Characterizing Boolean Functional Synthesis via Knowledge Representation

Preey Shah, Aman Bansal, S Akshay, Supratik Chakraborty

Abstract

Boolean functional synthesis concerns synthesizing outputs as Boolean functions of inputs such that a relational specification between inputs and outputs is satisfied. This has several applications, including design of safe controllers for autonomous systems, certified QBF solving, cryptanalysis etc. Despite complexity-theoretic hardness results, several algorithms proposed in the literature are known to work well in practice. This motivates the investigation of whether there exist representations of input specifications that permit and also characterize efficient synthesis.

In this paper, we present a normal form called SAUNF that precisely characterizes tractable synthesis in the following sense: a specification is polynomial time synthesizable if it can be compiled to SAUNF in polynomial time. Additionally, a specification admits a polynomial-sized functional solution iff there exists a semantically equivalent polynomial-sized SAUNF representation. SAUNF is exponentially more succinct than well-established normal forms like BDDs and DNNFs, used in the context of AI problems, and strictly subsumes other more recently proposed forms. It enjoys compositional properties that are similar to those of DNNF. Thus, SAUNF provides the right trade-off in knowledge representation for Boolean functional synthesis.

1. Introduction

Machine learning (ML) is increasingly being used for decision-making in safety-critical autonomous systems. As an example, self-driving cars use ML components to “understand” road traffic conditions. This “understanding” then feeds into decision-making for speeding up, slowing down or changing lanes. Clearly, a wrong decision can lead to catastrophic consequences. Ideally, only ML systems with formally proven safety guarantees should be used in such settings. Unfortunately, the complexity of ML systems that are amenable to automated formal verification significantly lags behind the complexity of ML systems that designers would like to deploy in autonomous systems. A pragmatic workaround is to use a monitor that checks in real-time if a safety-impacting decision, say \( D \), produced by an ML system satisfies a formal safety specification, say \( \varphi_{safe}(D, E) \), where \( E \) represents key environment parameters (viz. speed, distance to neighbouring vehicle, etc). If \( \varphi_{safe}(D, E) \) holds, actions (viz. speeding up) based on \( D \) can be safely taken. Otherwise, we must replace \( D \) by a variant \( D' \) that satisfies \( \varphi_{safe}(D', E) \), while remaining “close enough” to \( D \). Since \( D' \) must be generated quickly in systems with fast dynamics, we require an efficient circuit/program that takes \( D \) and \( E \) as inputs and outputs \( D' \) such that \( \varphi_{safe}(D', E) \land (dist(D, D') \leq \varepsilon) \) holds for a suitably defined distance metric \( dist \) for decisions, and a threshold \( \varepsilon \). Note that it is often much simpler to specify the relation between \( D, D' \) and \( E \) (as given above) than to express \( D' \) as an explicit function of \( D \) and \( E \). If \( D, D' \) and \( E \) are represented as Boolean vectors, the above problem translates to Boolean functional synthesis – a topic that has seen a revival of interest in recent years.

Interestingly, the above problem is also useful for “inverting” binarized neural networks, wherein we know the precise relation between outputs \( O \) and inputs \( I \) of a neural network, and are interested in asking what inputs \( I \) would cause the network to output \( O \). This is useful in understanding, for example, what a trained neural network “thinks” is a “bird” or “car”.

Indeed, the core computational problem in such settings is to synthesize Boolean outputs (viz., \( D' \)) as circuits (or loop-free programs) of Boolean inputs (viz. \( D, E \)) such that a given relational specification between inputs and outputs is satisfied. Several algorithmic approaches tackle this problem with varying levels of success (see related work). In this paper however, we are interested in viewing the problem through the lens of Knowledge Representation. That is, what would be the normal form representation of the input specification which could characterize polynomial time (and size) solutions? Our primary contributions are:

- We present SAUNF, a new knowledge representation form that characterizes poly-time Boolean function synthesis, and analyze various operations on it.
- We show that SAUNF is (often exponentially) more succinct and subsumes several other sub-classes (viz. dDNNF, ROBDD, SynNNF). We also present SAUNF...
representations of variants of the well-known factorization problem exhibiting its further utility.

- We present a novel algorithm for compiling a Boolean relational specification in CNF to SAUNF.

Related Work: Recent work on Boolean functional synthesis has mostly been towards design of efficient algorithms (Rabe and Tenstrup 2015; Marij Heule and Biere 2014; Rabe and Seshia 2016; Jiang and Balabanov 2011; Fried, Tabajara, and Vardi 2016; Akshay et al. 2017; Chakraborty et al. 2018; John et al. 2015; Kuncak et al. 2010; Akshay et al. 2018; Golia, Roy, and Meel 2020), which synthesizes outputs as functions of inputs. All these algorithms, using techniques ranging from CEGAR to decision tree learning, empirically work well on some large collections of benchmarks. However, unless some complexity theoretic conjectures are falsified, Boolean functional synthesis requires super-polynomial space and time (Akshay et al. 2018), prompting the search for efficient strategies, e.g., wDNF and ROBDDs give quadratic-time algorithms for synthesis but do not characterize it.

Knowledge representations of Boolean functions have been investigated extensively over the last decades (Cadoli and Donini 1997; Darwiche 2001a; Muise et al. 2016; Darwiche and Marquis 2011) resulting in normal forms with many interesting properties. For Boolean Functional Synthesis, the related question of existential quantification (or forgetting) (Lang, Liberatore, and Marquis 2011; Fried, Tabajara, and Vardi 2016) has been considered, but they work for all or clauses in which variables are quantified and hence are (exponentially) more restrictive than necessary. A recent work (Akshay et al. 2019) provides a normal form called SynNNF which only synthesizes a very specific type of (Skolem) function and hence does not provide a characterization of efficient synthesis itself. As we shall see, our normal form strictly subsumes SynNNF.

2. Preliminaries

Let V be a finite set of Boolean variables. A literal ℓ over V is either v or ¬v, where v ∈ V. Given a Boolean function F : V → {0, 1}, a classical (and often compact) representation of F is as a Boolean circuit with ∧ and ∨ gates, where the circuit inputs are literals over V. Such a circuit can be visualized as a rooted directed-acyclic graph in which nodes with incoming edges, also called internal nodes, are labeled by ∨ and ∧ operators, and nodes with no incoming edges, also called leaves, are labeled by literals over X. A circuit with the above properties is also said to be represent a function in Negation Normal Form (NNF) – such circuits will be the only ones of interest to us, unless stated otherwise. Figure 1 shows an example of such a circuit. As is the case in this figure, multiple leaves of a circuit may have the same literal label. A circuit is said to represent a conjunctive normal form (CNF) formula if it is of the form \( \bigwedge_{i=1}^{k} C_i \), where each C_i is a disjunction of literals.

