set that disagrees with our selection.

Given $(\mathcal{I}, \preceq)$-BFVs $\vec{F}$ and $\vec{G}$, we define conditions $f_i^x$ and $g_i^x$ to indicate when $\vec{F}$ and $\vec{G}$ can be excluded from the selection for the union. If $i$ is a root, then:

$$f_i^x = \mathbf{0}$$
$$g_i^x = \mathbf{0}$$

Otherwise, let $j = parent(i)$:

$$f_i^x = f_j^x + f_j^0 \cdot h_j + f_j^1 \cdot \neg h_j$$
$$g_i^x = g_j^x + g_j^0 \cdot h_j + g_j^1 \cdot \neg h_j$$

We now define $\vec{H}$ so that bit $i$ is forced-to-zero(one) in the union if and only if it is forced-to-zero(one) in the non-excluded sets:

$$h_i^0 = f_i^0 \cdot g_i^0 + f_i^0 \cdot g_i^x + f_i^x \cdot g_i^0$$
$$h_i^1 = f_i^1 \cdot g_i^1 + f_i^1 \cdot g_i^x + f_i^x \cdot g_i^1$$

**Theorem 4.** *Given $(\mathcal{I}, \preceq)$-BFVs $\vec{F}$ and $\vec{G}$, let $\vec{H}$ be defined as above. Then:*

$$\vec{H} = \vec{F} \sqcup \vec{G}$$

From Theorems 2 and 4, it follows that our algorithm computes an over-approximation of the corresponding set union, i.e., $\gamma(\vec{H}) \supseteq \gamma(\vec{F}) \cup \gamma(\vec{G})$.

### 3.2 Set Intersection

A vector can be selected from the intersection of two sets only if it can be selected in both sets. Hence, we can make selections for bits only if both operands agree on the selection. The selection is forced-to-zero(one) in the intersection if it is forced-to-zero(one) in either operand. We must be careful, however, to avoid conflicts that can occur when, for some bit, the selection in one operand is forced-to-zero while the selection is forced-to-one in the other operand because of the selections made for the ancestors of the bit in question.

Given $(\mathcal{I}, \preceq)$-BFVs $\vec{F}$ and $\vec{G}$, we define elimination conditions $e_i$ to indicate conflicting selections while computing the intersection:

$$e_i = \sum_{j \in children(i)} (f_j^0 \cdot g_j^1 + f_j^1 \cdot g_j^0 + \forall v_j . e_j)$$

If there is no possible selection for some bit, then the intersection is empty:

$$(\exists i . e_i = \mathbf{1}) \Rightarrow H = \bot$$

Otherwise, we obtain a vector $\vec{K}$ by eliminating the conflicts:

$$k_i^0 = f_i^0 + g_i^0 + e_{i|v_i \leftarrow 1}$$
$$k_i^1 = f_i^1 + g_i^1 + e_{i|v_i \leftarrow 0}$$

We then obtain $\vec{H}$ by normalizing $\vec{K}$ by propagating the selection constraints (introduced by the elimination) downstream:

$$h_i^0 = k^0_{i|v_j \leftarrow h_j, \forall j \in ancestors(i)}$$
$$h_i^1 = k^1_{i|v_j \leftarrow h_j, \forall j \in ancestors(i)}$$

**Theorem 5.** *Given $(\mathcal{I}, \preceq)$-BFVs $\vec{F}$ and $\vec{G}$, define $\vec{H}$ as above. Then:*

$$\vec{H} = \vec{F} \sqcap \vec{G}$$

As with set union, it follows from Theorems 5 and 2 that $\vec{H}$ is an over-approximation of the corresponding set intersection, i.e., $\gamma(\vec{H}) \supseteq \gamma(\vec{F}) \cap \gamma(\vec{G})$.

