Knowledge Compilation for Boolean Functional Synthesis

S. Akshay, Jatin Arora, Supratik Chakraborty, S. Krishna, Divya Raghunathan and Shetal Shah
Indian Institute of Technology Bombay, India

Abstract*—Given a Boolean formula $F(\mathbf{X}, \mathbf{Y})$, where $\mathbf{X}$ is a vector of outputs and $\mathbf{Y}$ is a vector of inputs, the Boolean functional synthesis problem requires us to compute a Skolem function vector $\Psi(\mathbf{Y})$ for $\mathbf{X}$ such that $F(\Psi(\mathbf{Y}), \mathbf{Y})$ holds whenever $\exists \mathbf{X}\, F(\mathbf{X}, \mathbf{Y})$ holds. In this paper, we investigate the relation between the representation of the specification $F(\mathbf{X}, \mathbf{Y})$ and the complexity of synthesis. We introduce a new normal form for Boolean formulas, called SynNNF, that guarantees polynomial-time synthesis and also polynomial-time existential quantification for some order of quantification of variables. We show that several normal forms studied in the knowledge compilation literature are subsumed by SynNNF, although SynNNF can be super-polynomially more succinct than them. Motivated by these results, we propose an algorithm to convert a specification in CNF to SynNNF, with the intent of solving the Boolean functional synthesis problem. Experiments with a prototype implementation show that this approach solves several benchmarks beyond the reach of state-of-the-art tools.

I. Introduction

Boolean functional synthesis is the problem of synthesizing outputs as Boolean functions of inputs, while satisfying a declarative relational specification between inputs and outputs. Also called *Skolem function synthesis*, this problem has numerous applications including certified QBF solving, reactive control synthesis, circuit and program repair and the like. While variants of the problem have been studied since long [17], [3], there has been significant recent interest in designing practically efficient algorithms for Boolean functional synthesis. The resulting breed of algorithms [14], [23], [22], [11], [25], [18], [13], [2], [1], [15], [7], [24] have been empirically shown to work well on large collections of benchmarks. Nevertheless, there are not-so-large examples that are currently not solvable within reasonable resources by any known algorithm. To make matters worse, it is not even fully understood what properties of a Boolean relational specification or of its representation make it amenable to efficient synthesis. In this paper, we take a step towards answering this question. Specifically, we propose a new sub-class of negation normal form called SynNNF, such that every Boolean relational specification in SynNNF admits polynomial-time synthesis. Furthermore, a Boolean relational specification admits polynomial-time synthesis (by any algorithm) *if and only if* there exists a polynomial-sized *refinement* of the specification in SynNNF.

To illustrate the hardness of Boolean functional synthesis, consider the specification $F(\mathbf{X}_1, \mathbf{X}_2, \mathbf{Y}) \equiv (\mathbf{Y} = (\mathbf{X}_1 \times_{[n]} \mathbf{X}_2)) \wedge (\mathbf{X}_1 \neq 0 \cdots 01) \wedge (\mathbf{X}_2 \neq 0 \cdots 01)$, where $|\mathbf{Y}| = 2n$, $|\mathbf{X}_1| = |\mathbf{X}_2| = n$ and $\times_{[n]}$ denotes multiplication of n-bit unsigned integers. This specification asserts that $\mathbf{Y}$, viewed as a $2n$-bit unsigned integer, is the product of $\mathbf{X}_1$ and $\mathbf{X}_2$, each viewed as an n-bit unsigned integer different from 1. The specification $F(\mathbf{X}_1, \mathbf{X}_2, \mathbf{Y})$ can be easily represented as a circuit of AND, OR, NOT gates with $\mathcal{O}(n^2)$ gates. However, synthesizing $\mathbf{X}_1$ and $\mathbf{X}_2$ as functions of $\mathbf{Y}$ requires us to obtain a circuit that factorizes a $2n$-bit unsigned integer into factors different from 1, whenever possible. It is a long-standing open question whether such a circuit of size polynomial in n exists. Thus, although the relational specification is succinctly representable, the outputs expressed as functions of the inputs may not have any known succinct representation.

It was recently shown [1] that unless some long-standing complexity theoretic conjectures are falsified, Boolean functional synthesis must necessarily require super-polynomial (or even exponential) space and time. In the same work [1], it was also shown that if a specification is represented in *weak decomposable negation normal form wDNNF*, synthesis can be accomplished in time polynomial in the size of the specification. While this was a first step towards identifying a normal form with the explicit objective of polynomial-time synthesis, experimental results in [1] indicate that wDNNF doesn't really characterize specifications that admit efficient synthesis. Specifically, experiments in [1] showed that a polynomial-time algorithm intended for synthesis from wDNNF specifications ends up solving the synthesis problem for a large class of specifications *not in wDNNF*. This motivates us to ask if there exists a weaker (than wDNNF) sub-class of Boolean relational specifications that admit polynomial-time synthesis.

We answer the above question affirmatively in this paper, the polynomial dependence being quadratic in the number of outputs and the size of the specification. En route, we also show that the weaker normal form, viz. SynNNF, admits polynomial-time existential quantifier elimination of a set of variables for *some (not all)* order of quantification of variables. Applications of such quantifier elimination abound in practice, viz. image computation in symbolic model checking, synthesis of QBF certificates, computation of interpolants etc. Note that ensuring efficient quantifier elimination *for some ordering* of variables is simpler than ensuring efficient quantifier elimination *for all orderings* of variables – the latter having been addressed by normal forms like DNNF [9].

Our primary contributions can be summarized as follows:

- We present a new sub-class of negation normal form, called SynNNF, that admits polynomial-time synthesis and quantifier elimination for a set of variables.

- We show that SynNNF is super-polynomially (in some cases, exponentially) more succinct than several other sub-classes studied in the literature (viz. wDNNF, dDNNF, DNNF, FBDD, ROBDD), unless some long-standing complexity theoretic conjectures are falsified.
- We show that by suitably weakening SynNNF, we can precisely characterize the class of Boolean specifications that admit polynomial-time synthesis by a simple algorithm originally proposed in [1].
- We define a natural notion of *refinement of specifications w.r.t synthesis* and show that every specification that admits polynomial-time synthesis necessarily has a polynomial-sized refinement that is in SynNNF.
- We present a novel algorithm for compiling a Boolean relational specification in CNF to a *refined* specification in SynNNF. We call this *knowledge compilation for synthesis and quantifier elimination*.
- Finally, we present experimental results that show that synthesis by compiling to SynNNF solves a large set of benchmarks, including several benchmarks beyond the reach of existing tools.

Related Work: The literature on knowledge compilation of Boolean functions is rich and extensive [6], [9], [20], [10]. While existential quantification or *forgetting* of propositions has been studied in [16], [10], neither Boolean functional synthesis nor existential quantification for *some (not all)* ordering of variables has received attention in earlier work on knowledge compilation. Sub-classes of negation normal forms like DNNF and other variants [10] admit efficient existential quantification *for all* orders in which variables are quantified. However, if we are interested in only the result of existentially quantifying a given set of variables, these forms can be unnecessarily restrictive and exponentially larger. Recent work on Boolean functional synthesis [13], [14], [18], [24], [11], [2], [1], [8] has focused more on algorithms to directly synthesize outputs as functions of inputs. Some of these algorithms (viz. [11], [1], [8]) exploit properties of specific input representations for optimizing the synthesis process. This has led to the articulation of *sufficient* conditions on representation of specifications for efficient synthesis. For example, [15] suggested using input-first ROBDDs for efficient synthesis, and a quadratic-time algorithm for synthesis from input-first ROBDDs was presented in [11]. This result was subsequently generalized in [1], where it was shown that specifications in wDNNF (which strictly subsumes ROBDDs) suffice to give a quadratic-time algorithm for synthesis. As we show later, wDNNF can itself be generalized to SynNNF. In another line of investigation, it was shown [8] that if a CNF specification is decomposed into an *input-part* and an *output-part*, then synthesis can be achieved in time linear in the size of the CNF specification and k, where k is the smaller of the count of *maximal falsifiable subsets (MFS)* of the input-part and the count of *maximal satisfiable subsets (MSS)* of the output-part. However, this does not yield an algorithm whose running time is polynomial in the size of the representation of $F(\mathbf{X}, \mathbf{Y})$.

II. Preliminaries and notations

A Boolean formula $F(z_1, \ldots z_p)$ on p variables is a mapping $F : \{0,1\}^p \rightarrow \{0,1\}$. The set of variables $\{z_1, \ldots z_p\}$ is called the *support* of the formula, and denoted $\mathsf{sup}(F)$. We normally use $\mathbf{Z}$ to denote the sequence $(z_1, \ldots z_p)$. For notational convenience, we will also use $\mathbf{Z}$ to denote a set of variables, when there is no confusion. A *satisfying assignment* or *model* of F is a mapping of variables in $\mathsf{sup}(F)$ to $\{0,1\}$ such that F evaluates to 1 under this assignment. If π is a model of F, we write $\pi \models F$ and use $\pi(z_i)$ to denote the value assigned to $z_i \in \mathsf{sup}(F)$ by π. If $\mathbf{Z}'$ is a subsequence of $\mathbf{Z}$, we use $\pi{\downarrow}\mathbf{Z}'$ to denote the projection of π on $\mathbf{Z}'$, i.e. $(\pi({z'}_1), \ldots \pi({z'}_k))$, where $k = |\mathbf{Z}'|$. We use $\mathsf{form}(\pi{\downarrow}\mathbf{Z}')$ to denote the conjunction of *literals* (i.e. variables or their negation) corresponding to $\pi{\downarrow}\mathbf{Z}'$. For example, if π assigns 1 to z_1, z_3 and 0 to z_2, z_4 and $\mathbf{Z}' = (z_1, z_4)$, then $\mathsf{form}(\pi{\downarrow}\mathbf{Z}') = z_1 \wedge \neg z_4$.

1) Negation normal form (NNF): This is the class of Boolean formulas in which (i) the only operators used are conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$), and (ii) negation is applied only to variables. Every Boolean formula can be converted to a semantically equivalent NNF formula. Moreover, this conversion can be done in linear time for representations like AIGs, ROBDDs, Boolean circuits etc.

2) Unate formulas: Let $F|_{z_i=0}$ (resp. $F|_{z_i=1}$) denote the positive (resp. negative) *cofactor* of F with respect to z_i. Then, F is *positive unate* in $z_i \in \mathsf{sup}(F)$ iff $F|_{z_i=0} \Rightarrow F|_{z_i=1}$. Similarly, F is *negative unate* in z_i iff $F|_{z_i=1} \Rightarrow F|_{z_i=0}$. A *literal* ℓ is said to be *pure* in an NNF formula F iff F has at least one instance of ℓ but no instance of $\neg\ell$. If z_i (resp. $\neg z_i$) is pure in F, then F is positive (resp. negative) unate in z_i.

3) Independent support and functionally defined variables: A subsequence $\mathbf{Z}'$ of $\mathbf{Z}$ is said to be an *independent support* of F iff every pair of satisfying assignments π, π' of F that agree on the assignment of variables in $\mathbf{Z}'$ also agree on the assignment of all variables in $\mathbf{Z}$. Variables not in $\mathbf{Z}'$ are said to be functionally defined by the independent support. Effectively, the assignment of variables in $\mathbf{Z}'$ uniquely determine that of functionally defined variables, when satisfying F. CNF encodings of Boolean functions originally specified as circuits, ROBDDs, AIGs etc. often use Tseitin encoding [26], which introduces a large number of functionally defined variables.

4) Boolean functional synthesis: Unless mentioned otherwise, we use $\mathbf{X} = (x_1, \ldots x_n)$ to denote a sequence of Boolean outputs, and $\mathbf{Y} = (y_1, \ldots y_m)$ to denote a sequence of Boolean inputs. The *Boolean functional synthesis* problem, henceforth denoted BFnS, asks: given a Boolean formula $F(\mathbf{X}, \mathbf{Y})$ specifying a relation between inputs $\mathbf{Y}$ and outputs $\mathbf{X}$, determine functions $\mathbf{\Psi} = (\psi_1(\mathbf{Y}), \ldots \psi_n(\mathbf{Y}))$ such that $F(\mathbf{\Psi}, \mathbf{Y})$ holds whenever $\exists \mathbf{X} F(\mathbf{X}, \mathbf{Y})$ holds. Thus, $\forall \mathbf{Y} (\exists \mathbf{X}\, F(\mathbf{X}, \mathbf{Y}) \Leftrightarrow F(\mathbf{\Psi}, \mathbf{Y}))$ must be a tautology. The function ψ_i is called a *Skolem function* for x_i in F, and $\mathbf{\Psi}$ is called a *Skolem function vector* for $\mathbf{X}$ in F.

For $1 \leq i \leq j \leq n$, we use $\mathbf{X}_i^j$ to denote the subsequence $(x_i, x_{i+1}, \ldots x_j)$. If $i \leq k < j$, we sometimes use $(\mathbf{X}_i^k, \mathbf{X}_{k+1}^j)$ interchangeably with $\mathbf{X}_i^j$ for notational convenience. Let

$F^{(i-1)}(\mathbf{X}_i^n, \mathbf{Y})$ denote $\exists \mathbf{X}_1^{i-1} F(\mathbf{X}_1^{i-1}, \mathbf{X}_i^n, \mathbf{Y})$. It has been argued in [14], [11], [2], [12] that the BFnS problem for $F(\mathbf{X}, \mathbf{Y})$ can be solved by first ordering the outputs, say as $x_1 \prec x_2 \cdots \prec x_n$, and then synthesizing a function $\psi_i(\mathbf{X}_{i+1}^n, \mathbf{Y}) \equiv F^{(i-1)}(\mathbf{X}_i^n, \mathbf{Y})[x_i \mapsto 1]$ for each x_i. This ensures that $F^{(i-1)}(\psi_i, \mathbf{X}_{i+1}^n, \mathbf{Y}) \Leftrightarrow \exists x_i F^{(i-1)}(x_i, \mathbf{X}_{i+1}^n, \mathbf{Y})$. Once all such ψ_is are obtained, one can substitute ψ_{i+1} through ψ_n for x_{i+1} through x_n respectively, in ψ_i to obtain a Skolem function for x_i as a function of $\mathbf{Y}$. The primary problem of using this approach as-is is the exponential blow-up incurred in the size of the Skolem functions.

5) DAG representations: For an NNF formula F, its DAG representation is naturally induced by the structure of F. Specifically, if F is simply a literal ℓ, its DAG representation is a leaf labeled ℓ. If F is F_1 op F_2 where op $\in \{\vee, \wedge\}$, its DAG representation is a node labeled op with two children, viz. the DAG representations of F_1 and F_2. W.l.o.g. we assume that a DAG representation of F is always in a *simplified* form, where $t \wedge 1$, $t \vee 0$, $t \wedge t$ and $t \vee t$ are replaced by t, $t \wedge 0$ is replaced by 0 and $t \vee 1$ is replaced by 1 for every node t. We use $|F|$ for the node count in the DAG representation of F.