Suppose the set of variables V is partitioned into two parts: X = {x_1, ..., x_m} representing "outputs", and I = {i_1, ..., i_n} representing "inputs". A circuit C with leaves labeled by literals over X ∪ I represents a (Boolean function, and hence) relational specification over the inputs and outputs. For notational convenience, we use C(X, I) to denote both a circuit with leaves labeled by literals over X ∪ I, and the Boolean function over X ∪ I represented by such a circuit, depending on the context. Specifically, the terms Boolean function and Boolean circuit (or simply circuit) representing a Boolean function, are used interchangeably throughout the paper, although it will be clear from the context which of the two we mean. Given a relational specification C(X, I), the Boolean Functional Synthesis problem asks us to find a set of Boolean functions \( \Psi(I) = \{\psi_i(I) \mid 1 \leq i \leq |I|\} \) such that \( \forall X (C(\Psi(I), I) \iff \exists X C(X, I)) \). As we have seen above, this is an important problem with diverse applications. The function \( \psi_i(I) \) is also called a Skolem function for \( x_i \) in C(X, I), and \( \Psi(I) \) is called a Skolem function vector for X. For example, if \( X = \{a, b\} \) and \( I = \{i\} \) in the circuit of Figure 1, then one (of possibly many) Skolem function vectors is \( \Psi(i) = (\psi_a(i) = \lnot i, \psi_b(i) = \lnot i) \).

A set L of leaves of a circuit C(X, I) is said to be literal-consistent if every leaf in L is labeled by the same literal. For a literal-consistent set of leaves L and for \( b \in \{0, 1\} \), we use C|_{L=b} to denote the circuit obtained by re-labeling each leaf in L with b. For a literal ℓ over X ∪ I, we use the term ℓ-leaves of C to denote the set of all leaves of C labeled ℓ. For a set of distinct literals \( \{\ell_1, ..., \ell_t\} \) and (possibly same) labels \( b_1, ..., b_t \), we abuse notation and use C|_{\ell_1=b_1, ..., \ell_t=b_t} to denote the circuit obtained by re-labeling all \( \ell_j \)-leaves of C by \( b_j \), for all \( j \in \{1, ..., t\} \). Note that since ℓ and ℓ are different literals, the notation C|_{\ell=b, \ell=e=b} is meaningful (and useful), and represents the circuit obtained by re-labeling all ℓ-leaves and ¬ℓ-leaves of C by b.

3. The Subset And-Unrealizable Normal Form (SAUNF)

We start with a basic definition of interest.

**Definition 1.** Given a circuit C(X, I) and a literal ℓ over X∪I, let \( \nu_i \) be the underlying variable of ℓ. We say that ℓ is ∧-realizable in C iff there exists an assignment \( \sigma : X \cup I \setminus \{\nu_i\} \to \{0, 1\} \) such that the Boolean function represented by C|_{w=e, ¬w=e} is semantically equivalent to \( w \land w' \) under the assignment \( \sigma \), where \( w, w' \) are fresh variables not in \( X \cup I \). Further, we say that ℓ is ∧-unrealizable in C if it is not ∧-realizable.

Clearly, if ℓ is ∧-realizable (resp. ∧-unrealizable) in C, then so is ¬ℓ. As an example, consider the circuit C(a,b,i) in Figure 1, and let C_{left} and C_{right} denote the sub-circuits rooted at the left and right child, respectively of the root node. Then b is ∧-realizable in C_{right} and also in C, but is ∧-unrealizable in C_{left}.
We now extend the notion of \(\wedge\)-unrealizability to that of sets of literal-consistent leaves.

**Definition 2.** Let \(S\) be the set of all \(\ell\)-leaves of \(C\). A subset \(S'\) of \(S\) is said to be \(\wedge\)-realizable (resp. \(\wedge\)-unrealizable) in \(C\) if \(\ell\) is \(\wedge\)-realizable (resp. \(\wedge\)-unrealizable) in \(C\) \(\setminus S\cup S'\).

As an example, consider the set \(S'\) of leaves labeled \(a\) (in black) in Figure 1. This is a singleton set containing the 4th leaf from the left. The entire set \(S\) of \(a\)-leaves, however, consists of the 4th and 11th leaves from the left. In order to check \(\wedge\)-(un)realizability of \(S'\), we relabel the 11th leaf (labeled \(a\)) with 0, the 4th leaf (labeled \(a\) in black) with a new variable \(w\), and all leaves labeled \(\neg a\) with \(w'\). It can now be verified that no assignment of \(a\) and \(i\) renders the function represented by the resulting circuit semantically equivalent to \(w\wedge w'\). Hence \(S'\) is \(\wedge\)-unrealizable in the circuit of Figure 1. We now use the above definitions to introduce a new normal form for Boolean circuits that precisely characterizes efficient Boolean functional synthesis.

**Definition 3.** A circuit \(C(X,I)\) is said to be in the Subset And-Unrealizable Normal Form (SAUNF, for short) w.r.t. \(X\) and a sequence \(S = (S_1,S_2,...,S_k)\) of subsets of leaves if the following hold:

1. For each \(i \in \{1,...,k\}\), all leaves in \(S_i\) are labeled by the same literal over \(X\).
2. \(S_1\) is \(\wedge\)-unrealizable in \(C\).
3. For each \(i \in \{2,...,k\}\), \(S_i\) is \(\wedge\)-unrealizable in \(C\) \(\setminus S_1\cup S_2\cup ... \cup S_{i-1}\).
4. The function represented by \(C\) \(\mid S_1;1,S_2;1,...,S_{i};1\) is semantically independent of \(X\).

Consider the circuit \(C(a,b,i)\) in Figure 1 with inputs \(I = \{i\}\) and outputs \(X = \{a,b\}\). Let the leaves of the circuit be denoted by \(L_0,\ldots,L_{15}\), where \(L_i\) denotes the \(i+1\)th leaf from the left in Figure 1. Now, consider the sequence \(S\) of subsets of leaves given by \(S = \{(L_3),(L_7),(L_5),(L_1)\}\). This corresponds to subsets of leaves with the (coloured) labels \(a\), \(b\), \(\neg b\), and \(\neg a\). As seen above, \(\{L_3\}\) is \(\wedge\)-unrealizable in \(C\). It can be verified that \(\{L_7\}\) is \(\wedge\)-unrealizable in \(C\) \(\mid \{L_3\};1\), \(\{L_5\}\) is \(\wedge\)-unrealizable in \(C\) \(\mid \{L_3\};1,\{L_7\};1\) and \(\{L_1\}\) is \(\wedge\)-unrealizable in \(C\) \(\mid \{L_3\};1,\{L_7\};1,\{L_5\};1\). Finally, the function represented by \(C\) \(\mid \{L_3\};1,\{L_7\};1,\{L_5\};1\) is semantically equivalent to 1, and hence independent of \(\{a,b\}\). Hence, the circuit \(C\) is in SAUNF w.r.t \(X = \{a,b\}\) and the sequence \((\{L_3\},\{L_7\},\{L_5\},\{L_1\})\).

**Relation with other normal forms**

A normal form for Boolean functional synthesis, called SynNNF, was recently proposed in (Akshay et al. 2019), where a conditional result was shown, namely, SynNNF is super-polynomially more succinct than DNNF (Darwiche 2001a) and dDNNF (Darwiche 2001b), unless some long-standing complexity theoretic conjectures are falsified. We use a result from (Bova et al. 2016) to show the following stronger result.