### 3.3 Projection

Given $\mathcal{I}' \subseteq \mathcal{I}$, the (existential) projection of a set $S \in \mathcal{P}([\mathcal{I} \mapsto \mathcal{B}])$ on $\mathcal{I}'$ is:

$$proj_{\mathcal{I}'}(S) = \{\vec{X} \in [\mathcal{I}' \mapsto \mathcal{B}] | \exists \vec{Y} \in S . \forall i \in \mathcal{I}' . \vec{Y}(i) = \vec{X}(i)\}$$

In selecting a vector for the projection, we can select an $\vec{X} \in [\mathcal{I}' \mapsto \mathcal{B}]$ as long as there is some selection for the bits in $(\mathcal{I} \setminus \mathcal{I}')$ that can extend $\vec{X}$ to some vector $\vec{Y}$ in $S$. The projection loses information about the relation between the bits retained and the bits projected out. We capture this information with the

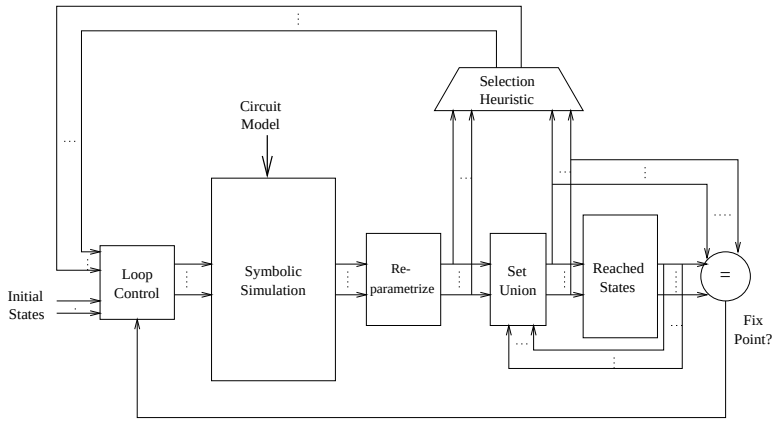



**Fig. 2.** Symbolic Reachability Analysis using Symbolic Simulation with Boolean Functional Vectors

don't-care conditions $f_i^{dc}$. If $i$ is a root, then $f_i^{dc} = \mathbf{0}$. Otherwise, let $j = parent(i)$ in:

$$\begin{aligned} f_i^{dc} &= f_j^{dc} && \text{if } j \notin \mathcal{I}', \\ &= f_j^{dc} + f_j^0 \cdot h_j + f_j^1 \cdot \neg h_j && \text{otherwise.} \end{aligned}$$

The abstract projection $\vec{H}$ is now defined so that a bit is forced-to-zero(one) if and only if it is forced-to-zero(one) in the care space irrespective of the values of the projected out bits. Let $V'' = \{v_i | i \in (\mathcal{I} \setminus \mathcal{I}')\}$. Then, for $i \in \mathcal{I}'$:

$$\begin{aligned} h_i^0 &= \forall V''.(f_i^0 + f_i^{dc}) \\ h_i^1 &= \forall V''.(f_i^1 + f_i^{dc}) \end{aligned}$$

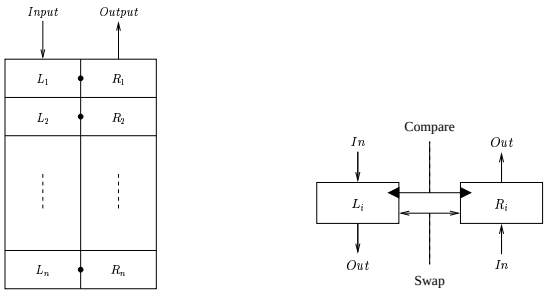
**Theorem 6.** *Given an $(\mathcal{I}, \preceq)$-BFV $\vec{F}$ and $\mathcal{I}' \subseteq \mathcal{I}$, let $\vec{H}$ be defined as above. Then:*

$$\vec{H} = \alpha(proj_{\mathcal{I}'}(\gamma(\vec{F})))$$

## 4 Symbolic Simulation, Model Checking and Abstraction

Using the algorithms from the previous section, we can perform the computations necessary for symbolic model checking using Boolean Functional Vectors. We can compute the image (or pre-image) of a given set of states relative to a transition relation by computing the relational cross product which involves intersection and projection. The fix-point iterations also require set union and an equality check to determine the fix point. If the model is a circuit, however, forward image computation can be performed by symbolic simulation, making it unnecessary to compute the transition relation and its intersection with the current set of states for forward reachability analysis (Figure 2).