FBDD and ROBDD are well-known representations of Boolean formulas and we skip their definitions. We briefly recall the definitions of DNNF, dDNNF and wDNNF below. Let α be the subformula represented by an internal node N (labeled by $\wedge$ or $\vee$) in a DAG representation of an NNF formula F. We use $lits(\alpha)$ to denote the set of literals labeling leaves that have a path to the node N representing α in the DAG representation of F. We also use $atoms(\alpha)$ to denote the underlying set of variables in sup(F) that appear in $lits(\alpha)$. For each $\wedge$-labeled internal node N in the DAG of F with $\alpha = \alpha_1 \wedge \ldots \wedge \alpha_k$ being the subformula represented by N, if for all distinct indices $r, s \in \{1, \ldots k\}$, $atoms(\alpha_r) \cap atoms(\alpha_s) = \emptyset$, then F is said to be in DNNF [9]. If, instead, for all distinct indices $r, s \in \{1, \ldots k\}$, $lits(\alpha_r) \cap \{\neg \ell \mid \ell \in lits(\alpha_s)\} = \emptyset$, then F is said to be in wDNNF [1]. Finally $F(\mathbf{X}, \mathbf{Y})$ is said to be in deterministic DNNF(or dDNNF) [10] if F is in DNNF and for each $\vee$-labeled internal node D in the DAG of F with $\beta = \beta_1 \vee \ldots \vee \beta_k$ being the subformula represented by D, $\beta_r \wedge \beta_s$ is a contradiction for all distinct indices r, s.

6) Positive form of input specification: Given a specification $F(\mathbf{X}, \mathbf{Y})$ in NNF, we denote by $\widehat{F}(\mathbf{X}, \overline{\mathbf{X}}, \mathbf{Y})$ the formula obtained by replacing every occurrence of $\neg x_i$ ($x_i \in \mathbf{X}$) in F with a fresh variable $\overline{x_i}$. This is also called the *positive form* of the specification and has been used earlier in [2]. Observe that for any F in NNF, $\widehat{F}$ is positive unate (or *monotone*) in all variables in $\mathbf{X}$ and $\overline{\mathbf{X}}$. For $i \in \{1, \ldots n\}$, we sometimes split $\mathbf{X}$ into two parts, $\mathbf{X}_1^i$ and $\mathbf{X}_{i+1}^n$, and represent $\widehat{F}(\mathbf{X}, \overline{\mathbf{X}}, \mathbf{Y})$ as $\widehat{F}(\mathbf{X}_1^i, \mathbf{X}_{i+1}^n, \overline{\mathbf{X}}_1^i, \overline{\mathbf{X}}_{i+1}^n, \mathbf{Y})$. For $b, c \in \{0, 1\}$, let $\mathbf{b}^i$ (resp. $\mathbf{c}^i$) denote a vector of i b's (resp. c's). For notational convenience, we use $\widehat{F}(\mathbf{b}^i, \mathbf{X}_{i+1}^n, \mathbf{c}^i, \overline{\mathbf{X}}_{i+1}^n, \mathbf{Y})$ to denote $\widehat{F}(\mathbf{X}_1^i, \mathbf{X}_{i+1}^n, \overline{\mathbf{X}}_1^i, \overline{\mathbf{X}}_{i+1}^n, \mathbf{Y})|_{\mathbf{X}_1^i = \mathbf{b}^i, \overline{\mathbf{X}}_1^i = \mathbf{c}^i}$.

III. A New Normal Form for Efficient Synthesis

In [1], it was shown that if $F(\mathbf{X}, \mathbf{Y})$ is represented as a ROBDD/FBDD or in DNNF or in wDNNF form, Skolem functions can be synthesized in time polynomial in $|F|$. In this section, we define a new normal form called SynNNF that subsumes and is more succinct than these other normal forms, and yet guarantees efficient synthesis of Skolem functions.

Definition 1. *Given a specification $F(\mathbf{X}, \mathbf{Y})$, for every $i \in \{1, \ldots n\}$ we define the i^{th}-reduct of $\widehat{F}$, denoted $[\widehat{F}]_i$, to be $\widehat{F}(1^{i-1}, \mathbf{X}_i^n, 1^{i-1}, \overline{\mathbf{X}}_i^n, \mathbf{Y})$. We also define $[\widehat{F}]_{n+1}$ to be $\widehat{F}(1^n, 1^n, \mathbf{Y})$.*

Note that $[\widehat{F}]_1$ is the same as $\widehat{F}$, and sup$([\widehat{F}]_i) = \mathbf{X}_i^n \cup \overline{\mathbf{X}}_i^n \cup \mathbf{Y}$ for $i \in \{1, \ldots n\}$.

Example 1. *Consider the* NNF *formula $K(x_1, x_2, y_1, y_2) = (x_1 \vee x_2) \wedge (\neg x_2 \vee y_1) \wedge (\neg y_1 \vee y_2)$. Then $\widehat{K} = ((x_1 \vee x_2) \wedge (\overline{x_2} \vee y_1) \wedge (\neg y_1 \vee y_2))$. Thus, we have $[\widehat{K}]_1 = \widehat{K}$ and $[\widehat{K}]_2 = \widehat{K}[x_1 \mapsto 1, \overline{x_1} \mapsto 1] = (\overline{x_2} \vee y_1) \wedge (\neg y_1 \vee y_2)$.*

Next, we define a useful property for the i^{th}-reduct, which will be crucial for efficient synthesis of Skolem functions.

Definition 2. *Given $F(\mathbf{X}, \mathbf{Y})$, let α_i^{jk} denote $[\widehat{F}]_i[x_i \mapsto j, \overline{x}_i \mapsto k, \overline{\mathbf{X}}_{i+1}^n \mapsto \neg \mathbf{X}_{i+1}^n]$, where $j, k \in \{0, 1\}$. We say that $[\widehat{F}]_i$ is $\wedge_i$-unrealizable if $\zeta = \alpha_i^{11} \wedge \neg \alpha_i^{10} \wedge \neg \alpha_i^{01}$ is unsatisfiable.*

In other words, there exists no assignment to $\mathbf{X}_{i+1}^n$ and $\mathbf{Y}$ such that $[\widehat{F}]_i$ is equivalent to $x_i \wedge \overline{x}_i$. Indeed, if an assignment makes ζ true, then it also makes $[\widehat{F}]_i$ equivalent to $x_i \wedge \overline{x}_i$ (i.e., $[\widehat{F}]_i = 1$ for $x_i, \overline{x_i}$ having values $(1, 1)$, but not for $(0, 1)$, $(1, 0)$, $(0, 0)$). Note that since $[\widehat{F}]_i$ is positive unate in x_i and $\overline{x_i}$, ζ is satisfiable iff $\zeta \wedge \neg \alpha_i^{00}$ is satisfiable; hence we need not conjoin $\neg \alpha_i^{00}$ in the definition of ζ. A sufficient condition for $[\widehat{F}]_i$ to be $\wedge_i$-unrealizable is that in the DAG representation of $[\widehat{F}]_i$, there is no pair of paths – one from x_i and the other from $\overline{x_i}$ – which meet for the first time at an $\wedge$-labeled node.

Coming back to Example 1, $[\widehat{K}]_1$ is $\wedge_1$-unrealizable since there is no leaf labeled $\overline{x_1}$ in its DAG representation. Similarly, $[\widehat{K}]_2 = (\overline{x_2} \vee y_1) \wedge (\neg y_1 \vee y_2)$ is $\wedge_2$-unrealizable since there is no leaf labeled x_2 in the DAG representation of $[\widehat{K}]_2$ (although such a leaf exists in the DAG representation of $[\widehat{K}]_1$).

Example 2. *Let $H(x_1, x_2, y_1, y_2) = (x_1 \vee x_2 \vee y_1) \wedge (\neg x_1 \vee (\neg x_2 \wedge y_2))$. Then $\widehat{H}(\mathbf{X}, \overline{\mathbf{X}}, \mathbf{Y}) = (x_1 \vee x_2 \vee y_1) \wedge (\overline{x_1} \vee (\overline{x_2} \wedge y_2))$. Using the notation in Definition 2, $\alpha_1^{11} = 1$, $\alpha_1^{10} = \neg x_2 \wedge y_2$ and $\alpha_1^{01} = (x_2 \vee y_1)$. There is an assignment ($x_2 = 0, y_2 = 0, y_1 = 0$) such that $(\alpha_1^{11} \wedge \neg \alpha_1^{10} \wedge \neg \alpha_1^{01})$ is satisfiable. Hence $[\widehat{H}]_1$ is not $\wedge_1$-unrealizable (equivalently, it is $\wedge_1$-realizable). However, $[\widehat{H}]_2 = \widehat{H}[x_1 \mapsto 1, \overline{x_1} \mapsto 1] = 1$; hence it is vacuously $\wedge_2$-unrealizable.*

We are now ready to define our new negation normal form for synthesis and for existential quantification of a set of variables.

Definition 3. *A formula $F(\mathbf{X}, \mathbf{Y})$ is said to be in synthesizable* NNF *(or SynNNF) wrt the sequence $\mathbf{X}$ if F is in* NNF*, and for all $1 \leq i \leq n$, $[\widehat{F}]_i$ is $\wedge_i$-unrealizable.*

Considering our previous examples, K is in SynNNF while H

is not. Also note that neither of them are in DNNF or wDNNF. Additionally, the functions as presented do not correspond to ROBDD/FBDD representations either. We now show *three* important properties of SynNNF which motivate our proposal of SynNNF as a normal form for synthesis and existential quantification.

1) SynNNF leads to efficient quantification and synthesis: Our first result shows that existentially quantifying $\mathbf{X}$ and synthesizing $\mathbf{X}$ are easy for SynNNF.

Theorem 1. *Suppose $F(\mathbf{X}, \mathbf{Y})$ is in SynNNF. Then,*

(i) $\exists \mathbf{X}_1^i F(\mathbf{X}, \mathbf{Y}) \Leftrightarrow [\widehat{F}]_{i+1}[\overline{\mathbf{X}}_{i+1}^n \mapsto \neg \mathbf{X}_{i+1}^n]$ *for* $i \in \{1, ..n\}$

(ii) a Skolem function vector Ψ_1^n for $\mathbf{X}_1^n$ can be computed in $\mathcal{O}(n^2 \cdot |F|)$ time and $\mathcal{O}(n \cdot |F|)$ space, where $|\mathbf{X}| = n$.

Proof. The proof of Part (i) is similar to that of Theorem 2(a) in [1]. We prove by induction on i. For $i = 1$, $\exists \mathbf{X}_1^1 F(\mathbf{X}, \mathbf{Y}) \Leftrightarrow \widehat{F}(1, \mathbf{X}_2^n, 0, \neg \mathbf{X}_2^n, \mathbf{Y}) \vee \widehat{F}(0, \mathbf{X}_2^n, 1, \neg \mathbf{X}_2^n, \mathbf{Y}) \Rightarrow \widehat{F}(1, \mathbf{X}_2^n, 1, \neg \mathbf{X}_2^n, \mathbf{Y}) = [\widehat{F}]_2[\overline{\mathbf{X}}_2^n \mapsto \neg \mathbf{X}_2^n]$ (by positive unateness of $\widehat{F}$ in $x_1, \overline{x_1}$). Conversely, as F is in SynNNF, $[\widehat{F}]_2$ is $\wedge_2$-unrealizable, which implies that with notation as in Definition 2, $\alpha_1^{11} \Rightarrow \alpha_1^{10} \vee \alpha_1^{01}$, i.e., $\widehat{F}(1, \mathbf{X}_2^n, 1, \neg \mathbf{X}_2^n, \mathbf{Y}) \Rightarrow \widehat{F}(1, \mathbf{X}_2^n, 0, \neg \mathbf{X}_2^n, \mathbf{Y}) \vee \widehat{F}(0, \mathbf{X}_2^n, 1, \neg \mathbf{X}_2^n, \mathbf{Y})$. This give us the proof in the reverse direction, i.e., $[\widehat{F}]_2[\overline{\mathbf{X}}_2^n \mapsto \neg \mathbf{X}_2^n] \Rightarrow \exists \mathbf{X}_1^1 F(\mathbf{X}, \mathbf{Y})$.

Suppose the statement holds for $1 \leq i < n$. We will show that it holds for $i + 1$ as well. By inductive hypothesis and definition of existential quantification, $\exists \mathbf{X}_1^{i+1} F(\mathbf{X}, \mathbf{Y}) \Leftrightarrow \exists x_{i+1} [\widehat{F}]_{i+1}[\overline{\mathbf{X}}_{i+1}^n \mapsto \neg \mathbf{X}_{i+1}^n] \Leftrightarrow [\widehat{F}]_{i+1}[x_i \mapsto 1, \overline{\mathbf{X}}_{i+1}^n \mapsto \neg \mathbf{X}_{i+1}^n] \vee [\widehat{F}]_{i+1}[x_i \mapsto 0, \overline{\mathbf{X}}_{i+1}^n \mapsto \neg \mathbf{X}_{i+1}^n]$. Again, using unateness of $[\widehat{F}]_{i+1}$ in x_{i+1} and $\overline{x_{i+1}}$ in one direction, and using the defining property of SynNNF ($\alpha_{i+1}^{11} \Rightarrow \alpha_{i+1}^{10} \vee \alpha_{i+1}^{01}$) in the other direction, we obtain $\exists \mathbf{X}_1^{i+1} F(\mathbf{X}, \mathbf{Y}) \Leftrightarrow [\widehat{F}]_{i+2}[\overline{\mathbf{X}}_{i+2}^n \mapsto \neg \mathbf{X}_{i+2}^n]$.

Part(ii): For $i \in \{1, \ldots n\}$, let $\psi_i'(\mathbf{X}_{i+1}^n, \mathbf{Y})$ denote $[\widehat{F}]_i[x_i \mapsto 1, \overline{x}_i \mapsto 0, \overline{\mathbf{X}}_{i+1}^n \mapsto \neg \mathbf{X}_{i+1}^n] = \alpha_i^{10}$. Further, from n to 1, we recursively define $\psi_n(\mathbf{Y}) = \psi_n'(\mathbf{Y})$ and $\psi_i(\mathbf{Y}) = \psi_i'(\Psi_{i+1}^n(\mathbf{Y}), \mathbf{Y})$. We can now show that $\psi_i(\mathbf{Y})$ is indeed a correct Skolem function for x_i in F. Starting from n to 1, we know from the preliminaries that $F^{(n-1)}[x_n \mapsto 1]$ gives a correct Skolem function for x_n in F. From part (i) above, $F^{(n-1)} \Leftrightarrow [\widehat{F}]_n[\overline{\mathbf{X}_n^n} \mapsto \neg \mathbf{X}_n^n]$. Hence $\alpha_n^{10} = \psi_n = \psi_n'$ gives a correct Skolem function for x_n in F. For any $i \in \{1, \ldots n-1\}$, assuming that Ψ_{i+1}^n gives a correct Skolem function vector for $\mathbf{X}_{i+1}^n$ in F, the same argument shows that $\psi_i'(\psi_{i+1}^n(\mathbf{Y}), \mathbf{Y})$ is a correct Skolem function for x_i in F.