**Lemma 1.** SAUNF is unconditionally exponentially more succinct than DNNF and dDNNF.

For lack of space, we omit the proof of Lemma 1 here. All omitted proofs are presented in the supplementary material. Next, we show that SynNNF is in fact a very special case of SAUNF, by showing that every SynNNF circuit is a SAUNF circuit, but not vice versa.

**Lemma 2.** Every SynNNF circuit \(C(X,I)\) is also an SAUNF circuit with a sequence of \(|2|X|\wedge\)-unrealizable subsets of leaves. However, there exist SAUNF circuits that are not in SynNNF.

We remark that the circuit in Figure 1, already shown to be in SAUNF, is not in SynNNF. This is because the definition of SynNNF (Akshay et al. 2019) entails that for a circuit \(C(X,I)\) to be in SynNNF, at least one output in \(X\) must be \(\wedge\)-unrealizable in \(C\). However, neither \(a\) nor \(b\) is \(\wedge\)-unrealizable in the circuit. Specifically, \(C\) \(\mid a=0,\neg a=0\) \(\Leftrightarrow w\wedge w'\) when \(i = 0\) and \(b = 0\), and \(C\) \(\mid b=0,\neg b=0\) \(\Leftrightarrow w\wedge w'\) when \(i = 1\) and \(a = 1\).

**4. SAUNF and Skolem functions**

We now discuss how Boolean functional synthesis can be efficiently solved if the relational specification \(C(X,I)\) is in SAUNF. Informally, the process involves
transforming $C$ to a semantically different but related specification $D(X, X', I)$, where $X'$ is a set of fresh output variables, such that a Skolem function vector for $X \cup X'$ in $D$ can be found efficiently, and a projection of this Skolem function vector on $X$ directly yields a Skolem function vector for $X$ in $C$. To formalize this notion, we begin with a few definitions.

**Definition 4. Equisynthesizable Under Projection.** Let $C(X, I)$ and $D(X, X', I)$ be circuits representing relational specifications on inputs $I$ and outputs $X$ and $X \cup X'$, respectively. We say that $C$ is equisynthesizable to $D$ under projection, as represented by $C \rightsquigarrow D$, iff the following hold:

1. $\forall \forall X \ (C(X, I) \implies \exists X' D(X, X', I))$
2. $\forall \forall \forall X' (D(X, X', I) \implies C(X, I))$

It follows from the above definition that $\rightsquigarrow$ is transitive. The following theorem is easily seen to be true.

**Theorem 1.** If $C(X, I) \rightsquigarrow D(X, X', I)$ and $|D|$ grows polynomially with $|C|$, then the following hold:

1. If Boolean function synthesis for $D(X, X', I)$ can be solved in time polynomial in $|D|$, the same problem for $C(X, Y)$ can be solved in time polynomial in $|C|$.
2. If there exists a Skolem function vector for $X \cup X'$ in $D$ of size polynomial in $|D|$, there exists a Skolem function vector for $X$ in $C$ of size polynomial in $|C|$.

**Role of auxiliary output variables**

Definition 4 and Theorem 1 hint at the important role played by auxiliary output variables $X'$ that are present in $D$, but not in $C$. We now investigate how these auxiliary variables can be introduced, and how they help in generating Skolem functions.

Let $C_1$ and $C_2$ be two sub-circuits of $C(X, I)$ such that $C_2$ is not a sub-circuit of $C_2$ and vice versa. For a fresh auxiliary variable $p \not\subseteq X \cup I$ and for $i \in \{1, 2\}$, define a circuit transformation $\tau^p_i$ that replaces the sub-circuit $C_i$ in $C$ with $C_i \land p$.

**Lemma 3.** If the Boolean function represented by $C_1 \land C_2$ is unsatisfiable, then $C \rightsquigarrow \tau^p_1(\tau^p_2(C))$.

**Proof.** Let $D(X, p, I)$ denote the circuit $\tau^p_1(\tau^p_2(C))$, and let $D_1$ and $D_2$ denote the sub-circuits $p \land C_1$ and $\neg p \land C_2$ respectively in $D$. We show below that the conditions for $C \rightsquigarrow D$ (see Definition 4) are satisfied.

Let $\sigma : X \cup I \rightarrow \{0, 1\}$ be an assignment for which $C(X, I)$ evaluates to 1. We have four cases to analyze depending on what $C_1$ and $C_2$ evaluate to under $\sigma$.

- $C_1 = 0 = C_2$, then for any value of $p$, $D_1$ and $D_2$ also evaluate to 0, and hence $D$ evaluates to the same value, viz. 1, as $C$.
- If $C_1 = 1, C_2 = 0$, then with $p = 1$, $D_1$ evaluates to 1 and $D_2$ evaluates to 0, and hence $D$ evaluates to the same value, viz. 1, as $C$.
- By a similar argument, if $C_1 = 0, C_2 = 1$, using $p = 0$ causes $D$ to evaluate to 1.
- The case of $C_1 = C_2 = 1$ doesn’t arise since $C_1 \lor C_2$ always evaluates to 0.

This shows that $\forall \forall X \ (C(X, I) \implies D(X, p, I))$.

Consider any assignment $\sigma' : X \cup \{p\} \cup I \rightarrow \{0, 1\}$ that causes $D$ to evaluate to 1. Let $\sigma : X \cup I \rightarrow \{0, 1\}$ be the projection of $\sigma'$ on $X \cup I$. By definition, the function represented by $D_{|p=0}$ is semantically equivalent to $C$. Since all internal gates in $D$, viz. $\land$ and $\lor$ gates, are monotone, $\sigma$ must cause $D_{|p=1}$ to evaluate to 1 as well. Hence, $\sigma$ causes $C(X, I)$ to evaluate to 1. This shows that $\forall \forall X \forall p \ (D(X, p, I) \implies C(X, I))$.

The argument in the above proof can be easily generalized to prove the following.

**Lemma 4.** Let $C_1 = \{C_{1,1}, \ldots, C_{1,s}\}$ and $C_2 = \{C_{2,1}, \ldots, C_{2,t}\}$ be two sets of sub-circuits of $C(X, I)$ such that (a) there are no distinct $C_{1,i}$ and $C_{2,j}$ where one is a sub-circuit of the other, and (b) $\lor_{i=1}^{s} C_{1,i} \implies \lor_{j=1}^{t} C_{2,j}$, where we have used $C_{k,i}$ to denote the Boolean function represented by the corresponding circuit. Let $\tau^p_{C_k}$ denote the circuit transformation that replaces every sub-circuit $C_{k,i}$ with $p \land C_{k,i}$. Then $C \rightsquigarrow \tau^p_{C_1}(\tau^p_{C_2}(C))$.