Consider a circuit with state elements $\vec{S} = \langle s_1, \ldots, s_j \rangle$ and transition functions $\vec{\Delta} = \langle \delta_1, \ldots, \delta_n \rangle$, we associate the choice variables $\vec{V} = \langle v_1, \ldots, v_n \rangle$ with




(a) Sorter

(b) A compare-and-swap element

**Fig. 3.** An Up/Down Sorter

$\vec{S}$. Given an $(\mathcal{I}, \preceq)$-BFV for a set of current states $\vec{F}$, we can simulate the circuit with the state variables set to $\vec{F}$ to obtain the next state vector $\vec{G} = \langle \delta_1(\vec{F}), \ldots, \delta_n(\vec{F}) \rangle$. We can then *reparameterize* $G$ using the *next-state* variables $\vec{V}' = \langle v_{n+1}, \ldots, v_{2n} \rangle$ to obtain a canonical abstraction $\vec{H}$. Note that the range of $\vec{G}$ is the exact image of the set of states we simulated with. The abstraction occurs during reparameterization and is determined by the partial order $\preceq$.

$\vec{H}$ is obtained by projecting out the current state bits from the extended $(\mathcal{I}_{2n}, \preceq_{2n})$-BFV $\langle v_1, \ldots, v_n, g_1, \ldots, g_n \rangle$ over $(\mathcal{I}_{2n}, \preceq_{2n})$ where $\mathcal{I}_{2n} = \{1, \ldots, 2n\}$ and $\preceq_{2n}$ is obtained by extending $\preceq$ to $\mathcal{I}_{2n}$ by requiring that all present state bits precede all next state bits. In practice, we project out one variable at a time, heuristically choosing a 'low-cost' variable. This allows us to optimize the projection algorithm by only modifying (and computing constraints for) the components that are dependent on the variable being projected out, thus avoiding the generation of the monolithic relation between the non-canonical $\vec{G}$ and the canonical $\vec{H}$.

We note that symbolic simulation followed by re-parameterization computes a complete *abstract interpretation* of the circuit transition function [3] when $(\mathcal{I}, \prec_{min})$ is a forest.

### 4.1 Example: Up/Down Sorter

Consider the Up/Down Sorter [9] of Figure 3. It has $2n$ words, each $m$ bits wide, arranged in two columns. Initially, all the words are set to 0. On any cycle, we may either insert a word at *Input* or extract the maximum word in the sorter from *Output*. The operation of the sorter may be viewed as a two step process: an insert or extract followed by a compare and swap. If we are inserting a word, then in the first step, all words in the left column are pushed down, with $L_1$ getting *Input*. For a read operation, in the first step all words in the right column are

pushed one up, with $R_n$ getting 0 and the value in $R_1$ read out at *Output*. In the second step, adjacent words are compared and if the entry in the left column is greater than the corresponding entry in the right column, the two are swapped.

1. Insert:
$$(L'_1 \leftarrow Input) \wedge (\forall 1 \leq i < n. L'_{i+1} \leftarrow L_i)$$
or Read:
$$(Output \leftarrow R_1) \wedge (\forall 1 \leq i < n. R'_i \leftarrow R_{i+1}) \wedge (R'_n \leftarrow 0)$$
2. Compare and Swap:
$$\forall 1 \leq i \leq n. (L'_i, R'_i) \leftarrow \text{if } (L_i > R_i) \text{ then } (R_i, L_i) \text{ else } (L_i, R_i)$$

The sorter works by maintaining the right column in sorted order so that $R_1 \geq R_2 \geq \cdots \geq R_n$. Additionally, the right entry is always greater than or equal to the corresponding left entry, i.e., $R_i \geq L_i$ for all $1 \leq i \leq n$. These two invariants guarantee that $R_1$ is the maximum entry in the sorter.

The basic data structure is a sorted array (the right column) which is hard to represent as a characteristic function with BDDs. Let $r_i^j$ represent the $j$-th bit of the $i$-th word of the right column. If we choose a word-based BDD variable order, i.e., $\langle r_1^1, r_1^2, \ldots, r_1^m, \ldots, r_n^1, \ldots, r_n^m \rangle$, then the BDD for the characteristic function is exponential in $m$. On the other hand, if we choose the bit-sliced order $\langle r_1^1, r_2^1, \ldots, r_n^1, \ldots, r_1^m, \ldots, r_n^m \rangle$, then the representation is exponential in $n$. If $m = n$, then we conjecture that the BDD for the characteristic function is exponential in $n$, irrespective of the variable order used.