Finally, note that $|\psi_n|$ is at most $|\widehat{F}|$, which is in $\mathcal{O}(|F|)$. A DAG representation of ψ_{n-k} requires a fresh copy of $[\widehat{F}]_{n-k}$, but can re-use the DAG representations of ψ_j for $j \in \{n-k+1, \ldots n\}$ as sub-DAGs. Thus, $|\psi_{n-k}|$ is in $\mathcal{O}(k \cdot |F|)$. Hence, if we use a multi-rooted DAG to represent all Skolem functions together, we need only $\mathcal{O}(n \cdot |F|)$ nodes. The time required is in $\mathcal{O}(n^2 \cdot |F|)$ since the resulting DAG has $\sum_{k=1}^n k$ edges (root of ψ_j connects to a leaf of every ψ_i for $i < j$). □

The above polynomial-time strategy based on $[\widehat{F}]_i$ was used in [1] for computing over-approximations of Skolem functions $\psi_i(\mathbf{X}_{i+1}, \mathbf{Y})$ for each $x_i \in \mathbf{X}$. Specifically, it was shown that $[\widehat{F}]_i[x_i \mapsto 1, \overline{x_i} \mapsto 1]$ over-approximates $\exists \mathbf{X}_1^i F(\mathbf{X}, \mathbf{Y})$ and $[\widehat{F}]_i[x_i \mapsto 1, \overline{x_i} \mapsto 0]$ over-approximates a Skolem function for x_i in F. In the remainder of this paper, we refer to the functions ψ_i used in the proof of Part (ii) above as GACKS functions (after the author names of [1]). We use Ψ_1^n to denote the GACKS (Skolem) function vector $(\psi_1, \ldots, \psi_n)$.

2) Succinctness of SynNNF: We now show that SynNNF strictly subsumes many known representations used for efficient analysis of Boolean functions. In the following theorem, sizes and times are in terms of the number of input and output variables. Proofs of all subsequent lemmas and theorems are defered to the Appendix for lack of space.

Theorem 2. *(i) Every specification in ROBDD/FBDD, dDNNF, DNNF or wDNNF form is either already in SynNNF or can be compiled in linear time to SynNNF.*

(ii) There exist poly-sized SynNNF specifications that only admit

a) exponential sized FBDD representations.

b) super-polynomial sized dDNNF representations, unless P = VNP

c) super-polynomial sized wDNNF and DNNF representations, unless P = NP.

(iii) There exist poly-sized NNF-representations that only admit super-polynomial sized SynNNF representations, unless the polynomial hierarchy collapses.

In the above, VNP is the algebraic analogue of NP [27]. Also, (iii) shows that we cannot always hope to obtain a succinct SynNNF representation.

3) SynNNF "almost" characterizes efficient synthesis using GACKS *functions:* We now show that SynNNF precisely characterizes specifications that admit linear-time existential quantification of output variables *a la* Theorem 1(i). Further, a slight weakening of the SynNNF condition by restricting the assignments on $\mathbf{X}_{i+1}^n$ gives us a necessary and sufficient condition for poly-time synthesis using GACKS functions.

Theorem 3. *Given a relational specification $F(\mathbf{X}, \mathbf{Y})$,*

1) F is in SynNNF iff $\exists \mathbf{X}_1^i F(\mathbf{X}, \mathbf{Y}) \Leftrightarrow [\widehat{F}]_{i+1}[\overline{\mathbf{X}}_{i+1}^n \mapsto \neg \mathbf{X}_{i+1}^n]$

2) The GACKS-function vector Ψ_1^n is a Skolem function vector for $\mathbf{X}_1^n$ in $F(\mathbf{X}, \mathbf{Y})$ iff $[\widehat{F}]_i[\mathbf{X}_{i+1}^n \mapsto \Psi_{i+1}^n, \overline{\mathbf{X}_{i+1}^n} \mapsto \neg \Psi_{i+1}^n]$ is $\wedge_i$-unrealizable for all $i \in \{1 \ldots n\}$.

In [14], it was shown that an *error formula* ε for Ψ_1^n, defined as $F(\mathbf{X}, \mathbf{Y}) \wedge \neg(\mathbf{X}', \mathbf{Y}) \wedge \bigwedge_{i=1}^n (x_i' \leftrightarrow \Psi_i)$ is unsatisfiable iff Ψ_1^n is a Skolem function vector for F. Therefore, an (un)satisfiability check for ε serves to check if $[\widehat{F}]_i[\mathbf{X}_{i+1}^n \mapsto \Psi_{i+1}^n]$ is $\wedge_i$-unrealizable for all $i \in \{1 \ldots n\}$. Further, in [1], it was observed experimentally, that GACKS functions give correct Skolem functions, even when the specifications are not in wDNNF. This surprising behavior, which was left unexplained in [1], can now be explained using SynNNF, thanks to Theorem 3(2).

Note that Theorem 3(2) weakens the requirement of SynNNF since $\mathbf{X}^n_{i+1}$ are constrained to take only the values defined by Ψ^n_{i+1}. For an example of a specification not in SynNNF for which GACKS functions are correct Skolem functions, consider again H from Example 2, which we saw was not in SynNNF. In this case, $\psi_1'(x_2, \mathbf{Y}) = [\widehat{H}]_1[x_1 \mapsto 1, \overline{x}_1 \mapsto 0, \overline{x}_2 \mapsto \neg x_2] = \neg x_2 \wedge y_2$ and $\psi_2(\mathbf{Y}) = \psi_2'(\mathbf{Y}) = [\widehat{H}]_2[x_2 \mapsto 1, \overline{x}_2 \mapsto 0] = 1$. Therefore, $\psi_1(\mathbf{Y}) = \psi_1'[x_2 \mapsto \psi_2(\mathbf{Y})] = 0$. It can be verified that $x_1 = \psi_1(\mathbf{Y}) = 0, x_2 = \psi_2(\mathbf{Y}) = 1$ is indeed a correct Skolem function vector for $\mathbf{X}$ in H. Note also that H satisfies the condition of Theorem 3(2) since $[\widehat{H}]_1[x_2 \mapsto \psi_2, \overline{x}_2 \mapsto \neg\psi_2] = \overline{x}_1 \not\Leftrightarrow (x_1 \wedge \overline{x}_1)$, and $[\widehat{H}]_2 = 1$.

IV. Refinement for Synthesis

Given a specification $F(\mathbf{X}, \mathbf{Y})$, sometimes it is easier to solve the BFnS problem for a "simpler" specification $\widetilde{\mathsf{F}}(\mathbf{X}, \mathbf{Y})$ such that a solution for $\widetilde{\mathsf{F}}$ also serves as a solution for F. While "simplifications" of this nature have been used in earlier work [14], [1], [22], [7], we formalize this notion below as one of refinement.

Definition 4. *Let $F(\mathbf{X}, \mathbf{Y})$ and $\widetilde{\mathsf{F}}(\mathbf{X}, \mathbf{Y})$ be Boolean relational specifications on the same input and output vectors. We say that $\widetilde{\mathsf{F}}$ refines F* w.r.t. synthesis, *denoted $\widetilde{\mathsf{F}} \preceq_{syn} F$, iff the following conditions hold: (a) $\forall \mathbf{Y}\left(\exists \mathbf{X} F(\mathbf{X}, \mathbf{Y}) \Rightarrow \exists \mathbf{X}' \widetilde{\mathsf{F}}(\mathbf{X}', \mathbf{Y})\right)$, and (b) $\forall \mathbf{Y} \forall \mathbf{X}' \left(\left(\exists \mathbf{X} F(\mathbf{X}, \mathbf{Y}) \wedge \widetilde{\mathsf{F}}(\mathbf{X}', \mathbf{Y})\right) \Rightarrow F(\mathbf{X}', \mathbf{Y})\right)$.*

Informally, condition (a) specifies that $\widetilde{\mathsf{F}}$ doesn't restrict the set of input valuations (i.e. $\mathbf{Y}$) over which the specification F can be satisfied, and condition (b) specifies that for all such input valuations $\mathbf{Y}$, any $\mathbf{X}'$ that satisfies $\widetilde{\mathsf{F}}$ also satisfies F.

Lemma 4. *If $\widetilde{\mathsf{F}} \preceq_{syn} F$, every Skolem function vector for $\mathbf{X}$ in $\widetilde{\mathsf{F}}$ is also a Skolem function vector for $\mathbf{X}$ in F.*

We say $\widetilde{\mathsf{F}}$ *refines* F w.r.t. synthesis because the set of all Skolem function vectors for $\mathbf{X}$ in $\widetilde{\mathsf{F}}$ is a subset of that for $\mathbf{X}$ in F. Note that Definition 4 provides a direct 2QBF-SAT based check of whether $\widetilde{\mathsf{F}}$ refines F without referring to the details of how $\widetilde{\mathsf{F}}$ is obtained from F.

Example 3. *Let $G(x_1, x_2, y_1, y_2) \equiv (\neg x_1 \vee x_2 \vee y_1) \wedge (x_1 \vee \neg x_2) \wedge (x_1 \vee \neg y_1) \wedge (x_2 \vee y_2)$ and $\widetilde{\mathsf{G}}(x_1, x_2, y_1, y_2) \equiv x_2 \wedge x_1$. Although $G \not\Leftrightarrow \widetilde{\mathsf{G}}$, both conditions (a) and (b) of Definition 4 are satisfied; hence $\widetilde{\mathsf{G}} \preceq_{syn} G$.*

Proposition 5. *1) $\preceq_{syn}$ is a reflexive and transitive relation on all Boolean relational specifications on $\mathbf{X} \cup \mathbf{Y}$.*

2) If $\bigwedge_{y_j \in \mathbf{Y}} \left(F|_{y_j=0} \Leftrightarrow F|_{y_j=1}\right)$ and $\pi \models F(\mathbf{X}, \mathbf{Y})$, then $\mathsf{form}(\pi{\downarrow}\mathbf{X}) \preceq_{syn} F$.

3) If $\bigwedge_{x_i \in \mathbf{X}} \left(F|_{x_i=0} \Leftrightarrow F|_{x_i=1}\right)$, then $1 \preceq_{syn} F$.

4) If F is positive (resp. negative) unate in $x_i \in \mathbf{X}$, then $x_i \wedge F|_{x_i=1}$ (resp. $\neg x_i \wedge F|_{x_i=0}$) $\preceq_{syn} F$.

5) If $\widetilde{\mathsf{F}}_1 \preceq_{syn} F_1$ and $\widetilde{\mathsf{F}}_2 \preceq_{syn} F_2$, then

a) $(\widetilde{\mathsf{F}}_1 \vee \widetilde{\mathsf{F}}_2) \preceq_{syn} (F_1 \vee F_2)$.

b) $(\widetilde{\mathsf{F}}_1 \wedge \widetilde{\mathsf{F}}_2) \preceq_{syn} (F_1 \wedge F_2)$ if the output supports of F_1 and F_2 are disjoint.

Propositions 5(2) and 5(3) effectively require $F(\mathbf{X}, \mathbf{Y})$ to be semantically (but not necessarily syntactically) independent of $\mathbf{Y}$ and $\mathbf{X}$ respectively. While these may appear to be degenerate cases, we will soon see that both these propositions turn out to be useful when recursively compiling a CNF specification into refined SynNNF specification. Interestingly, a version of Proposition 5(4) was used in a pre-processing step of BFSS [1], although the precise notion of refinement w.r.t. synthesis was not defined there. Thanks to Definition 4, we can now generalize Proposition 5(4) to refine a specification even when F is not unate in any output variable. We discuss below how this can be done.

Suppose the specification $F(\mathbf{X}, \mathbf{Y})$ uniquely defines an output variable as a function of other input and output variables. For example, if $F(\mathbf{X}, \mathbf{Y}) \equiv (\neg x_i \vee x_j) \wedge (\neg x_i \vee y_k) \wedge (x_i \vee \neg x_j \vee \neg y_k) \wedge \cdots$, then $F(\mathbf{X}, \mathbf{Y}) \Rightarrow (x_i \Leftrightarrow (x_j \wedge y_k))$. Such specifications arise naturally when a non-CNF Boolean formula is converted to CNF via Tseitin encoding [26]. Variables like x_i above are said to be *functionally determined* (henceforth called FD) in F, and implied functional dependencies like $(x_i \leftrightarrow (x_j \wedge y_k))$ are called *functional definitions* (henceforth called *f-defs*) of FD variables in F.

Let $\mathbf{T} \subseteq \mathbf{X}$ be a set of FD output variables in F, and let $\mathsf{Fun}_{\mathbf{T}}$ be the conjunction of f-defs of all variables in $\mathbf{T}$. We say that $(\mathbf{T}, \mathsf{Fun}_{\mathbf{T}})$ is an *acyclic system of f-defs* if no variable in $\mathbf{T}$ transitively depends on itself via the functional definitions in $\mathsf{Fun}_{\mathbf{T}}$. In other words, $\mathsf{Fun}_{\mathbf{T}}$ induces an acyclic system of functional dependencies between variables in $\mathbf{T}$. For $x_i \in \mathbf{X} \setminus \mathbf{T}$, define $\theta_{F,\mathbf{T},x_i,a}$ to be the formula $\Big(F(\mathbf{X}, \mathbf{Y})|_{x_i=a} \wedge \bigwedge_{x_j \in \mathbf{X} \setminus (\mathbf{T} \cup \{x_i\})} (x_j \Leftrightarrow x_j') \wedge \mathsf{Fun}_{\mathbf{T}}(\mathbf{X}', \mathbf{Y})|_{x_i'=1-a}\Big) \Rightarrow F(\mathbf{X}', \mathbf{Y})|_{x_i'=1-a}$, where $a \in \{0, 1\}$ and $\mathbf{X}'$ is a sequence of fresh variables $(x_1', \ldots x_n')$. Informally, $\theta_{F,\mathbf{T},x_i,a}$ asserts that if the specification F can be satisfied by setting a non-FD output x_i to a, then it can also be satisfied by setting x_i to the complement value $(1 - a)$, while preserving the values of all other non-FD outputs. The FD outputs in $\mathbf{T}$ must of course be set as per the functional definitions in $\mathsf{Fun}_{\mathbf{T}}$.