A particularly easy application of Lemma 4 is obtained by choosing any literal $\ell$ that labels leaves of $C$, and by choosing $C_1$ and $C_2$ to be subsets of $\ell$-leaves and $\neg \ell$-leaves, respectively. Note that if $L$ is a subset of $\ell$-leaves of $C$, then $\tau^p_{L}(C)$ gives the same circuit as $C |_{L \lor \ell}$.

Given a relational specification $C(X, I)$ and a literal $\ell$ over $X$, let $v_\ell$ be the underlying variable in $X$, and $S_\ell$ be the set of all $\ell$-leaves of $C(X, I)$.

**Theorem 2.** Suppose $S \subseteq S_\ell$ is $\land$-unrealizable in $C(X, I)$. For a fresh auxiliary output variable $p \not\subseteq X \cup I$, let $E(X, p, I) = C |_{S_\ell \lor \{p\} \land N \land S_{\neg \ell} \land p = 0}$ and $D(X \setminus \{v_\ell\}, p, I) = E |_{\ell = 1, \neg \ell = 1}$. Then the following statements hold.

1. $\exists v_\ell C \equiv \exists v_\ell D \equiv \exists v_\ell C |_{S:1}$
2. If $\Psi_D(I)$ is a Skolem function vector for $X \setminus \{v_\ell\} \cup \{p\}$ in $D$, then the projection of $\Psi_D$ on $X \setminus \{v_\ell\}$ augmented with the function $E |_{\ell = 1, \neg \ell = 0}$ gives a Skolem function vector $\Psi_C(I)$ for $X$ in $C$.

**Proof.** By Lemma 4, we have $C \rightsquigarrow E$. It then follows from Definition 4 that $\exists v_\ell C \equiv \exists v_\ell D \equiv \exists v_\ell E$. Furthermore, since $S$ is $\land$-unrealizable in $C$, it follows from the definition of $E$ and Definition 2 that $v_\ell$ is $\land$-unrealizable in $E$. Therefore, by a result of (Akshay et al. 2018), we have $\exists v_\ell E \equiv \exists v_\ell E |_{\ell = 1, \neg \ell = 1} \equiv D$. Hence, $\equiv \exists v_\ell D \equiv \exists v_\ell E \equiv \exists v_\ell E |_{\ell = 1, \neg \ell = 0} \equiv \exists v_\ell D$ (as shown in (Akshay et al. 2018)). Since $\exists v_\ell E \equiv D$, the Skolem function vector for $X \setminus \{v_\ell\}$ in $E$ is $\Psi_D(I)$ projected on $X \setminus \{v_\ell\}$. The Skolem function vector for $\ell$ is given by $E |_{\ell = 1, \neg \ell = 0}(\Psi_D(Y), 1, Y)$, as described in (Akshay et al. 2018; Jiang 2009; Fried, Tabajara, and Vardi 2016;...
Algorithm 1: SkGen($C, S, r$)

Input: $C(X, Y)$: Relational spec in SAUNF; $S = (S_1, S_2, \ldots, S_k)$: Sequence of $\land$-unrealizable subsets of $X$-leaves of $C$; $r$: Recursion level

Output: $\Psi_C(Y)$: Skolem function vector for $C$

1. if $r = k + 1$ then
2. $\Psi_C(Y) := \text{GetAnyFuncVec}(\{X, Y\})$;
3. // Returns an $|X|$-dim vector of (arbitrary) functions of $Y$
4. else
5. $\ell := \text{Literal label of leaves in } S_r$;
6. $p_r := \text{newOutputVar}()$;
7. $E(X', \ell, Y) := \text{GetCkt}(C, S_r, \ell, p_r)$;
8. // Replace all $\ell$-labeled leaves of $C$
9. $D := \text{CPropSimp}(E|_{r=1, \neg r=1})$;
10. // $\text{CPropSimp}$ propagates constants and eliminates gates with constant outputs
11. $\psi(Y) := \text{CPropSimp}(E(\Psi_D(Y), \ell, 1, Y))$;
12. $\Psi_C(Y) := (\Psi_D(Y) \land \psi_p(Y), \psi_l(Y))$
13. return $\Psi_C(Y)$;

Trivedi 2003). The second part of the theorem now follows from the observation that $C \rightsquigarrow E$. □

Theorem 2 suggests an efficient algorithm for generating a Skolem function vector from a SAUNF specification. Algorithm 4 presents the pseudo-code of algorithm SkGen. The purpose of sub-routines used in SkGen is explained in comments. If $|C|$ denotes the size (i.e., count of nodes and edges) of the circuit $C$, it is easy to see that SkGen runs in $O(|C|^2)$ time, assuming the vector returned in line 2 can be constructed in time $O(|C|^2 \cdot |C'|)$. Since any arbitrary function vector suffices in line 2, the assumption is easily satisfied by choosing constant functions, for example.

We illustrate the running of SkGen by considering its execution on the specification $C(a, b, i)$ shown in Fig. 1. Here, the outputs are $X = \{a, b\}$ and the input is $I = \{i\}$. As discussed earlier, we use $L_0$ through $L_{15}$ to denote the leaves of the circuit in left-to-right order. We have also seen earlier that $C$ is in SAUNF for the sequence of subsets of leaves $(S_1, S_2, S_3, S_4)$, where $S_1 = \{L_1\}, S_2 = \{L_2\}, S_3 = \{L_1\}$ and $S_4 = \{L_2\}$. As algorithm SkGen proceeds, labels of different leaves of $C$ need to be updated. For notational convenience, we use $C^{(r)}$, $D^{(r)}$ and $E^{(r)}$ to refer to the circuits $C$, $D$ and $E$ in the $r$th level of recursion of SkGen. Table 1 shows how $C^{(r)}$, $D^{(r)}$ and $E^{(r)}$ are obtained by replacing the labels of suitable leaves of $C$.

Table 1: Run of Algorithm 4 on Fig. 1

Each entry in this table lists which leaf labels of $C$ must be updated, where $L_{i,j,k} : f$ denotes updation of the leaf of each label in $\{L_i, L_j, L_k\}$ by $f$. All leaves whose label updates are not specified are assumed to have the same labels as in $C$. It can be verified that $C$ with leaf labels updated as in the table entry corresponding to $D^{(4)}$ simplifies to $1$ by constant propagation. Hence $D^{(4)}$ is semantically independent of output variables. This is not a coincidence, but is guaranteed by the definition of SAUNF. Hence, at recursion level 5 of SkGen, any vector of functions $(f_3(i), f_4(i))$ can be returned in line 2 of Algorithm 4 as a Skolem function vector for $(p_3, p_4)$ in $D^{(2)}$.

As the recursive calls return, we obtain $E^{(4)}(p_3 = f_3(i), p_4 = f_4(i), p_1 = 1, i)$ as Skolem function for $p_1$ in $E^{(4)}$. Call this function $f_1(i)$. Next, we get $f_2(i) = E^{(3)}(p_1 = f_1(i), p_3 = f_3(i), p_2 = 1, i)$ as Skolem function for $p_2$ in $E^{(3)}$. Continuing further, we obtain $f_0(i) = E^{(2)}(p_1 = f_1(i), p_2 = f_2(i), b = 1, i)$ as Skolem function for $b$ in $E^{(2)}$, and $f_0(i) = E^{(1)}(b = f_0(i), p_1 = f_1(i), a = 1, i)$ as Skolem function for $a$ in $E^{(1)}$. The final return gives $(f_0(i), f_0(i))$ as a Skolem function vector for $(a, b)$ in $C$. Note that different choices of $f_3(i), f_4(i)$ yield different Skolem function vectors of $C$, all of which are correct.