The sorted array can be efficiently represented by a Boolean Functional Vector using BDDs, by organizing the words as a balanced binary search tree, e.g., Figure 4, and using a bit-sliced BDD variable order. The words are constrained only by the values of their ancestors. The relation between the unrelated words is captured implicitly by transitivity through common ancestors. The largest BDDs are for the $m$-th bits of the leaves. These are linear in $m$ and exponential in the depth of the tree, hence linear in $n$. Since there are $n \cdot m$ functions in the Boolean Functional Vector, one for each bit of each entry, the bound for the overall representation is quadratic in $n$ and $m$.

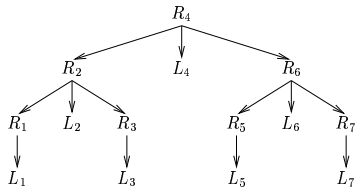



**Fig. 4.** Partial Order on Words for an Up-Down Sorter with $n = 7$.

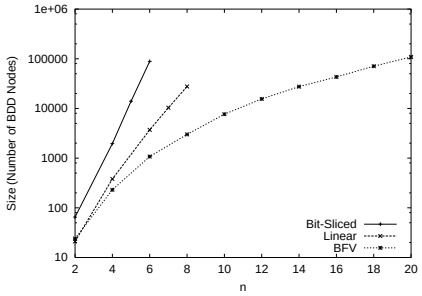



**Fig. 5.** Size of BDD representations for Up-Down Sorters with with $n = m$.

Figure 5 plots the sizes of the Boolean Functional Vector (using the binary tree layout) and characteristic function representations (using the linear and bit-sliced BDD variable orders) for the reachable states of an up-down sorter with a depth of $n$ ($2n$ entries), each $n$ bits wide ($m = n$).

The partial order in this case is accurate enough to represent the actual state-space of the sorter so that there is no information lost in the abstraction. The ordering constraints between entries unrelated by the partial order are captured by implicit transitivity through their common ancestor. Hence, the use of the partial order does not affect the size of the state-space; any linear extension of the partial order (e.g. the DFS order) would give the same size for the final state-space. However, the partial order is useful during reachability analysis since it prevents the computation of unnecessary constraints.

Figure 6 shows the peak live node count and runtime for reachability analysis, starting from a state with all entries set to 0. The experiments were performed on a Sun-Fire 280R (SUN UltraSPARC-III+ 1015 MHz CPU) with the memory limit set to 1 GB and the time limit set to 10 hours. The experiments with the characteristic function representation were performed using VIS with the MLP heuristics [8]. The current-state and next-state variables are interleaved in the BDD variable order. As expected, there is an exponential gap between the Boolean Functional Vector and characteristic function based approaches. Note that there is a significant performance improvement obtained by using the partial order BFVs instead of the totally ordered BFVs. The largest experiment with partially ordered BFVs ($n = 20$) had 800 state bits, all of which are relevant to the property, i.e. the global sorted order.

### 4.2 Example: FIFO Equivalence

Consider the two implementations of a FIFO queue shown in Figure 7. In the shift register, new entries are inserted at the front of the queue, the other entries shifting over by one. In the ring buffer, the new entries are inserted at the *head* pointer. In both implementations, the oldest entries are read out at the *tail* pointer. We can check the equivalence of the two implementations by performing reachability analysis on the product machine.

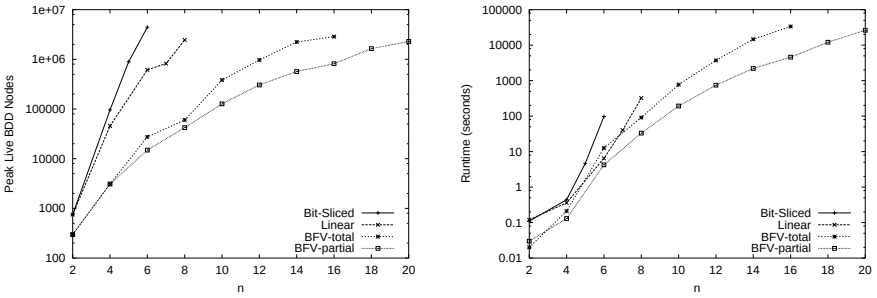
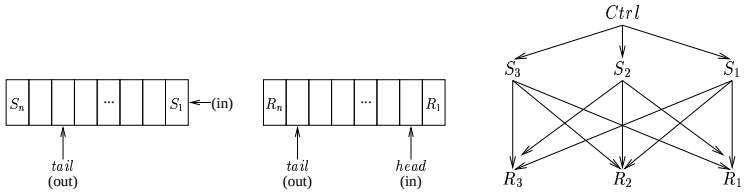