Lemma 6. *Let $(\mathbf{T}, \mathsf{Fun}_{\mathbf{T}})$ be an acyclic system of f-defs in F.*

1) If $\mathbf{X} = \mathbf{T}$, then $\mathsf{Fun}_{\mathbf{T}} \preceq_{syn} F$.

2) If $\mathbf{X} \setminus \mathbf{T} \neq \emptyset$, then for every $x_i \in \mathbf{X} \setminus \mathbf{T}$, we have: If $\theta_{F,\mathbf{T},x_i,0}$ is a tautology, then $(x_i \wedge F|_{x_i=1}) \preceq_{syn} F$. Similarly, if $\theta_{F,\mathbf{T},x_i,1}$ is a tautology, then $(\neg x_i \wedge F|_{x_i=0}) \preceq_{syn} F$.

If $\mathbf{T} = \emptyset$, Lemma 6(2) simply reduces to Proposition 5(4). However, if $\mathbf{T} \neq \emptyset$ (as is often the case), Lemma 6(2) shows that $x_i \wedge F|_{x_i=1}$ (resp. $\neg x_i \wedge F|_{x_i=0}$) can refine F even if F is not positive (resp. negative) unate in x_i. As an illustration, the specification $G(x_1, x_2, y_1, y_2)$ in Example 3 is not unate in either x_1 or x_2. However, with $\mathbf{T} = \{x_1\}$ and $\mathsf{Fun}_{\mathbf{T}} \equiv (x_1 \Leftrightarrow (x_2 \vee y_1))$, we have $\theta_{F,\mathbf{T},x_2,0} \equiv 1$. Hence, $x_2 \wedge G|_{x_2=1} \equiv (x_1 \wedge x_2) \preceq_{syn} G$. When F is refined by an application of

Lemma 6(2), we say that F has been refined by *pivoting on* x_i.

Lemma 7. *Let* $(\mathbf{T}, \mathsf{Fun}_{\mathbf{T}})$ *and* $(\mathbf{T}', \mathsf{Fun}_{\mathbf{T}'})$ *be acyclic systems of f-defs in* F*, where* $\mathbf{T}' \subseteq \mathbf{T} \subseteq \mathbf{X}$ *and* $\mathsf{Fun}_{\mathbf{T}} \equiv \mathsf{Fun}_{\mathbf{T}'} \wedge \mathsf{Fun}_{\mathbf{T}\setminus\mathbf{T}'}$*. For* $a \in \{0, 1\}$*, if* $\theta_{F,\mathbf{T}',x_i,a}$ *is a tautology, then so is* $\theta_{F,\mathbf{T},x_i,a}$*.*

Lemma 7, along with Lemma 6(2), shows that if $\mathbf{T}' \subsetneq \mathbf{T} \subseteq \mathbf{X}$, the system of acyclic f-defs $(\mathbf{T}, \mathsf{Fun}_{\mathbf{T}})$ potentially provides more opportunities for refinement compared to $(\mathbf{T}', \mathsf{Fun}_{\mathbf{T}'})$. Hence, it is advantageous to augment the set $\mathbf{T}$ of FD outputs (and correspondingly $\mathsf{Fun}_{\mathbf{T}}$) whenever possible.

The following theorem shows that SynNNF is not too restrictive after all.

Theorem 8. *For every relational specification* $F(\mathbf{X}, \mathbf{Y})$*, there exists a polynomial-sized Skolem function vector for* $\mathbf{X}$ *in* F *iff there exists a* SynNNF *specification* $\widetilde{\mathsf{F}}(\mathbf{X}, \mathbf{Y})$ *such that* $\widetilde{\mathsf{F}} \preceq_{syn} F$ *and* $\widetilde{\mathsf{F}}$ *is polynomial-sized in* F*.*

Theorem 8 guarantees that whenever a polynomial-sized Skolem function vector exists for a specification $F(\mathbf{X}, \mathbf{Y})$, there is also a polynomial-sized refined specification in SynNNF. It is therefore interesting to ask if we can compile $F(\mathbf{X}, \mathbf{Y})$ to a "small enough" SynNNF specification $\widetilde{\mathsf{F}}(\mathbf{X}, \mathbf{Y})$ that refines F. In the next two sections, we present such a compilation algorithm and results of our preliminary experiments using this algorithm. Note that as shown in [1], there exist problem instances for which there are no polynomial-sized Skolem function vectors, unless the Polynomial Hierarchy (PH) collapses. Thus, any algorithm for compilation to SynNNF must incur super-polynomial blow-up (unless PH collapses). Nevertheless, as our experiments show, the compilation-based approach works reasonably well in practice, even solving benchmarks beyond the reach of existing state-of-the-art BFnS tools.

V. A Refining CNF to SynNNF Compiler

We now describe C2Syn – an algorithm that takes as input a CNF specification $F(\mathbf{X}, \mathbf{Y})$ given as a set of clauses, and outputs a DAG representation of a SynNNF specification $\widetilde{F}(\mathbf{X}, \mathbf{Y})$ that refines $\widetilde{F}(\mathbf{X}, \mathbf{Y})$ w.r.t. synthesis. Given a set $\mathcal{S}$ of clauses, we use $\varphi_{\mathcal{S}}$ to denote the formula $\bigwedge_{C_i \in \mathcal{S}} C_i$.

Let $\mathcal{S} = \{C_1, \ldots C_r\}$ be a set of clauses. Abusing notation introduced in Section II, let $atoms(C_i) = \{z \mid z \in \mathbf{X} \cup \mathbf{Y}, lits(C_i) \cap \{z, \neg z\} \neq \emptyset\}$. We define an undirected graph $G_{\mathcal{S}} = (V_{\mathcal{S}}, E_{\mathcal{S}})$, where $V_{\mathcal{S}} = \{C_1, \ldots C_r\}$ and $(C_i, C_j) \in E_{\mathcal{S}}$ iff $i \neq j$ and $atoms(C_i) \cap atoms(C_j) \cap \mathbf{X} \neq \emptyset$. Thus, there exists an edge (C_i, C_j) iff C_i and C_j share an output atom. Let $\{\mathcal{S}_1, \ldots \mathcal{S}_q\}$ be the set of maximally connected components (henceforth called MCCs) of $G_{\mathcal{S}}$. It is easy to see that $\varphi_{\mathcal{S}} \equiv \bigwedge_{k=1}^{q} \varphi_{\mathcal{S}_k}$; moreover, the output supports of $\varphi_{\mathcal{S}_k}$ for $k \in \{1, \ldots q\}$ are mutually disjoint. We use $C_i \sim_{\mathcal{S}} C_j$ to denote that clauses C_i and C_j are in the same MCC of $G_{\mathcal{S}}$. We will soon see how factoring $\varphi_{\mathcal{S}}$ based on MCCs of $G_{\mathcal{S}}$ allows us to decompose the CNF-to-SynNNF compilation problem into independent sub-problems, thanks to Proposition 5(5)b. Note that factoring based on MCCs has also been used in DSharp [20] for converting a CNF formula to dDNNF. However, unlike $G_{\mathcal{S}}$ above, the underlying graph in DSharp has an edge between every pair of clauses that shares any atom, including input variables. Thus, $G_{\mathcal{S}}$ has potentially fewer edges, and hence smaller MCCs, than the corresponding graph constructed by DSharp.

Algorithm 1: FDRefine

```
    Input: $\mathcal{S}$: set of clauses,  $(\mathbf{T}, \mathsf{Fun}_{\mathbf{T}})$: acyclic f-defs in $\varphi_{\mathcal{S}}$
    Output: $\mathcal{S}'$: set of clauses s.t. $\varphi_{\mathcal{S}'} \preceq_{syn} \varphi_{\mathcal{S}}$,
            $(\mathbf{T}', \mathsf{Fun}_{\mathbf{T}'})$: Augmented acyclic f-defs in $\varphi_{\mathcal{S}'}$
 1  $\mathbf{Out} := \mathsf{sup}(\varphi_{\mathcal{S}}) \cap \mathbf{X}$;
 2  $\mathcal{S}' := \mathcal{S}$;  $(\mathbf{T}', \mathsf{Fun}_{\mathbf{T}'}) := (\mathbf{T}, \mathsf{Fun}_{\mathbf{T}})$;          /* initialization */
 3  repeat
 4      $(\mathbf{T}', \mathsf{Fun}_{\mathbf{T}'})$ := FindFD($\mathcal{S}', \mathbf{T}', \mathsf{Fun}_{\mathbf{T}'}$);
 5      Let $F$ be the formula $\varphi_{\mathcal{S}'}$;
 6      foreach $x_i \in \mathbf{Out} \setminus \mathbf{T}'$ do
 7          if $\theta_{F,\mathbf{T}',x_i,0}$ is a tautology then
 8              $\mathcal{S}' := \mathcal{S}'|_{x_i=1} \cup \{x_i\}$;   $\mathbf{T}' = \mathbf{T}' \cup \{x_i\}$;
 9              $\mathsf{Fun}_{\mathbf{T}'} := \mathsf{Fun}_{\mathbf{T}'} \wedge (x_i \Leftrightarrow 1)$;
10          else if $\theta_{F,\mathbf{T}',x_i,1}$ is a tautology then
11              $\mathcal{S}' := \mathcal{S}'|_{x_i=0} \cup \{\neg x_i\}$;  $\mathbf{T}' = \mathbf{T}' \cup \{x_i\}$;
12              $\mathsf{Fun}_{\mathbf{T}'} := \mathsf{Fun}_{\mathbf{T}'} \wedge (x_i \Leftrightarrow 0)$;
13  until either $\mathbf{T}'$ or $\mathcal{S}'$ changes;
14  return $(\mathcal{S}', \mathbf{T}', \mathsf{Fun}_{\mathbf{T}'})$;
```

Before delving into Algorithm C2Syn, we first discuss some important sub-routines used in the algorithm. Sub-routine FDRefine takes as inputs a set $\mathcal{S}$ of clauses and a (possibly empty) acyclic system of f-defs $(\mathbf{T}, \mathsf{Fun}_{\mathbf{T}})$ in $\varphi_{\mathcal{S}}$. It returns a (possibly augmented) acyclic system of f-defs $(\mathbf{T}', \mathsf{Fun}_{\mathbf{T}'})$ and a set of clauses $\mathcal{S}'$ such that $\varphi_{\mathcal{S}'} \preceq_{syn} \varphi_{\mathcal{S}}$ and $\varphi_{\mathcal{S}'} \Rightarrow \mathsf{Fun}_{\mathbf{T}'}$. Sub-routine FDRefine works by iteratively finding new FD ouptut variables and refining the specification using Lemma 6(2) whenever possible. In the pseudo-code of FDRefine (see Algorithm 1), sub-routine FindFD matches a pre-defined set of clause-patterns in $\mathcal{S}'$ to identify new FD output variables not already in $\mathbf{T}'$. The patterns currently matched correspond to CNF encodings of the input-output relation of common Boolean functions, viz. and, or, nand, nor, xor, xnor, not and identity. For example, we match the pattern $(\neg\alpha \vee \beta_1) \wedge (\neg\alpha \vee \beta_2) \wedge (\neg\beta_1 \vee \neg\beta_2 \vee \alpha)$, where α, β_1, β_2 are place-holders, to identify the functional definition $(\alpha \leftrightarrow (\beta_1 \wedge \beta_2))$. Each new FD output variable thus identified is added to $\mathbf{T}'$ and the corresponding functional definition is added to $\mathsf{Fun}_{\mathbf{T}'}$ unless this introduces a cyclic dependency among the f-defs already in $\mathsf{Fun}_{\mathbf{T}'}$. Assuming all patterns used by FindFD to determine functional dependencies are sound, the (possibly augmented) $(\mathbf{T}', \mathsf{Fun}_{\mathbf{T}'})$ computed by FindFD is a system of acyclic f-defs in $\varphi_{\mathcal{S}'}$. In lines 6-12 of Algorithm 1, we next check if Lemma 6(2) can be applied to refine $\varphi_{\mathcal{S}'}$ by pivoting on some variable $x_i \in \mathbf{Out} \setminus \mathbf{T}'$. The refinement, if applicable, is easily done by replacing each clause $C_i \in \mathcal{S}'$ by $C_i|_{x_i=1}$ (resp. $C_i|_{x_i=0}$) and by adding the unit clause x_i (resp. $\neg x_i$) to $\mathcal{S}'$. The pivot x_i is also added to $\mathbf{T}'$ and the corresponding functional definition ($x_i \Leftrightarrow 1$ or $x_i \Leftrightarrow 0$ as the case may be) is added to $\mathsf{Fun}_{\mathbf{T}'}$.

In general, identifying an acyclic system of f-defs in F potentially enables refinement of F via Lemma 6(2), which in turn, can lead to augmenting the acyclic system of f-defs further. Therefore, the loop in lines 3-13 of Algorithm 1 is iterated until no new FD outputs or additional refinements are obtained. Once this happens, subroutine FDREFINE returns the resulting acyclic system of f-defs $(\mathbf{T}', \mathsf{Fun}_{\mathbf{T}'})$ and the resulting set of refined clauses $\mathcal{S}'$.

Two other important sub-routines used in C2Syn are GETCKT and GETDEFCKT. Sub-routine GETCKT takes as input an NNF formula $G(\mathbf{X}, \mathbf{Y})$ and returns the DAG representation of $G(\mathbf{X}, \mathbf{Y})$. Sub-routine GETDEFCKT takes as input a system of acyclic f-defs $(\mathbf{T}, \mathsf{Fun}_{\mathbf{T}})$, where $\mathbf{X} \cap \mathsf{sup}(\mathsf{Fun}_{\mathbf{T}}) = \mathbf{T}$ (i.e. $\mathbf{T}$ is the entire output support of $\mathsf{Fun}_{\mathbf{T}}$). It returns a DAG representation of a SynNNF specification equivalent to $\mathsf{Fun}_{\mathbf{T}}$. Without loss of generality, let $x_1 \sqsubset \ldots \sqsubset x_n$ be a linear ordering of the output variables in $\mathbf{T}$ such that the functional definition of x_i in $\mathsf{Fun}_{\mathbf{T}}$ does not depend on any x_j for $j \geq i$. Such an ordering always exists since $(\mathbf{T}, \mathsf{Fun}_{\mathbf{T}})$ is an acyclic system of f-defs. Let $x_i \Leftrightarrow \mathsf{op}_i(u_1, \ldots u_{n_i})$ be the functional definition of x_i in $\mathsf{Fun}_{\mathbf{T}}$, where op_i is a Boolean function identified via clause-pattern matching in sub-routine FINDFD. For each i in $\sqsubset$-order in $\{1, \ldots n\}$, we now construct a DAG $\mathcal{D}_i$ representing $\mathsf{op}_i(u_1, \ldots u_{n_i})$ in NNF. While constructing $\mathcal{D}_i$, we ensure that every $x_j \in \mathbf{T}$ that is also an argument of op_i is replaced by the root, say t_j, of the DAG $\mathcal{D}_j$. Since $(\mathbf{T}, \mathsf{Fun}_{\mathbf{T}})$ is an acyclic system of f-defs, this is always possible. Finally, we construct the overall DAG, say $\mathcal{D}$, representing $\bigwedge_{x_i \in \mathbf{T}} ((x_i \wedge t_i) \vee (\neg x_i \wedge \neg t_i))$. It is easy to see that for every $x_i \in \mathbf{T}$, there are no paths from x_i and $\neg x_i$ that meet for the first time at an $\wedge$-labeled node in $\mathcal{D}$. Abusing notation and using $\mathcal{D}$ to denote the specification represented by the above DAG, we therefore have $[\widehat{\mathcal{D}}]_i$ is $\wedge_i$-unrealizable for all $i \in \{1, \ldots n\}$; hence $\mathcal{D}$ is in SynNNF.