Theorem 3. Given a specification $C(X, I)$ in SAUNF, Algorithm SkGen generates a polynomial-sized Skolem function vector in polynomial-time.

Next, we show that if we already know one (out of possibly many) Skolem function vector of a relational specification $C(X, I)$, we can easily derive a semantically equivalent specification in SAUNF.

Lemma 5. Let $\Psi(I)$ be a Skolem function vector for $X$ in $C(X, I)$. Define $G(I)$ to be $C(\Psi(I), I)$, $\widehat{F}(X, I)$ to be $\bigwedge_{1 \leq n} \left( (x_i \land \psi_i) \lor (\neg x_i \land \neg \psi_i) \right)$, and $H(X, I)$ to be $\widehat{F}(X, I) \lor F(X, I) \land G(I)$. Then $H(X, I)$ is in SAUNF.
and both $H(X, I)$ and $C(X, I)$ represent semantically equivalent specifications.

In the above Lemma, $H(X, I)$ is the required specification. The following theorem is an immediate consequence of Lemma 5 and Theorem 3.

**Theorem 4.** Given a relational specification $C(X, I)$,
1. A Skolem function vector can be computed in polynomial time if a semantically equivalent SAUNF form can be computed in polynomial time.
2. A Skolem function vector has polynomial size if there exists a semantically equivalent SAUNF form of the specification that is polynomial sized.

Thus, SAUNF truly characterizes efficient Boolean functional synthesis. Note that Theorem 4 is significantly stronger than sufficient conditions for efficient synthesis given in (Akshay et al. 2018; 2019).

5. Operations on SAUNF

In this section, we discuss the application of basic operations like conjunction, disjunction and existential quantification of variables on SAUNF representations.

**Lemma 6.** Suppose $G(X, I)$ is in SAUNF w.r.t $X$ and a sequence $S^G$, and $H(X, I)$ is in SAUNF w.r.t. $X$ and $S^H$. Then $G \vee H$ obtained by adding a two-input OR gate fed by outputs of $G$ and $H$ is in SAUNF w.r.t $X$ and $(S^G, S^H)$.

**Proof.** The core of the proof is the claim that $(S^G, S^H)$ is a sequence of subsets of leaves of $G \vee H$ that satisfies the conditions of Definition 3. To see why this is so, note that by Definition 2, when considering a subset, say $S^G$, of $\ell$-labeled leaves in $S^G$, all $\ell$-labeled leaves of $H$ must be re-labeled 0. Hence, $H$ can only contribute $\neg \ell$, and can, at worst, combine with $\ell$ contributed by $G$ at the $\neg \ell$-labeled root of $G \vee H$. Since the set of $\ell$-labeled leaves in $S^G$ was already $\land$-unrealizable in $G$, we find that $S^G$ is $\land$-unrealizable in $G \vee H$ as well. By repeating this argument, we find that all sub-sets of leaves in $(S^G, S^H)$ are set to 1, the circuit $G \vee H$ represents a function that is semantically independent of $X$. Hence, $G \vee H$ is in SAUNF.

Significantly, Lemma 6 does not require any assumptions on the relation between ordering of subsets in $S^G$ and $S^H$. Other popular normal forms do not enjoy this property. For example, disjoining two ROBDDs (Reduced Ordered Binary Decision Diagrams) constructed with different ordering of variables does not always yield an ROBDD in polynomial-time. Similarly, combining two SynNNF circuits with an OR gate may not result in a SynNNF circuit unless the ordering of output variables in both circuits are the same. This shows that disjunction is more efficiently computable in SAUNF than in ROBDDs and in SynNNF.

**Lemma 7.** Suppose $C(X, I)$ in SAUNF w.r.t $X$ and a sequence $S$. Let $L$ be the set of all leaves of $C$ that are labeled by a literal over $X$. Then $\exists X C \Leftrightarrow C \mid L:1.$
In Section 1, we had discussed applications of Boolean functional synthesis to synthesis of real-time monitors in safety-critical autonomous systems and to inversion of binarized neural networks. In this section, we discuss some other interesting applications. Specifically, we consider the bounded integer factorization problem—a problem of immense interest in cryptanalysis, and show some interesting partial results using SAUNF circuits. The relational specification $C_{fact}$ for factorization is given by $X \times Y = I$ which outputs 1 if and only if the given product relation holds, where $X, Y$ are $n$-bit outputs vectors and $I$ is a $2n$ bit input vector. A bit at position $j$ refers to the $j^{th}$ least significant bit of the vector. For $1 \leq i \leq j \leq 2n$, define $R[i, j]$ to be the relation that outputs 1 if and only if the bits from position $i$ to $j$ of $X \times Y$ are equal to bits from position $i$ to $j$ of $I$.

From the results presented in this paper, it follows that if $R[1, 2n]\land(X \neq 1)\land(Y \neq 1)$ can be represented as a SAUNF circuit of size $\alpha(n)$, (Skolem function) circuits of size polynomial in $\alpha(n)$ can be obtained for bounded integer factorization with non-trivial (i.e. $\neq 1$) factors. This would have serious ramifications for cryptanalysis. While we are not close to achieving such a result, our initial studies show some interesting results in trying to represent $R[i, j]$ in SAUNF. Note that it is already known (Bryant 1991), that representing $R[n, n]$ requires exponentially large ROBDDs. With SAUNF circuits however, we obtain a significant improvement.

**Theorem 6.** For $i \leq j \leq 2n, j - i < n$, $R[i, j]$ is representable by a polynomial (in $n$) sized SAUNF circuit.

To the best of our knowledge, such a result is not known for other normal forms like DNNF, dDNNF, or even SynNNF. Surprisingly, we can use Theorem 6 to also show that a restricted version of division has a polynomial sized SAUNF representation. Consider $n$-bit numbers $X, Y$ and a $2n$-bit number $I$ such that the relational specification circuit outputs 1 iff $X \times Y = I$ holds. For the division problem, we treat $I, Y$ as inputs to the problem and $X$ as the output and write the relation as $X = I/Y$.

**Theorem 7.** The relation $X = I/Y$, with inputs $I, Y$ restricted to odd numbers (i.e. the relation evaluates to 0 if $I$ or $Y$ is even), is representable as a polynomial (in $n$) sized SAUNF circuit.