**Fig. 6.** Symbolic Reachability Analysis for Up-Down Sorters with $n = m$.

Given the value for the *head* pointer, there is a correspondence between the entries in the two queues. In general, though, we cannot fix such a correspondence since the *head* pointer can change. The BDD for the characteristic function of the reachable state-space is provably exponential in $n$, irrespective of the variable order. In [7], McMillan showed that there is a canonical conjunctive decomposition for this state-space that is quadratic in $n$. The basic observation is that the entries are *conditionally independent* since we can fix the correspondence once the value of the control signals is known. The conjunctive decomposition factors out the conditionally independent variables into separate components. The same observations essentially apply to the Boolean Functional Vector representation.

We can take this further by realizing that the entries in any one of the queues are not correlated. The correlation we are interested in is between entries in the shift register and the corresponding entries (given the control signals) in the ring buffer. Hence, we can use a partial order such as the one in Figure 7(c). We obtain approximately a 3X improvement in runtime with the partially ordered BFVs as compared to the totally ordered BFVs for all values of $n$ tried.




(a) Shift Register (b) Ring Buffer (c) Partial Order

**Fig. 7.** Equivalence Checking for two Implementations of a FIFO

## 5 Conclusions and Future Work

We have developed a general framework for symbolic simulation, model checking and abstraction using partially ordered Boolean Functional Vectors. Our examples demonstrated that this framework can allow us to efficiently verify some circuits where characteristic functions would fail. The most important future work from a practical point of view is the development of an automated abstraction-refinement framework and a dynamic reordering procedure for BFV components to work in tandem with BDD variable reordering.

## References


[1] Mark D. Aagaard, Robert B. Jones, and Carl-Johan H. Seger. Formal Verification Using Parametric Representations of Boolean Constraints. In *Proceedings of the 36th Design Automation Conference (DAC' 99)*, pages 402–407, 1999.

[2] Randal E. Bryant, Derek L. Beatty, and Carl-Johan H. Seger. Formal hardware verification by symbolic ternary trajectory evaluation. In *Proceedings of the 28th Design Automation Conference (DAC'91)*, pages 397–402, 1991.

[3] Ching-Tsun Chou. The Mathematical Foundation of Symbolic Trajectory Evaluation. In *Proceedings of the 11th International Conference on Computer Aided Verification (CAV'99)*, pages 196–207, 1999.

[4] O. Coudert, C. Berthet, and J. C. Madre. Verification of Sequential Machines using Boolean Functional Vectors. In *Proceedings of the IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 179–196, 1989.

[5] Amit Goel and Randal E. Bryant. Set Manipulation with Boolean Functional Vectors for Symbolic Reachability Analysis. In *2003 Design Automation and Test in Europe(DATE'03)*, pages 816–821, 2003.

[6] Shankar G. Govindaraju, David L. Dill, Alan J. Hu, and Mark A. Horowitz. Approximate Reachability with BDDs using Overlapping Projections. In *Proceedings of the 35th Design Automation Conference (DAC'98)*, pages 451–456, 1998.

[7] Kenneth L. McMillan. A Conjunctively Decomposed Boolean Representation for Symbolic Model Checking. In *Proceedings of the 8th International Conference on Computer Aided Verification (CAV'96)*, pages 13–24, 1996.

[8] In-Ho Moon, Gary D. Hachtel, and Fabio Somezni. Border-Block Triangular Form and Conjunction Schedule in Image Computation. In *3rd Internation Conference on Formal Methods in Computer Aided Design (FMCAD'00)*, pages 73–90, 2000.

[9] Simon W. Moore and Brian T. Graham. Tagged up/down sorter – A hardware priority queue. *The Computer Journal*, 38(9):695–703, 1995.

[10] H. J. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit State Enumeration of Finite State Machines Using BDDs. In *Proceedings of the IEEE International Conference on Computer-Aided Design(ICCAD'90)*, pages 130–133, 1990.

[11] Jin Yang and Carl-Johan H. Seger. Introduction to Generalized Symbolic Trajectory Evaluation. In *Proceedings of the IEEE International Conference on Computer Design(ICCD'01)*, pages 360–367, 2001.

[12] Jin Yang and Carl-Johan H. Seger. Generalized Symbolic Trajectory Evaluation - Abstraction in Action. In *Formal Methods in Computer-Aided Design (FMCAD'02)*, pages 70–87, 2002.