We are now in a position to describe Algorithm C2Syn. The algorithm is recursive and takes as inputs a set $\mathcal{S}$ of clauses, a (possibly empty) system of acyclic f-defs $(\mathbf{T}, \mathsf{Fun}_{\mathbf{T}})$ in $\varphi_{\mathcal{S}}$, and the recursion level ℓ. Initially, C2Syn is invoked with $\mathcal{S}$ = given set of CNF clauses, $\mathbf{T} = \emptyset$, $\mathsf{Fun}_{\mathbf{T}} = 1$ and $\ell = 0$. The pseudocode of C2Syn, shown in Algorithm 2, first computes the output support $\mathbf{Out}$ of φ_{S}, and then checks a few degenerate cases (lines 2-8) to determine if a refined SynNNF specification can be easily obtained. In case these checks fail, sub-routine FDREFINE is invoked to augment the set $\mathbf{T}'$ of functionally dependent outputs and their corresponding acyclic f-defs $\mathsf{Fun}_{\mathbf{T}'}$, and also to obtain a (possibly) refined set $\mathcal{S}'$ of clauses. If all outputs in $\mathbf{Out}$ get functionally determined by this, Lemma 6(1) guarantees that $\mathsf{Fun}_{\mathbf{Out}} \preceq_{syn} \varphi_{\mathcal{S}'}$; hence an invocation of GETDEFCKT($\mathbf{Out}, \mathsf{Fun}_{\mathbf{Out}}$) gives the desired result in line 12. Otherwise, we check in lines 14-17 if Theorem 3(2) can be applied. Recall that Theorem 3(2) relaxes the requirements of the SynNNF definition by requiring $\wedge_i$-unrealizability only when GACKS functions are substituted for the $\mathbf{X}$ variables. As discussed in Section III-3, the relaxed requirement can be checked by testing the unsatisfiability of the error formula ε for the GACKS function vector Ψ. If ε is indeed unsatisfiable, Ψ is a Skolem function vector for $\mathbf{Out}$ in $\varphi_{\mathcal{S}'}$, and hence $\bigwedge_{x_i \in \mathbf{Out}} (x_i \Leftrightarrow \Psi_i)$ refines $\varphi_{\mathcal{S}'}$.

Algorithm 2: C2Syn

```
    Input: $\mathcal{S}$: set of clauses, $(\mathbf{T}, \mathsf{Fun}_{\mathbf{T}})$: acyclic f-defs in $\varphi_{\mathcal{S}}$, $\ell$: recursion level
    Output: Boolean circuit for $\widetilde{\mathsf{F}}$ in SynNNF s.t. $\widetilde{\mathsf{F}} \preceq_{syn} \varphi_{\mathcal{S}}$
1   $\mathbf{Out} := \mathsf{sup}(\varphi_{\mathcal{S}}) \cap \mathbf{X}$;
2   if $\varphi_{\mathcal{S}}$ is valid (resp. inconsistent) then
3       return GETCKT(1) (resp. GETCKT(0));
4   else if $\varphi_{\mathcal{S}}$ is semantically independent of inputs $\mathbf{Y}$ then
5       Let $\pi$ be a satisfying assignment of $\varphi_{\mathcal{S}}$;
6       return GETCKT($\mathsf{form}(\pi{\downarrow}\mathbf{Out})$);
7   else if $\varphi_{\mathcal{S}}$ is semantically independent of $\mathbf{Out}$ then
8       return GETCKT(1);
9   else
10      $(\mathbf{T}', \mathsf{Fun}_{\mathbf{T}'}, \mathcal{S}')$ := FDREFINE($\mathcal{S}, \mathbf{T}, \mathsf{Fun}_{\mathbf{T}}$);
11      if $\mathbf{Out} \setminus \mathbf{T}' = \emptyset$ then
12          return GETDEFCKT($\mathbf{Out}, \mathsf{Fun}_{\mathbf{Out}}$);
13      else
14          Let $\Psi$ be GACKS Skolem function vector for $\mathbf{Out}$ in $\varphi_{\mathcal{S}'}$;
15          Let $\varepsilon$ :=
              $\varphi_{\mathcal{S}'}(\mathbf{Out}, \mathbf{Y}) \wedge \neg\varphi_{\mathcal{S}'}(\mathbf{Out}', \mathbf{Y}) \wedge \bigwedge_{x_i \in \mathbf{Out}} (x_i' \Leftrightarrow \Psi_i)$
16          if $\varepsilon$ is unsat then
17              return GETDEFCKT($\mathbf{Out}, \bigwedge_{x_i \in \mathbf{Out}} (x_i \Leftrightarrow \Psi_i)$);
18          $x$ := CHOOSEOUTPUTVAR($\mathcal{S}', \mathbf{Out} \setminus \mathbf{T}'$);
19          Pos := $\{C_j \in \mathcal{S}' \mid x \in lits(C_j)\}$;
20          Neg := $\{C_j \in \mathcal{S}' \mid \neg x \in lits(C_j)\}$;
21          $\mathcal{S}_1 := \{C_i \in \mathcal{S}' \mid \exists C_j \in \text{Pos } (C_i \sim_{\mathcal{S}'} C_j)\}$;
22          $\mathbf{T}_1 := \mathbf{T}' \cap \mathsf{sup}(\varphi_{\mathcal{S}_1})$;
23          $\mathcal{S}_2 := \{C_i \in \mathcal{S}' \mid \exists C_j \in \text{Neg } (C_i \sim_{\mathcal{S}'} C_j)\}$;
24          $\mathbf{T}_2 := \mathbf{T}' \cap \mathsf{sup}(\varphi_{\mathcal{S}_2})$;
25          $\mathcal{S}_3 := \{C_i \in \mathcal{S}' \mid \forall C_j \in \text{Pos} \cup \text{Neg } (C_i \not\sim_{\mathcal{S}'} C_j)$;
26          $\mathbf{T}_3 := \mathbf{T}' \cap \mathsf{sup}(\varphi_{\mathcal{S}_3})$;
27          Let $t_1$ := root of C2Syn($\mathcal{S}_1|_{x=0}, \mathbf{T}_1, \mathsf{Fun}_{\mathbf{T}_1}|_{x=0}, \ell + 1$);
28          Let $t_2$ := root of C2Syn($\mathcal{S}_2|_{x=1}, \mathbf{T}_2, \mathsf{Fun}_{\mathbf{T}_2}|_{x=1}, \ell + 1$);
29          Let $t_3$ := root of C2Syn($\mathcal{S}_3, \mathbf{T}_3, \mathsf{Fun}_{\mathbf{T}_3}, \ell + 1$);
30          return GETCKT($t_3 \wedge ((x \wedge t_2) \vee (\neg x \wedge t_1))$)
```

If ε is satisfiable, we use a sub-routine CHOOSEOUTPUTVAR that heuristically chooses an output variable $x \in \mathbf{Out} \setminus \mathbf{T}'$ on which to branch. Currently, we use a VSIDS [19] score based heuristic, similar to that used in DSHARP [20], to rank variables in $\mathbf{Out} \setminus \mathbf{T}'$, and then choose the variable with the highest score. This allows us to represent $\varphi_{\mathcal{S}'}$ as $x_i \wedge \varphi_{\mathcal{S}'}|_{x=1} \vee \neg x_i \wedge \varphi_{\mathcal{S}'}|_{x=0}$, so that we can refine the two disjuncts independently, thanks to Proposition 5(5)a. However, this may lead to some duplicate processing of clauses. We can avoid this by factoring out the subset of clauses whose satisfiability is independent of whether x_i is set to 1 or 0. Let $\mathcal{S}_1$ (resp. $\mathcal{S}_2$) be the subset of clauses in $\mathcal{S}'$ that are in the same MCC of $G_{\mathcal{S}'}$ as some C_j that has x (resp. $\neg x$) as a literal. Let $\mathcal{S}_3$ be the set of all clauses in $\mathcal{S}'$ that are neither in $\mathcal{S}_1$ nor $\mathcal{S}_2$. By definition of $G_{\mathcal{S}'}$, the sub-specifications $\varphi_{\mathcal{S}_1}$ and $\varphi_{\mathcal{S}_3}$ (and similarly, $\varphi_{\mathcal{S}_1}$ and $\varphi_{\mathcal{S}_3}$) do not share any output variable in their supports, and can be refined independently. This is exactly what algorthm C2Syn does in lines 19-30. The outputs of the circuits resulting from the recursive calls in lines 27, 28 and 29 are finally combined as in line 30 to yield the desired circuit.

Theorem 9. *For every set $\mathcal{S}$ of clauses, C2Syn$(\mathcal{S}, \emptyset, 1, 0)$ always terminates and returns a Boolean circuit implementing a SynNNF specification $\widetilde{\mathsf{F}}$ such that $\widetilde{\mathsf{F}} \preceq_{syn} \varphi_{\mathcal{S}}$.*

The proof is straightforward and can be found in the Appendix.

VI. Experimental results

We ran Algorithm C2Syn on a suite of CNF specifications comprised of benchmarks from the Prenex 2QBF track of QBFEval 2018 [21], and the .qdimacs version of Factorization benchmarks [1], which we will refer to as FA.QD. By Theorem 2(i), a ROBDD/FBDD specification can be compiled in a straightforward way to an equivalent SynNNF specification in linear time. Therefore, any algorithm that compiles a CNF specification to an ROBDD can be viewed as an alternative to C2Syn for compiling a CNF specification to SynNNF (albeit without refinement). We compare the performance of C2Syn with that of a BDD compiler and two state-of-the-art boolean function synthesis tools, namely, (i) the AIG-NNF pipeline of bfss [1] with ABC's MiniSat as the SAT solver and (ii) Cadet [22], [24]. For the BDD Compiler, the .qdimacs input was converted to an AIG using simple Tseitin variable detection; this AIG was then simplified and ROBDDs built using dynamic variable ordering (of all input and output variables) – this is part of the BDD pipeline of bfss [1], henceforth called BDD$^{\text{BFSS}}$. We also ran DSharp [20] which compiles a CNF formula into dDNNF (and hence, into SynNNF by Theorem 2(i)), but it was successful on very few of our benchmarks; hence we do not present its performance. Each tool took as input the same .qdimacs file. Experiments were performed on a cluster with 20 cores and 64 GB memory per node, each core being a 2.2 GHz Intel Xeon processor running CentOS6.5. Each run was performed on a single core, with timeout of 1 hour and main memory limited to 16GB.

For C2Syn runs, we noticed that quite a few problems were solved in recursion level 0 itself, either in lines 10-12 or in lines 14-17 of Algorithm 2. We therefore identify 3 stages of C2Syn runs: lines 10-12 of recursion level 0 (Stage-I), lines 14-17 of recursion level 0 (Stage-II), and the rest of C2Syn (Stage-III). Table I compares the overall results for C2Syn and BDD$^{\text{BFSS}}$. For QBFEval, C2Syn solved a total of 181 benchmarks, wherein 61 were solved by Stage-I, 41 by Stage-II and 79 were solved by Stage-III. BDD$^{\text{BFSS}}$ could completely compile for 153 benchmarks. For FA.QD, both C2Syn (Stage-III) and BDD$^{\text{BFSS}}$ solved all six. Since BDDs are also in SynNNF, the total number of benchmarks in QBFEval which could be compiled into SynNNF (benchmarks solved by either compiler) is a whopping 281!

Bench mark	No.of Bench-marks	Compiled By C2Syn				BDD comp-lation	Total in SynNNF
		Stage I	Stage II	Stage III	Total		
QBFEval	402	61	41	79	181	153	**281**
FA.QD	6	0	0	6	6	6	**6**

TABLE I: Compilation into SynNNF

Figure 1 compares the run-times of C2Syn and BDD$^{\text{BFSS}}$: for most QBFEval benchmarks that were solved by both, C2Syn took less time, while for FA.QD, C2Syn took more time. There were 125 QBFEval benchmarks that C2Syn solved by BDD$^{\text{BFSS}}$ couldn't, whereas there were 100 benchmarks that BDD$^{\text{BFSS}}$ solved but C2Syn couldn't. This indicates that the two approaches to SynNNF compilation have orthogonal strengths.

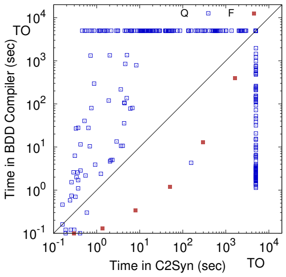

Fig. 1: Performance of C2Syn and BDD$^{\text{BFSS}}$

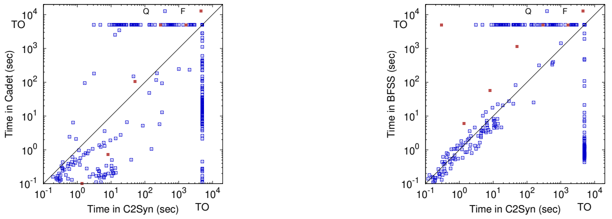

Fig. 2: Comparison of C2Syn with Cadet and bfss

We next compare the performance of C2Syn with those of Cadet and bfss. Cadet (resp. bfss) solved 213 (resp. 181) benchmarks in QBFEval and 4 (resp. 3) in FA.QD respectively. The comparison in terms of how many benchmarks were solved by each tool but not by other tools is given in Table II.