### 8. Conclusion

In this paper, we presented a normal form SAUNF for relational specifications that characterizes efficient Boolean functional synthesis. This is a significantly stronger characterization than those used in earlier work. SAUNF is (unconditionally) exponentially more succinct than DNNF, dDNNF while enjoying similar composability properties. It also strictly subsumes the recently proposed SynNNF. As future work, we plan to implement the algorithm to transform a given circuit to SAUNF and apply it on benchmarks from cryptanalysis, ML-based controller synthesis and inverting binarized neural nets.
Broader Impact and Ethics Statement

Identifying and exploiting knowledge representations for various problems (e.g., model counting, neural network analysis etc) has been a classical area of study in AI. For example, knowledge representations for model counting has been considered by researchers for more than two decades and this has led a deeper understanding of this problem and its potential applications.

In this work, we have focussed on the specific problem of Boolean functional synthesis and undertaken a study of knowledge representation characterization for this particular problem. We have also pointed out two/three potential applications of this problem in an AI and cryptanalysis settings. Understanding the structure of the specification in these domains via the lens of knowledge representation allows us better insight into the problems and hopefully to build better algorithms with theoretical and rigid guarantees. This is indeed one of the holy grails of AI and the current paper could be seen as a step in the direction.

We understand however that the techniques in the paper can potentially lead to reverse engineering (binary-coded) neural networks and other systems representable as Boolean Functions. Thus one could obtain functional insight into opaque black-box systems (like cryptosystems). We hope that with sound regulatory mechanisms, potential abuse can be avoided. We also hope that this work at its foundational level would help researchers in these domains understand and hence build techniques to counter and avoid such approaches.

References


Muir, C.; McIlraith, S. A.; Beck, J. C.; and Hsu, E. 2016. DSHARP: Fast d-DNF Compilation with sharp-SAT. In AAAI-16 Workshop on Beyond NP.


Characterizing Boolean Functional Synthesis via Knowledge Representation (Supplementary Material)

Typos: Please consider the following small, but important, fixes of typos in the paper. The characters in red should be replaced by those in blue.

• Line 298: $C(X, Y)$ should be $C(X, I)$
• Line 382: $E \mid_{\ell=1, \neg \ell=0} (\Psi_D(Y), 1, Y)$ should be $E \mid_{\ell=1, \neg \ell=0} (\Psi_D(I), 1, I)$.
• In Algorithm 1, all instances of $Y$ should be replaced by $I$ to be notationally consistent with the previous discussion.
• Line 446: $(\tilde{F}(X, I) \lor F(X, I)) \land G(I)$ should be $(\tilde{F}(X, I) \lor C(X, I)) \land G(I)$.

Skipped proofs of Lemmas and Theorems:

Lemma 1. SAUNF is unconditionally more succinct than DNNF and dDNNF.

Proof. In Proposition 11 of (Bova et al. 2016), a family of Boolean functions $J_{S_{2n}}$ is described that can be represented in CNF in size $O(n^2)$, whereas any DNNF (and hence also DNF) representation of the function requires size $2^\Omega(n^3)$. The function $J_{S_{2n}}$ asserts that for every triple $(v_1, v_2, v_3)$ of variables in a carefully constructed set $A_{2n}$ of triples, at least one of $v_1$, $v_2$ or $v_3$ must be false. Furthermore, it is shown in (Bova et al. 2016) than $|A_{2n}| = O(n^2)$. Clearly, the CNF representation of $J_{S_{2n}}$ has $|A_{2n}|$ clauses, where each clause has three negated variables as literals. Therefore, no literal-consistent subset of leaves can be $\land$-realizable in the circuit corresponding to this CNF representation. This implies that $J_{S_{2n}}$ can be represented in SAUNF in size $O(n^2)$. This establishes that SAUNF is unconditionally exponentially more succinct compared to DNNF and dDNNF.

To prove Lemma 2, we need to recall the definition of SynNNF from (Akshay et al. 2019). Using our notation, this definition can be written as follows.

Definition 5. A circuit $C(X, I)$ is in SynNNF w.r.t. $X$ and a linear ordering $x_1 < x_2 < \ldots < x_n$ of variables in $X$ if the following conditions hold:

• $x_1$ is not $\land$-realizable in $C$.
• For $2 \leq i \leq n$, $x_i$ is not $\land$-realizable in $C \mid_{x_1=1, \neg x_1=1, \ldots, x_{i-1}=1, \neg x_{i-1}=1}$.

We can now prove the following lemma.

Lemma 2. Every SynNNF circuit $C(X, I)$ is also an SAUNF circuit with a sequence of $2|X| - \land$-realizable subsets of leaves. However, there exist SAUNF circuits that are not in SynNNF.

Proof. This is a direct application of the definitions. Suppose a circuit $C(X, I)$ is in SynNNF, and let $|X| = n$. We use the ordering $x_1 < \ldots < x_n$ of variables in Definition 5 to define a sequence of $2n$ literal-consistent subsets of leaves of $C$ as follows. For each $i \in \{1, \ldots, n\}$, we define $S_{2i-1}$ to be the set of all leaves of $C$ labeled $x_i$, and $S_{2i}$ to be the set of all leaves of $C$ labeled $\neg x_i$. It is now easy to see from Definition 5 and Definition 3 that $C$ is in SAUNF w.r.t. $X$ and the sequence $S = (S_1, \ldots, S_{2n})$ of subsets of literal-consistent leaves. This proves the first part of the lemma.

To show the second part, we need to show a circuit that is in SynNNF but not in SAUNF. We have already discussed earlier that the circuit in Figure 1 satisfies this requirement. This completes the proof.

Theorem 1. If $C(X, I) \equiv D(X, X', I)$ and $|D|$ grows polynomially with $|C|$, then the following hold:

1. If Boolean function synthesis for $D(X, X', I)$ can be solved in time polynomial in $|D|$, the same problem for $C(X, I)$ can be solved in time polynomial in $|C|$.
2. If there exists a Skolem function vector for $X \cup X'$ in $D$ of size polynomial in $|D|$, there exists a Skolem function vector for $X$ in $C$ of size polynomial in $|C|$.

Proof. Since $C$ and $D$ are equisatisfiable under projection, for any assignment of $X, X'$ and $I$ that satisfies $D$, the projection of this assignment on $X \cup I$ must satisfy $C$. The Skolem function vector of $C$ is then simply obtained from the Skolem function vector of $D$ by dropping the Skolem functions of variables $X'$ not present in $C$. Therefore, if a Skolem function vector for $|D|$ is computable in time (resp. has size) polynomial in $|D|$, we can obtain a Skolem function vector for $|C|$ in time (resp. of size) polynomial in $|D|$. Since $|D|$ grows polynomially with $|C|$, we have the result.

Lemma 4. Let $C_1 = \{C_{1,1}, \ldots, C_{1,t}\}$ and $C_2 = \{C_{2,1}, \ldots, C_{2,t}\}$ be two sets of sub-circuits of $C(X, I)$ such that (a) there are no distinct $C_{1,i}$ and $C_{1,j}$ where one is a sub-circuit of the other, and (b) $\bigvee_{i=1}^t C_{1,i} \implies \bigwedge_{j=1}^t \neg C_{2,j}$, where we have used $C_{k,i}$ to denote the Boolean function represented by the corresponding circuit. Let $\tau_{C_1}^p$ denote the circuit transformation that replaces every sub-circuit $C_{k,i}$ with $p \land C_{k,i}$. Then $C \equiv \tau_{C_1}^p (\tau_{C_2}^p(C))$.