Bench mark	C2Syn vs Cadet		C2Syn vs bfss		C2Syn \ (Cadet ∪ bfss)
	C2Syn\ Cadet	Cadet\ C2Syn	C2Syn\ bfss	bfss\ C2Syn	
QBFEval	74	106	78	78	**71**
FA.QD	2	0	3	0	**2**

TABLE II: Comparison Results of C2Syn

Figure 2 compares the run-times of C2Syn and those of Cadet and bfss. As expected, since C2Syn does complete compilation, it takes more time than Cadet and marginally more than bfss on many benchmarks but for most of these benchmarks, the time taken is less than a minute. In fact for FA.QD, C2Syn takes less time than bfss on all benchmarks. Overall, C2Syn appears to have strengths orthogonal to BDD$^{\text{BFSS}}$, bfss and Cadet, and therefore adds to the repertoire of state-of-the-art tools for Boolean functional synthesis.

VII. Conclusion

We presented a new class of negation normal formulas called SynNNF that admit quadratic-time synthesis and linear-time existential quantification of a set of variables. Our prototype compiler is able to several benchmarks that cannot be handled by other state-of-the-art tools. Since representations like ROBDDs, DNNF and the like are either already in or easily transformable to SynNNF, our work is widely applicable and can be used in tandem with other techniques.

References

[1] S. Akshay, Supratik Chakraborty, Shubham Goel, Sumith Kulal, and Shetal Shah. What's Hard About Boolean Functional Synthesis? In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, pages 251–269, 2018.

[2] S. Akshay, Supratik Chakraborty, Ajith K. John, and Shetal Shah. Towards Parallel Boolean Functional Synthesis. In *TACAS 2017 Proceedings, Part I*, pages 337–353, 2017.

[3] G. Boole. *The Mathematical Analysis of Logic*. Philosophical Library, 1847.

[4] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, August 1986.

[5] Randal E. Bryant. On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Trans. Computers*, 40(2):205–213, 1991.

[6] Marco Cadoli and Francesco M. Donini. A survey on knowledge compilation. *AI Commun.*, 10(3-4):137–150, 1997.

[7] Supratik Chakraborty, Dror Fried, Lucas M. Tabajara, and Moshe Y. Vardi. Functional synthesis via input-output separation. In *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018*, pages 1–9, 2018.

[8] Supratik Chakraborty, Dror Fried, Lucas M. Tabajara, and Moshe Y. Vardi. Functional synthesis via input-output separation. In *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018*, pages 1–9, 2018.

[9] Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001.

[10] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *CoRR*, abs/1106.1819, 2011.

[11] Dror Fried, Lucas M. Tabajara, and Moshe Y. Vardi. BDD-based boolean functional synthesis. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, pages 402–421, 2016.

[12] J.-H. R. Jiang. Quantifier elimination via functional composition. In *Proc. of CAV*, pages 383–397. Springer, 2009.

[13] J.-H. R. Jiang and V Balabanov. Resolution proofs and Skolem functions in QBF evaluation and applications. In *Proc. of CAV*, pages 149–164. Springer, 2011.

[14] A. John, S. Shah, S. Chakraborty, A. Trivedi, and S. Akshay. Skolem functions for factored formulas. In *FMCAD*, pages 73–80, 2015.

[15] V. Kuncak, M. Mayer, R. Piskac, and P. Suter. Complete functional synthesis. *SIGPLAN Not.*, 45(6):316–329, June 2010.

[16] Jérôme Lang, Paolo Liberatore, and Pierre Marquis. Propositional independence - formula-variable independence and forgetting. *CoRR*, abs/1106.4578, 2011.

[17] L. Lowenheim. Über die Auflösung von Gleichungen in Logischen Gebietkalkul. *Math. Ann.*, 68:169–207, 1910.

[18] Martina Seidl Marijn Heule and Armin Biere. Efficient Extraction of Skolem Functions from QRAT Proofs. In *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014*, pages 107–114, 2014.

[19] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, pages 530–535, New York, NY, USA, 2001. ACM.

[20] Christian Muise, Sheila A. McIlraith, J. Christopher Beck, and Eric Hsu. DSHARP: Fast d-DNNF Compilation with sharpSAT . In *AAAI-16 Workshop on Beyond NP*, 2016.

[21] QBFLib. QBFEval 2018. http://www.qbflib.org/qbfeval18.php.

[22] M. N. Rabe and S. A. Seshia. Incremental determinization. In *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, pages 375–392, 2016.

[23] M. N. Rabe and L. Tentrup. CAQE: A certifying QBF solver. In *Formal Methods in Computer-Aided Design, FMCAD 2015, Austin, Texas, USA, September 27-30, 2015.*, pages 136–143, 2015.

[24] Markus N. Rabe, Leander Tentrup, Cameron Rasmussen, and Sanjit A. Seshia. Understanding and extending incremental determinization for 2qbf. In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, pages 256–274, 2018.

[25] Lucas M. Tabajara and Moshe Y. Vardi. Factored boolean functional synthesis. In *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017*, pages 124–131, 2017.

[26] G. S. Tseitin. On the complexity of derivation in propositional calculus. *Structures in Constructive Mathematics and Mathematical Logic, Part II, Seminars in Mathematics*, pages 115–125, 1968.

[27] L. G. Valiant. Completeness classes in algebra. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 249–261, New York, NY, USA, 1979. ACM.

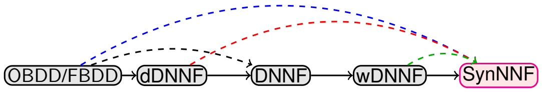

Fig. 3: An edge $A \rightarrow B$ means that A is a proper subset of B. A blue edge from A to B means B is exponentially more succinct than A, while a red edge from A to B means that unless $\mathsf{P} = \mathsf{VNP}$, B is super-polynomially more succinct than A. The green edge from A to B means that unless $\mathsf{P} = \mathsf{NP}$, B is super-polynomially more succinct than A. The black edge is the exponential succinctness of DNNF w.r.t FBDD[9].

Appendix

A. Proof of Theorem 2 of Section III

This section is dedicated to the proof of Theorem 2. We show that SynNNF is a space-efficient DAG-based representation of boolean functions, when compared with other representations using FBDD, DNNF and dDNNF.

First, observe that Part(i) is easy. That is, it has been shown, e.g., in [9] that FBDDcan be converted to DNNF with a linear complexity blowup. Now, focussing on dDNNF, DNNF, wDNNF, an examination of their definitions immediately gives us that each of these forms is already in SynNNF. Further, from the definition again it is clear that dDNNFis subsumed by DNNF, which is further subsumed by wDNNF, as depicted in Figure 3. To show strictness, it suffices to consider Example 1, which is in SynNNF but not in wDNNF since x_2 and $\neg x_2$ indeed meet up at an $\wedge$-node in G. This completes Part (i).

For part (ii), we start by noting that it has been shown in [9] that the DNNF representation is exponentially more succinct than FBDD. We now show that SynNNF is super-polynomially (resp. exponentially) more succinct than dDNNF, DNNF and wDNNF (resp. FBDD) representations, unless some long-standing complexity conjectures are falsified. To do this, we describe a family of specifications having a polynomial sized the SynNNF representation, but for which the other representations are necessarily super-polynomially larger, unless these complexity conjectures are falsfied.

Consider the family $F(\mathbf{X}, \mathbf{Y})$ of specifications defined as follows. Let $\mathbf{X} = \{x_1, \ldots, x_n\}$, and let $f_i(\mathbf{X}^n_{i+1}, \mathbf{Y})$, $1 \leq i \leq n-1$ be arbitrary boolean functions in NNF over $x_{i+1}, \ldots, x_n, \mathbf{Y}$. We define the family $F(\mathbf{X}, \mathbf{Y})_{(\mathsf{op}'_1, \mathsf{op}_1, \ldots, \mathsf{op}'_n, \mathsf{op}_n)}$, parametrized by $\mathsf{op}_i \in \{\wedge, \vee\}$, and $\mathsf{op}'_i \in \{\wedge, \vee, \oplus\}$ as

$$(x_1 \mathsf{op}'_1 f_1(\mathbf{X}^n_2, \mathbf{Y}))\mathsf{op}_1(x_2 \mathsf{op}'_2 f_2(\mathbf{X}^n_3, \mathbf{Y}))\mathsf{op}_2 \ldots \mathsf{op}_{n-1}(x_n \mathsf{op}'_n f_n(\mathbf{Y}))\mathsf{op}_n f_{n+1}(\mathbf{Y})$$

Lemma 10. *Let g be a function in the family of specifications $F(\mathbf{X}, \mathbf{Y})_{(\mathsf{op}'_1, \mathsf{op}_1, \ldots, \mathsf{op}'_n, \mathsf{op}_n)}$. Then*
1) *If $\mathsf{op}'_1 = \cdots = \mathsf{op}'_n = \vee$, then g is in SynNNF.*
2) *If $\mathsf{op}'_1 = \cdots = \mathsf{op}'_n = \oplus$, then g is in SynNNF.*

Proof. 1) Let g be any function in the family with all the $\mathsf{op}'_i = \vee$. Figure 4 depicts the DAG representation of g. It is easy to see that g is in SynNNF, using the sufficient condition in Section III. That is, in $[\widehat{g}]_1$, there is no $\overline{x_1}$, so we never have a x_1 and $\overline{x}_1$ meeting at the root. Further, $[\widehat{g}]_2$ after replacing x_1 with 1, the leftmost subtree rooted at $\vee$ having children x_1, f_1 is no longer there after constant propagation. In the rest of the tree, we have no occurrences of $\overline{x}_2$, hence no way for x_2 and $\overline{x}_2$ to meet at the root. Thus, for each $[\widehat{g}]_i$, the argument is similar, since on replacing $x_1, \ldots, x_{i-1}$ with 1 and doing constant propagation, the remaining DAG will not have x_{i+1} and $\overline{x}_{i+1}$ together, which shows that g is in SynNNF.

2) Let g be any function in the family with all the $\mathsf{op}'_i = \oplus$. Note that in this case, we cannot use the sufficient condition as above, since in the rightmost DAG in Figure 4, clearly, $x_1, \overline{x}_1$ meet at a $\wedge$ in $[\widehat{g}]_1$. Nevertheless g is in SynNNF,

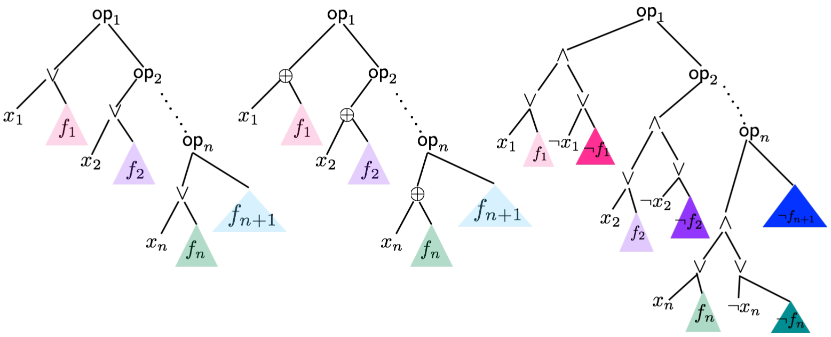

Fig. 4: On the left and center are families in SynNNF form. The rightmost figure is obtained by opening up the $\oplus$s (i.e., XOR-gates).

if we consider $[\widehat{g}]_i$ for $1 \leq i \leq n$, and consider the root node with children $\alpha_1 = x_i \vee f_i$ and $\alpha_2 = \overline{x}_i \vee \neg f_i$, after substituting $x_1, \ldots, x_{i-1}, \overline{x}_1, \ldots, \overline{x}_{i-1}$ to 1, $\overline{x}_{i+1}, \ldots, \overline{x}_n$ to $\neg x_{i+1}, \ldots, \neg x_n$, and constant propagation, it is easy to see that $\alpha_1^{11} \wedge \alpha_2^{11} = 1 \ \mathsf{op}_1 G$, $\alpha_1^{10} \wedge \alpha_2^{10} = \neg f_i \mathsf{op}_1 G$ and $\alpha_1^{01} \wedge \alpha_2^{01} = f_i \mathsf{op}_1 G$ for $\mathsf{op}_1 \in \{\vee, \wedge\}$ and some G. Thus $((\alpha_1^{11} \wedge \alpha_2^{11}) \wedge \neg(\alpha_1^{10} \wedge \alpha_2^{10}) \wedge \neg(\alpha_1^{01} \wedge \alpha_2^{01}))$ is unsatisfiable. Thus, g is $\wedge_i$ unrealizable for any i.

□

Theorem 11 (Restatement of Theorem 2(ii))**.** *(a) There are functions which admit polynomial sized* SynNNF *representations, yet admit only exponential sized* FBDD *representations.*

(b) Unless P = VNP*, there are functions which admit polynomial sized* SynNNF *representations, yet admit only super-polynomial sized* dDNNF *representations.*

(c) Unless P = NP*, there are functions which admit polynomial sized* SynNNF *representations, yet admit only super-polynomial sized* wDNNF *and* DNNF *representations.*

Proof. We use the family of specifications $F(\mathbf{X}, \mathbf{Y})$ defined above, with different instantiations to obtain all three results. Set $\mathsf{op}_1 = \cdots = \mathsf{op}_n = \wedge$, $\mathsf{op}'_1 = \cdots = \mathsf{op}'_n = \vee$, $f_i(\mathbf{X}^n_{i+1}, \mathbf{Y}) = \top$ for $2 \leq i \leq n$, obtaining $g = x_1 \vee f_1(\mathbf{X}^n_2, \mathbf{Y})$. Let $\mathbf{Y} = \{y_1, \ldots, y_{n-1}\}$. As seen in Lemma 10, g is in SynNNF. In each of the subparts below, we define $f_1(\mathbf{X}^n_2, \mathbf{Y})$ appropriately.

Item (a): **Succinctness w.r.t** FBDD. Let $k = n - 1$. We use the k-bit multiplier function over $\{x_2, \ldots, x_n\} \cup \{y_1, \ldots, y_{n-1}\}$ in the construction of $f_1(\mathbf{X}^n_2, \mathbf{Y})$. The two k bit arguments to the multiplier are respectively, $\{x_2, \ldots, x_n\}$ and $\{y_1, \ldots, y_{n-1}\}$ with x_n, y_{n-1} being the most significant bits, and x_2, y_1 being the least significant bits. Let $f_1(\mathbf{X}^n_2, \mathbf{Y})$ be the boolean function representing the kth bit of the k-bit multiplier function. The size of $f_1(\mathbf{X}^n_2, \mathbf{Y})$ is quadratic in k, since the size of any multiplier circuit consisting of $\vee, \wedge$ gates is quadratic in k (sum of k^2 partial products). For this $f_1(\mathbf{X}^n_2, \mathbf{Y})$, the size of g is $\mathcal{O}(k^2 + 1)$.

Let rep_1 be a representation of g using FBDD, by fixing a certain variable order. Set x_1=0. This assignment makes $g = f_1(\mathbf{X}^n_2, \mathbf{Y})$. Indeed, the FBDD representation obtained as a restriction of rep_1 with respect to this truth assignment is simpler [4]. It is known [5] that any FBDD, OBDD representations for $f_1(\mathbf{X}^n_2, \mathbf{Y})$ is exponential in k. This establishes the exponential succinctness of SynNNF over FBDD.

Item (b): **Succinctness w.r.t dDNNF**. We use a CNF encoding of the perfect matchings of a bipartite graph G (denoted $\mathsf{pm}(G)$) in the construction of $f_1(\mathbf{X}^n_2, \mathbf{Y})$. Given a bipartite graph G with two parts $U = \{u_1, \ldots, u_m\}$ and $V = \{v_1, \ldots, v_m\}$, we can define a 0-1 matrix $A = (a_{ij}), 1 \leq i, j \leq m$ such that $a_{ij} = 1$ iff there is an edge between $u_i \in U$ and $v_j \in V$. It is easy to see from the definition of the permanent of A (denoted $\mathsf{perm}(A)$) that $\mathsf{perm}(A) = \mathsf{pm}(G)$. Likewise, the number of perfect matchings of a bipartite graph is the permanent of its incidence matrix. Set $f_1(\mathbf{X}^n_2, \mathbf{Y})$ as the function which encodes $\mathsf{pm}(G)$.

Let rep_2 be the dDNNF representation of g. As in the first case, choose an assignment x_1=0 obtaining $g = 0 \vee f_1(\mathbf{X}^n_2, \mathbf{Y}) = f_1(\mathbf{X}^n_2, \mathbf{Y})$. Then it can be seen that the number of solutions of f_1 is exactly the number of perfect matchings of the bipartite graph G. Fixing the assignment x_1=0 results in a simpler dDNNF representation (say rep_3) for g (now f_1). Counting the models of rep_3 can be done in time polynomial in the size of rep_3 [10]. This implies that we can find the number of perfect matchings of the underlying bipartite graph G in time polynomial in the size of rep_3. Unless P = VNP, rep_3 cannot have a polynomial representation, since otherwise, we would obtain a polynomial time solution for computing $\mathsf{perm}(A)$. This shows the super-polynomial succinctness of SynNNF over dDNNF, unless P = VNP.

Item(c): **Succinctness w.r.t wDNNF and DNNF** Let $\mathsf{op}'_1 = \cdots = \mathsf{op}'_n = \vee$, $f_i(\mathbf{X}^n_{i+1}, \mathbf{Y}) = \top$ for $2 \leq i \leq n$, obtaining $g = x_1 \vee f_1(\mathbf{X}^n_2, \mathbf{Y})$. As shown in Lemma 10, g is in SynNNF, where $f_1(\mathbf{X}^n_2, \mathbf{Y})$ is an arbitrary SAT formula. If we can obtain a poly-sized DNNF representation for the function g, then using the assignment $x_1 = 0$ in g, we obtain a DNNF representation for $f_1(\mathbf{X}^n_2, \mathbf{Y})$. But it is known [10] that consistency checking is poly-time for DNNF representations. A polynomial sized DNNF representation for g would imply a polynomial time solution for the satisfiability checking of an arbitrary SAT formula. Thus, unless P = NP, any DNNF representation for g will necessarily be super polynomial. □

This completes the proof of Part (ii) of Theorem 2.

Part(iii). By Theorem 1 of [1], we know that there exist instances of poly-sized NNF formulas whose Skolem functions are necessarily super-polynomial size (resp. exponential) unless the polynomial hierarchy collapses (resp. the non-uniform exponential hypothesis is falsified). For any such instance, suppose we were able to obtain a poly-sized SynNNF representation, then by Theorem 1, we will be able to synthesize polynomial-sized Skolem functions, which contradicts the above.

To see a concrete example where SynNNF is not likely to be succinct, we refer to Theorem 1 of [1], where a constructive reduction of the parameterized clique problem to BFnS was given. The specification, in this case, has a polynomial-sized representation, but unless some long-standing complexity-theory conjectures are violated, it was shown that any Skolem function must have exponential/super-polynomial size. Thus, unless these conjectures are violated, the same specification in SynNNF must also be exponential/super-polynomial sized.

This proves Part (iii) and completes the proof of this theorem.

Essentially this means that though we obtain succinctness with respect to several known forms (using classical complexity-theoretic results), it is not the case that SynNNF will always be able to produce a poly-sized representation.

B. Proof of Theorem 3

Let us recall the characterization theorem from Section III.

Theorem 3. *Given a relational specification $F(\mathbf{X}, \mathbf{Y})$,*

1) *F is in SynNNF iff $\exists \mathbf{X}_1^i F(\mathbf{X}, \mathbf{Y}) \Leftrightarrow [\widehat{F}]_{i+1}[\overline{\mathbf{X}}^n_{i+1} \mapsto \neg \mathbf{X}^n_{i+1}]$*
2) *The GACKS-function vector Ψ_1^n is a Skolem function vector for $\mathbf{X}_1^n$ in $F(\mathbf{X}, \mathbf{Y})$ iff $[\widehat{F}]_i[\mathbf{X}^n_{i+1} \mapsto \Psi^n_{i+1}, \overline{\mathbf{X}^n_{i+1}} \mapsto \neg \Psi^n_{i+1}]$ is $\wedge_i$-unrealizable for all $i \in \{1 \dots n\}$.*

Proof. Part 1): The forward direction follows from Theorem 1. For the reverse direction, we will prove the contrapositive: if F is not in SynNNF, i.e., if $[\widehat{F}]_i[\mathbf{X}^n_{i+1} \mapsto \neg \mathbf{X}^n_{i+1}]$ is not $\wedge_i$-unrealizable for some $i \in \{1 \dots n\}$, we will show that for some i, $\exists \mathbf{X}_1^i F(\mathbf{X}, \mathbf{Y}) \not\Leftrightarrow [\widehat{F}]_{i+1}[\overline{\mathbf{X}}^n_{i+1} \mapsto \neg \mathbf{X}^n_{i+1}]$. Fix any $\mathbf{Y} \in \{\mathbf{Y}' \mid \exists \mathbf{X}', F(\mathbf{X}', \mathbf{Y}')\}$, i.e., it is a realizable valuation of inputs. Consider i to be the largest index such that $[\widehat{F}]_i$ is not $\wedge_i$-unrealizable, i.e., the corresponding ζ is satisfiable. As a result, we have $\alpha^{11} = 1$, i.e., $[\widehat{F}]_{i+1}[\overline{\mathbf{X}}^n_{i+1} \mapsto \neg \mathbf{X}^n_{i+1}] = 1$. On the other hand $\alpha^{01} = \widehat{F}(1^{i-1}, 0, \mathbf{X}^n_{i+1}, 1^{i-1}, 1, \neg \mathbf{X}^n_{i+1}, \mathbf{Y}) = 0$ and $\alpha^{10} = \widehat{F}(1^{i-1}, 1, \mathbf{X}^n_{i+1}, 1^{i-1}, 0, \neg \mathbf{X}^n_{i+1}, \mathbf{Y}) = 0$. By monotonicity, every assignment of $x_1, \dots x_{i-1}, x_i$ will also result in 0 in $\widehat{F}$, which implies that $\exists \mathbf{X}_1^i F(\mathbf{X}, \mathbf{Y}) = 0$. Thus for this i, $\exists \mathbf{X}_1^i F(\mathbf{X}, \mathbf{Y}) \not\Leftrightarrow [\widehat{F}]_{i+1}[\overline{\mathbf{X}}^n_{i+1} \mapsto \neg \mathbf{X}^n_{i+1}]$, which completes the proof.

Part 2): **Forward direction:** We will prove the contrapositive, i.e., if $[\widehat{F}]_i[\mathbf{X}^n_{i+1} \mapsto \Psi^n_{i+1}]$ is not $\wedge_i$-unrealizable for some $i \in \{1 \dots n\}$, we will show that Ψ_1^n is not a correct Skolem function vector for $\mathbf{X}_1^n$ in $F(\mathbf{X}, \mathbf{Y})$. Fix any $\mathbf{Y} \in \{\mathbf{Y}' \mid \exists \mathbf{X}', F(\mathbf{X}', \mathbf{Y}')\}$, i.e., it is a realizable valuation of inputs. Consider i to be the largest index such that $[\widehat{F}]_i[\mathbf{X}^n_{i+1} \mapsto \Psi^n_{i+1}, \overline{\mathbf{X}}^n_{i+1} \mapsto \neg \Psi^n_{i+1}]$ is not $\wedge_i$-unrealizable, i.e., the corresponding ζ is satisfiable.

We claim that one of the Ψ^n_{i+1} must be an incorrect skolem function for this $\mathbf{Y}$. Suppose not, i.e., suppose all of them are correct. Then we have

$$\exists x_1, \dots, x_i \widehat{F}(x_1, \dots, x_i, \Psi^n_{i+1}, \neg x_1, \dots \neg x_i, \neg \Psi^n_{i+1}, \mathbf{Y}) = 1 \tag{1}$$

However, because at i, ζ is satisfiable, we have $\widehat{F}(1^{i-1}, 0, \Psi^n_{i+1}, 1^{i-1}, 1, \neg \Psi^n_{i+1}, Y) = 0$ and $\widehat{F}(1^{i-1}, 1, \Psi^n_{i+1}, 1^{i-1}, 0, \neg \Psi^n_{i+1}, Y) = 0$. By monotonicity, every assignment of $x_1, \dots x_{i-1}$ will also result in 0 in $\widehat{F}$. But this contradicts (1). Hence all the skolem functions cannot be correct for this $\mathbf{Y}$, proving the forward direction.

Reverse direction: Again, we prove by taking the contrapositive. Suppose, Ψ^n_{i+1} is not a correct Skolem function vector. In [14], it was shown that for any function vector φ_1^n, it is a Skolem function vector for F iff the *error formula* $\varepsilon_\varphi \equiv F(\mathbf{X}, \mathbf{Y}) \wedge \neg F(\mathbf{X}', \mathbf{Y}) \wedge \bigwedge_{i=1}^n (x_i' \leftrightarrow \varphi_i)$ is unsatisfiable. We will use this characterization now, i.e., since Ψ^n_{i+1} is not a correct Skolem function vector, the error formula ε_Ψ must be satisfiable.

Hence, we start by considering $\mathbf{Y}^*$ which gives a satisfying assignment for the error formula ε_Ψ. That is,

$$\exists \mathbf{X}' F(\mathbf{X}', \mathbf{Y}^*) \wedge \exists 1 \leq i \leq n \ \neg \exists \mathbf{X}_1^{i-1} F(\mathbf{X}_1^{i-1}, \Psi_i^n(\mathbf{Y}^*), \mathbf{Y}^*) \tag{2}$$

Let k be the highest such i such that the above statement holds. That is, after k, the Skolem functions given by Ψ are correct, and at k they are incorrect. Then, we observe that the value at k must be 1, i.e.,

$$\Psi_k(\mathbf{Y}^*) = \widehat{F}(\mathbf{1}^{k-1}, 1, \Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{1}^{k-1}, 0, \neg \Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{Y}^*) = 1 \tag{3}$$

To see this, observe that $\exists \mathbf{X}' F(\mathbf{X}', \mathbf{Y}^*)$ along with maximality of k implies that $\exists \mathbf{X}_1^k F(\mathbf{X}_1^k, \Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{Y}^*) = 1$, which in turn implies that

$$\widehat{F}(\mathbf{1}^{k-1}, 1, \Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{1}^{k-1}, 0, \neg \Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{Y}^*) \vee \widehat{F}(\mathbf{1}^{k-1}, 0, \Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{1}^{k-1}, 1, \neg \Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{Y}^*) = 1$$

Now, if $\Psi_k(\mathbf{Y}^*) = 0$, this implies $\widehat{F}(\mathbf{1}^{k-1}, 0, {\psi'}^n_{k+1}(\mathbf{Y}^*), \mathbf{1}^{k-1}, 1, \neg {\psi'}^n_{k+1}(\mathbf{Y}^*), \mathbf{Y}^*) = 1$. But then, setting $x_k = 1$ is indeed correct, which would imply that there is no error at k, which violates the assumption on k. Thus we must have $\Psi_k(\mathbf{Y}^*) = 1$.

Now, we know that this is an incorrect assignment to x_k, which implies that the correct assignment is a 0 and we know that $\exists \mathbf{X}_1^{k-1} F(\mathbf{X}_1^{k-1}, 1, \Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{Y}^*)$ is a correct assignment to x_k. Hence, we must have

$$\begin{aligned} \exists \mathbf{X}_1^{k-1} F(\mathbf{X}_1^{k-1}, 1, \Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{Y}^*) = 0 \\ \Rightarrow \exists \mathbf{X}_1^{k-1} \widehat{F}(\mathbf{X}_1^{k-1}, 1, \Psi^n_{k+1}(\mathbf{Y}^*), \neg \mathbf{X}_1^{k-1}, 0, \neg \Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{Y}^*) = 0 \end{aligned} \tag{4}$$

The fact that equations (3), (4) hold together imply that the Skolem function Ψ is wrong *at level* k, since it gives value 1, but fixing $x_k = 1$, there is no way to set lower variables to get 1. The rest of the proof is a careful case-analysis, where we either show that ζ (with Skolem functions assigned according to Ψ) at level k is satisfiable, i.e., $[\widehat{F}]_{k+1}[\mathbf{X}^n_{k+1} \mapsto \Psi^n_{k+1}]$ is not

$\wedge_k$-unrealizable and hence the proof terminates, or we show that these equations are satisfied at a lower level (i.e., there is an error at a lower level). Since number of levels is finite this procedure will terminate. We describe the different cases now:

- Case 1: The first case is if

$$\widehat{F}(\mathbf{1}^{k-2}, 1, 1, \Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{1}^{k-2}, 0, 0, \neg\Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{Y}^*) = 0 \tag{5}$$

$$\text{and}\widehat{F}(\mathbf{1}^{k-2}, 0, 1, \Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{1}^{k-2}, 1, 0, \neg\Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{Y}^*) = 0 \tag{6}$$

  then, x_{k-1} behaves as an AND gate, i.e.,

$$\widehat{F}(\mathbf{1}^{k-2}, x_{k-1}, 1, {\psi'}^n_{k+1}(\mathbf{Y}^*), \mathbf{1}^{k-2}, \bar{x}_{k-1}, 0, \neg{\psi'}^n_{k+1}(\mathbf{Y}^*), \mathbf{Y}^*) \leftrightarrow x_{k-1} \wedge \bar{x}_{k-1} \tag{7}$$

  which implies that ζ (with the Skolem functions assigned according to Ψ) will be satisfiable at $k-1$ and hence this terminates the proof.
- Case 2: This case is if

$$\widehat{F}(\mathbf{1}^{k-2}, 1, 1, \Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{1}^{k-2}, 0, 0, \neg\Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{Y}^*) = 1 \tag{8}$$