Proof. Let $D(X, p, I)$ denote the circuit $\tau_{C_1}^p (\tau_{C_2}^p(C))$, and let $D_1 = \{p \land C_{1,1}, \ldots, p \land C_{1,s}\}$ and $D_2 = \{\neg p \land C_{2,1}, \ldots, \neg p \land C_{2,t}\}$. We show below that the conditions for $C \equiv D$ (see Definition 4) are satisfied.

Let $\sigma : X \cup I \to \{0, 1\}$ be an assignment for which $C(X, I)$ evaluates to 1. We have four cases to analyze depending on what $\bigvee_{i=1}^s C_{1,i}$ and $\bigwedge_{j=1}^t \neg C_{2,j}$ evaluate to under $\sigma$.

• $\bigvee_{i=1}^s C_{1,i} = 0$ and $\bigwedge_{j=1}^t \neg C_{2,j} = 0$: Clearly, all $C_{1,i} = 0$. Therefore, with $p = 0$, each sub-circuit in $D_1$ and $D_2$ evaluates to the same value as the corresponding
sub-circuit in \( C_1 \) (each sub-circuit in \( C_1 \) evaluates to 0) and \( C_2 \) respectively. Hence \( D \) evaluates to the same value as \( C \).

- \( \forall i=1 \; C_{1,i} = 1 \) and \( \land_{t=1}^{|C|} \neg C_{2,j} = 1 \): With \( p = 1 \), each sub-circuit in \( D_1 \) and \( D_2 \) evaluates to the same value as the corresponding sub-circuit in \( C_1 \) and \( C_2 \) (each sub-circuit in \( C_2 \) evaluates to 0) respectively. Hence \( D \) evaluates to the same value as \( C \).

- \( \forall i=1 \; C_{1,i} = 0 \) and \( \land_{j=1}^{|C|} \neg C_{2,j} = 1 \): Clearly, all \( C_{1,i} = 0 \) and all \( C_{2,j} = 0 \) as well. Therefore, for any value of \( p \), each sub-circuit in \( D_1 \) and \( D_2 \) evaluates to the same value as the corresponding sub-circuit in \( C_1 \) and \( C_2 \) respectively (all subcircuits in both \( C_1 \) and \( C_2 \) evaluate to 0). Hence \( D \) evaluates to the same value as \( C \).

- The only remaining case is when \( \forall i=1 \; C_{1,i} = 1 \) and \( \land_{j=1}^{|C|} \neg C_{2,j} = 0 \). This cannot happen because of the condition \( \forall i=1 \; C_{1,i} \implies \land_{j=1}^{|C|} \neg C_{2,j} \).

This shows that \( \forall i \forall X \; (C(X,I) \iff \exists p \; D(X,p,I)) \).

Conversely, consider any assignment \( \sigma' : X \cup \{p\} \cup I \to \{0,1\} \) that causes \( D \) to evaluate to 1. Let \( \sigma : X \cup I \to \{0,1\} \) be the projection of \( \sigma' \) on \( X \cup I \).

By definition of circuit \( D \), the function represented by the circuit \( D|_{p=1,\neg p=1} \) is semantically equivalent to the function represented by \( C \). Since all internal gates in \( D \), viz. \( \land \) and \( \lor \) gates, are monotone, and since \( \land \) in which either \( p \) or \( \neg p \) is 0 causes \( D \) to evaluate to 1, by monotonicity, \( \sigma \) must cause \( D|_{p=1,\neg p=1} \) to evaluate to 1 as well. Hence, \( \sigma \) causes \( C(X,I) \) to evaluate to 1. This shows that \( \forall i \forall X \forall p \; (D(X,p,I) \iff C(X,I)) \).

Hence \( C \iff D \) holds.

**Theorem 3.** Given a specification \( C(X,I) \) in \( \text{SAUNF} \), Algorithm \( \text{SkGen} \) generates a polynomial-sized Skolem function vector in polynomial-time.

**Proof.** The fact that Algorithm \( \text{SkGen} \) generates a correct Skolem function vector follows from an inductive application of Theorem 2. Specifically, for each level \( i \) of recursion, if the Skolem function \( \Phi_D \) returned in line 9 by the \( i+1 \)th recursive call of \( \text{SkGen} \) is correct for \( D \), Theorem 2 ensures that the Skolem function \( \Phi_C \) computed in lines 10 and 11 of the \( i \)th recursive call is correct for \( C \).

To see why the terminating case of this recursion yields correct Skolem functions, note that when the recursion level is \( k+1 \) (lines 1-2 of Algorithm 4), by Definition 3, the function represented by \( C \) is semantically independent of \( X \). Hence any Skolem function vector for \( X \) suffices in line 2 of Algorithm 4.

To prove the polynomial bound on running time and hence, on size, of the generated Skolem function vector, we assume that the vector of functions used in line 2 of Algorithm 4 has size in \( O(k^2 \cdot |C|) \). Since an arbitrary vector of functions of \( I \) suffices in line 2, we can choose a \( |X| \)-dimensional constant function vector, say \((0, \ldots, 0)\) as the output of \( \text{GetAnyFuncVec} \). Hence, the above assumption can always be satisfied.

Algorithm \( \text{SkGen} \) has exactly \( k+1 \) recursive calls, and in each of the first \( k \) calls, the steps in lines 4, 5, 6, 7 and 8 take time polynomial in \( |C| \) and generate circuits \( E \) and \( D \) that are of size in \( O(|C|) \). Indeed, the circuit \( D \) in each recursion level \( \leq k \) is simply \( C \) with the literal labels of some of its leaves replaced by other literals or by Boolean constants (possibly followed by simplification by constant propagation). Therefore, \( |D| \leq |C| \) in each recursion level \( \leq k \). The circuit \( E \) is similarly obtained by replacing some leaves of \( C \) with Boolean constants, other literals or conjunctions of two literals. Therefore, \( |E| \leq 2 \times |C| \) in each recursion level \( \leq k \). In the \( k+1 \)th recursive call, line 2 is executed, and as discussed above, we restrict it to take time in \( O(k^2 \cdot |C|) \). This also ensures that the size of the Skolem function vector returned in line 2 is in \( O(k^2 \cdot |C|) \).

Once the recursive calls start returning, lines 10 and 11 of Algorithm \( \text{SkGen} \) are executed. Note that in line 10, the Skolem function for \( \ell \) is obtained by feeding the inputs of circuit \( E \) (as obtained in the current level of recursion) by outputs of Skolem functions computed in later (or higher) levels of the recursion. We have already seen above that \( |E| \) is at most \( 2 \times |C| \) in each recursion level \( \leq k \). A Skolem function computed at recursion level \( j \) can potentially feed into \( E \) at all recursion levels \( \{1, \ldots, j-1\} \). Therefore, a maximum of \( \sum_{j=1}^{k-1} (j-1) = O(k^2) \) connections may need to be created between the output of a Skolem functions generated at some recursion level and the input of \( E \) at a lower level of recursion. Therefore, constructing the entire Skolem function vector at recursion level \( 1 \) requires time (and hence space) in \( O(k^2 \cdot |C|) \).