  In this case, note that $\Psi_{k-1}(\mathbf{Y}*) = 1$ and from Equation (4), we have

$$\exists \mathbf{X}_1^{k-2} \widehat{F}(\mathbf{X}_1^{k-2}, 1, 1, \Psi^n_{k+1}(\mathbf{Y}^*), \neg\mathbf{X}_1^{k-2}, 0, 0, \neg\Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{Y}^*) = 0 \tag{9}$$

  Thus the Skolem function Ψ is wrong at level $k-1$, since it gives 1 but fixing $x_{k-1} = 1$, there is no way to set lower variables to 1. In other words, we have reduced the problem by one level and can recursively apply this argument at level $k-1$.
- Case 3:

$$\widehat{F}(\mathbf{1}^{k-2}, 1, 1, \Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{1}^{k-2}, 0, 0, \neg\Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{Y}^*) = 0 = \Psi_{k-1}(\mathbf{Y}^*) \tag{10}$$

$$\text{and } \widehat{F}(\mathbf{1}^{k-2}, 0, 1, \Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{1}^{k-2}, 1, 0, \neg\Psi^n_{k+1}(\mathbf{Y}^*), \mathbf{Y}^*) = 1 \tag{11}$$

$$\text{i.e., } \widehat{F}(\mathbf{1}^{k-2}, \Psi^n_{k-1}(\mathbf{Y}^*), \mathbf{1}^{k-2}, \neg\Psi^n_{k-1}(\mathbf{Y}^*), \mathbf{Y}^*) = 1 \tag{12}$$

  Note that this case is possible only if $k-2 \geq 1$. But if this is not the case, i.e., if $k-2 = 0$, and $\widehat{F}(\Psi^n_1(\mathbf{Y}^*), \mathbf{1}^{k-2}, \neg\Psi^n_1(\mathbf{Y}^*), \mathbf{Y}^*) = 1$, this implies that there exists no counter-example which contradicts Equation (2). Now we have three subcases:

  - Case 3(a):

$$\widehat{F}(\mathbf{1}^{k-3}, 1, \Psi^n_{k-1}(\mathbf{Y}^*), \mathbf{1}^{k-3}, 0, \neg\Psi^n_{k-1}(\mathbf{Y}^*), \mathbf{Y}^*) = 0$$
$$\widehat{F}(\mathbf{1}^{k-3}, 0, \Psi^n_{k-1}(\mathbf{Y}^*), \mathbf{1}^{k-3}, 1, \neg\Psi^n_{k-1}(\mathbf{Y}^*), \mathbf{Y}^*) = 0$$

    But this case reduces to Case 1 above, i.e., we can see that x_{k-2} behaves as an AND gate (i.e., it is not $\wedge_{k-1}$-unrealizable), and so it terminates.
  - Case 3(b):

$$\widehat{F}(\mathbf{1}^{k-3}, 1, \Psi^n_{k-1}(\mathbf{Y}^*), \mathbf{1}^{k-3}, 0, \neg\Psi^n_{k-1}(\mathbf{Y}^*), \mathbf{Y}^*) = 1$$
$$\text{and from (4), we have,}$$
$$\exists_1^{k-3} \widehat{F}(\mathbf{X}_1^{k-3}, 1\Psi^n_{k-1}(\mathbf{Y}^*), \neg\mathbf{X}_1^{k-3}, 0, \neg\Psi^n_{k-1}(\mathbf{Y}^*), \mathbf{Y}^*) = 0$$

    which as in Case 2, reduces the problem by two levels.
  - Case 3(c):

$$\widehat{F}(\mathbf{1}^{k-3}, 1, \Psi^n_{k-1}(\mathbf{Y}^*), \mathbf{1}^{k-3}, 0, \neg\Psi^n_{k-1}(\mathbf{Y}^*), \mathbf{Y}^*) = 0$$
$$\widehat{F}(\mathbf{1}^{k-3}, 0, \Psi^n_{k-1}(\mathbf{Y}^*), \mathbf{1}^{k-3}, 1, \neg\Psi^n_{k-1}(\mathbf{Y}^*), \mathbf{Y}^*) = 1$$

    But reduces to Case 3 at level $k-3$, thus ensuring strict progress in this case as well.

  Together this completes the proof.

□

C. Proofs from Section IV

Lemma 4. *If $\widetilde{\mathsf{F}} \preceq_{syn} F$, every Skolem function vector for $\mathbf{X}$ in $\widetilde{\mathsf{F}}$ is also a Skolem function vector for $\mathbf{X}$ in F.*

Proof. Let $\mathbf{G}(\mathbf{Y})$ be a Skolem function vector for $\mathbf{X}$ in $\widetilde{\mathsf{F}}(\mathbf{X},\mathbf{Y})$. From condition (a) of Definition 4, we know that $\forall\mathbf{Y}\left(\exists\mathbf{X}F(\mathbf{X},\mathbf{Y}) \Rightarrow \widetilde{\mathsf{F}}(\mathbf{G}(\mathbf{Y}),\mathbf{Y})\right)$. Further, from condition (b) of Definition 4 and using $\mathbf{G}(\mathbf{Y})$ for $\mathbf{X}'$, we have $\forall\mathbf{Y}\left(\exists\mathbf{X}F(\mathbf{X},\mathbf{Y}) \Rightarrow F(\mathbf{G}(\mathbf{Y}),\mathbf{Y})\right)$. This shows that $\mathbf{G}(\mathbf{Y})$ is a Skolem function vector for $\mathbf{X}$ in F. □

Lemma 6. *Let $(\mathbf{T}, \mathsf{Fun}_{\mathbf{T}})$ be an acyclic system of f-defs in F.*

1) If $\mathbf{X} = \mathbf{T}$, then $\mathsf{Fun}_{\mathbf{T}} \preceq_{syn} F$.

2) If $\mathbf{X} \setminus \mathbf{T} \neq \emptyset$, then for every $x_i \in \mathbf{X} \setminus \mathbf{T}$, we have:
If $\theta_{F,\mathbf{T},x_i,0}$ is a tautology, then $(x_i \wedge F|_{x_i=1}) \preceq_{syn} F$. Similarly, if $\theta_{F,\mathbf{T},x_i,1}$ is a tautology, then $(\neg x_i \wedge F|_{x_i=0}) \preceq_{syn} F$.

Proof. To prove part (1), notice that $F \Rightarrow \mathsf{Fun}_{\mathbf{T}}$. Hence, whenever $F(\mathbf{X},\mathbf{Y})$ is satisfied, each of the functional definitions in $\mathsf{Fun}_{\mathbf{T}}$ are also satisfied. Therefore, condition (a) of Definition 4 is satisfied. For condition (b) of Definition 4, notice that for every value of $\mathbf{Y}$, only when the value of $\mathbf{X}'$ is as given by $\mathsf{Fun}_{\mathbf{T}}(\mathbf{X}',\mathbf{Y})$, does $\widetilde{\mathsf{F}}(\mathbf{X}',\mathbf{Y})$ evaluate to 1. For these values of $\mathbf{X}'$, if $\mathbf{Y}$ is such that $\exists\mathbf{X}F(\mathbf{X},\mathbf{Y})$ holds, then $F(\mathbf{X}',\mathbf{Y})$ must also hold since $\mathbf{X} = \mathbf{T}$ and $F \Rightarrow \mathsf{Fun}_{\mathbf{T}}$.

To prove part (2), consider $\theta_{F,\mathbf{T},x_i,0}$ to be a tautology; the proof for the case of $\theta_{F,\mathbf{T},x_i,1}$ being a tautology is analogous. We show below that (a) $\forall\mathbf{Y}\left(\exists\mathbf{X}F(\mathbf{X},\mathbf{Y}) \Rightarrow \exists\mathbf{X}'(x_i' \wedge F(\mathbf{X}',\mathbf{Y})|_{x_i'=1})\right)$, and (b) $\forall\mathbf{Y}\forall\mathbf{X}\left((x_i \wedge F(\mathbf{X},\mathbf{Y})|_{x_i=1}) \Rightarrow F(\mathbf{X},\mathbf{Y})\right)$. Let σ be an arbitrary element in $2^{|\mathbf{Y}|}$. To see why (a) holds, suppose $F(\mathbf{X},\sigma) = 1$. If $x_i = 1$, we set $\mathbf{X}' = \mathbf{X}$ and it follows that $(x_i \wedge F(\mathbf{X}',\sigma)|_{x_i=1}) = 1$. If $x_i = 0$, we set $x_j' = x_j$ for every $x_j \in \mathbf{X} \setminus (\mathbf{T} \cup \{x_i\})$, set $x_i' = 1$ and set the value of every x_j' for $x_j \in \mathbf{T}$ according its functional definition in $\mathsf{Fun}_{\mathbf{T}}(\mathbf{X}',\mathbf{Y})$. Since $\theta_{F,\mathbf{T},x_i,0}$ is a tautology, it follows that $(x_i' \wedge F(\mathbf{X}',\sigma)|_{x_i'=1}) = 1$. To see why (b) holds, suppose $(x_i \wedge F(\mathbf{X},\mathbf{Y})|_{x_i=1}) = 1$. It follows trivially that x_i must be set to 1, and $F(\mathbf{X},\mathbf{Y}) = 1$. □

Lemma 7. *Let $(\mathbf{T}, \mathsf{Fun}_{\mathbf{T}})$ and $(\mathbf{T}', \mathsf{Fun}_{\mathbf{T}'})$ be acyclic systems of f-defs in F, where $\mathbf{T}' \subseteq \mathbf{T} \subseteq \mathbf{X}$ and $\mathsf{Fun}_{\mathbf{T}} \equiv \mathsf{Fun}_{\mathbf{T}'} \wedge \mathsf{Fun}_{\mathbf{T}\setminus\mathbf{T}'}$. For $a \in \{0,1\}$, if $\theta_{F,\mathbf{T}',x_i,a}$ is a tautology, then so is $\theta_{F,\mathbf{T},x_i,a}$.*

Proof. Observe that for any system of acyclic f-defs $(\mathbf{T}, \mathsf{Fun}_{\mathbf{T}})$ in F, since $F(\mathbf{X},\mathbf{Y}) \Rightarrow \mathsf{Fun}_{\mathbf{T}}$, the formula $\theta_{F,\mathbf{T},x_i,a}$ is a tautology iff $F(\mathbf{X},\mathbf{Y})|_{x_i=a} \Rightarrow \exists\mathbf{T}\,F(\mathbf{X},\mathbf{Y})|_{x_i=1-a}$ is a tautology. It is now easy to see that if $\mathbf{T}' \subseteq \mathbf{T} \subseteq \mathbf{X}$ and $\theta_{F,\mathbf{T}',x_i,a}$ is valid, then $\theta_{F,\mathbf{T},x_i,a}$ is valid as well. □

Theorem 8. *For every relational specification $F(\mathbf{X},\mathbf{Y})$, there exists a polynomial-sized Skolem function vector for $\mathbf{X}$ in F iff there exists a* SynNNF *specification $\widetilde{\mathsf{F}}(\mathbf{X},\mathbf{Y})$ such that $\widetilde{\mathsf{F}} \preceq_{syn} F$ and $\widetilde{\mathsf{F}}$ is polynomial-sized in F.*

Proof of Theorem 8. The reverse direction is proved by first applying Theorem 1(ii) to $\widetilde{\mathsf{F}}$, and then noting that since $\widetilde{\mathsf{F}} \preceq_{syn} F$, every Skolem function vector for $\mathbf{X}$ in $\widetilde{\mathsf{F}}$ is also a Skolem function vector for $\mathbf{X}$ in F. For the forward direction, let $\mathbf{\Psi}(\mathbf{Y})$ be a Skolem function vector for $\mathbf{X}$ in F such that the size of an AND/OR/NOT gate circuit representation of $\mathbf{\Psi}$ (denoted $|\mathbf{\Psi}|$) is polynomial in $|F|$. As mentioned in Section II, every such circuit can be converted to NNF in time $\mathcal{O}(|\mathbf{\Psi}|)$. Hence the NNF representation of $\mathbf{\Psi}$ is of size at most polynomial in $|F|$. Therefore, w.l.o.g we consider $\mathbf{\Psi}$ to be in NNF. Now consider the specification $\widetilde{\mathsf{F}}(\mathbf{X},\mathbf{Y}) \equiv \bigwedge_{i=1}^{n}\left((x_i \wedge \psi_i(\mathbf{Y})) \vee (\neg x_i \vee \neg\psi_i(\mathbf{Y}))\right)$. Since no paths from x_i and $\neg x_i$ ($x_i \in \mathbf{X}$) meet at an $\wedge$-labeled node in the circuit representation of $\widetilde{\mathsf{F}}$, it follows that $\widetilde{\mathsf{F}}(\mathbf{X},\mathbf{Y})$ is in SynNNF. Furthermore, by construction of $\widetilde{\mathsf{F}}$, every Skolem function vector for $\mathbf{X}$ in $\widetilde{\mathsf{F}}$ is necessarily component-wise semantically equivalent to $\mathbf{\Psi}$, which is itself a Skolem function vector for $\mathbf{X}$ in F. Therefore, conditions (a) and (b) in Definition 4 are satisfied by $\widetilde{\mathsf{F}}$, and hence $\widetilde{\mathsf{F}} \preceq_{syn} F$. □

D. Proof from Section V

Theorem 9. *For every set $\mathcal{S}$ of clauses,* $\mathsf{C2Syn}(\mathcal{S},\emptyset,1,0)$ *always terminates and returns a Boolean circuit implementing a* SynNNF *specification $\widetilde{\mathsf{F}}$ such that $\widetilde{\mathsf{F}} \preceq_{syn} \varphi_{\mathcal{S}}$.*

Proof. To see that C2Syn always terminates, notice that every time the recursion level ℓ in Algorithm 2 increases, the set of output variables in the remaining set of clauses reduces by 1. Hence, the maximum value of ℓ can only be $|\mathbf{X}|$, and the recursion always terminates. To see why FDRefine (Algorithm 1) terminates, notice that every time $\mathbf{T}'$ changes, its size increases by at least 1, and $\mathbf{T}$ can at most be $\mathbf{X}$. Similarly, every time $\mathcal{S}'$ changes, at least one variable is added to $\mathbf{T}'$, and hence $\mathcal{S}'$ cannot change more than $|\mathbf{X}|$ times.

To see that the returned specification refines $\varphi_{\mathcal{S}}$, notice that each of the **return** statements (lines 3, 6, 8, 12, 17 and 30) uses one of the properties of refinement already discussed in Section IV. The correctness of line 3 is trivial. The correctness of lines 6 and 8 use Propositions 5(2) and 5(2). The correctness of lines 12 and 17 use Lemma 6(1). The correctness of line 30 uses Proposition 5(5). □