**Lemma 5.** Let \( \Psi(I) \) be a Skolem function vector for \( X \) in \( C(X,I) \). Define \( G(I) \) to be \( C(\Psi(I),I) \), \( \bar{F}(X,I) \) to be \( \land_{i=1}^{|X|} (x_i \land \psi_i) \lor (\neg x_i \land \neg \psi_i) \), and \( H(X,I) \) to be \( \bar{F}(X,I) \lor \neg C(X,I) \). Then \( H(X,I) \) is in \( \text{SAUNF} \) and both \( H(X,I) \) and \( C(X,I) \) represent semantically equivalent specifications.

**Proof.** We first show that \( C(X,I) \iff H(X,I) \). This involves showing two implications.

- \( H(X,I) \implies C(X,I) \): We know from the definitions that \( G(I) \iff C(\Psi(I),I) \iff \forall X(\bar{F}(X,I) \iff C(X,I)) \).

The last implication follows from the observation that \( \bar{F}(X,I) \) simply asserts that \( \land_{i=1}^{|X|} (x_i \iff \psi_i) \), and hence \( C(\Psi(I),I) \iff \forall X(\bar{F}(X,I) \iff C(X,I)) \).

We also know from the definition of \( H(X,I) \) that \( H(X,I) \iff G(I) \) and \( H(X,I) \iff (\bar{F}(X,I) \lor \neg C(X,I)) \).

However, since \( G(I) \iff \forall X(\bar{F}(X,I) \iff C(X,I)) \), it follows that \( H(X,I) \iff (G(I) \iff C(X,I)) \).

- \( C(X,I) \implies H(X,I) \): We know that \( C(X,I) \iff \exists X(\bar{C}(X,I) \iff C(\Psi(I),I)) \iff G(I) \) by definition. It
follows that \( C(X, I) \Rightarrow (C(X, I) \land G(I)) \). However, from the definition of \( H(X, I) \), we know that 
\( (C(X, I) \land G(I)) \Rightarrow H(X, I) \). Hence, \( C(X, I) \Rightarrow H(X, I) \).

To show that \( H(X, I) \) is in SAUNF, note that
\[
\bar{F}(X, I) \equiv \bigwedge_{i=1}^{n} ((x_i \land \psi_i) \lor (\neg x_i \land \neg \psi_i))
\]
can be represented as a SAUNF circuit. Specifically, we represent
\[
\bigwedge_{i=1}^{n} ((x_i \land \psi_i) \lor (\neg x_i \land \neg \psi_i))
\]
in the natural way as a 3-level circuit with an \( \land \)-gate at the root having
\( n \) children, each labeled \( \lor \). The \( i \)-th child of the root
has two children labeled \( \land \) and representing the subfunctions
\( (x_i \land \psi_i) \) and \( (\neg x_i \land \neg \psi_i) \) respectively.
It is easy to see that every \( x_i \) and \( \neg x_i \) is \( \land \)-unrealizable
in this circuit. Hence, the circuit is in SAUNF w.r.t. \( X \)
and the sequence of literal-consistent subsets of leaves
\( S = (S_1, S_2, \ldots, S_{2^n}) \), where \( S_{2i-1} \) is the set of all leaves
labeled \( x_i \) and \( S_{2i} \) is the set of all leaves labeled \( \neg x_i \),
for \( 1 \leq i \leq n \).

By Lemma 7, we also know that
\[
\bar{F}(X, I) \mid_{x_1 = 1, \neg x_1 = 1, \ldots, x_n = 1, \neg x_n = 1} \Leftrightarrow \exists X \bar{F}(X, I) \Leftrightarrow 1.
\]
The last equivalence follows from the definition of
\( \bar{F}(X, I) \); specifically \( \bar{F}(\Psi(I), I) = 1 \) for all values of
\( I \). It now follows from Definition 3 that the circuit
\( (\bar{F}(X, I) \lor C(X, I)) \land G(I) \) is in SAUNF w.r.t. \( X \)
and the sequence of literal-consistent subsets of leaves
\( S = (S_1, S_2, \ldots, S_{2^n}) \) described above. To see why this is
so, let \( R \) denote the circuit \( \bar{F}(X, I) \lor C(X, I) \land G(I) \).
Since each \( S_i \) is \( \land \)-unrealizable in \( \bar{F}(X, I) \), since
\( \bar{F}(X, I) \) is combined with \( C(X, I) \) through an \( \lor \)-gate,
and since \( G(I) \) is semantically independent of \( X \),
the first 3 conditions of Definition 3 are easily seen
to be satisfied. To check if the fourth condition
also holds, note that \( R \mid_{S_1:1, S_2:1, \ldots, S_{2^n}:1} \)
represents
\[
(\bar{F}(X, I) \mid_{x_1 = 1, \neg x_1 = 1, \ldots, x_n = 1, \neg x_n = 1} \lor C(X, I) \lor G(I)) \Leftrightarrow G(I),
\]
which is semantically independent of \( X \).
This shows that condition 4 of Definition 3 holds,
and, hence \( R \) is in SAUNF w.r.t. \( X \) and the sequence
\( S = (S_1, S_2, \ldots, S_{2^n}) \).

\[\] \[\]

**Theorem 4.** Given a relational specification \( C(X, I) \),
1. A Skolem function vector can be computed in polynomial
time if a semantically equivalent SAUNF form
exists and can be computed in polynomial time.
2. A Skolem function vector has polynomial size if there
exists a semantically equivalent SAUNF form of the
specification that is polynomial sized.

**Proof.** 1. By Theorem 3, we can generate a Skolem
function vector from a specification represented in
SAUNF in time polynomial in the size of the rep-
resentation. Therefore, if a semantically equivalent
SAUNF form of the specification can be computed
in polynomial time, we can obtain a Skolem function
vector in polynomial time. In the reverse direction,
if a Skolem function vector \( \Psi(I) \) can be computed in
time polynomial in the size of the specification, we
can construct the semantically equivalent specification
\( H(X, I) = (\bar{F}(X, I) \lor C(X, I)) \land G(I) \), which by
Lemma 5, is in SAUNF. Clearly, \( G(Y) = C(\Psi(I), I) \)
and \( \bar{F}(X, I) \) can be obtained in polynomial time if the
Skolem function vector \( \Psi(I) \) can be computed in polynomial
time. This proves the first part of the theorem.

2. By Theorem 3, we can generate a polynomial sized
Skolem function vector from a polynomial sized
SAUNF form. In the reverse direction, we consider
the semantically equivalent form \( H(X, I) = (\bar{F}(X, I) \lor C(X, I)) \land G(I) \), which by Lemma 5, is
already in SAUNF. Note that \( G(Y) = C(\Psi(I), I) \)
and \( \bar{F}(X, I) \) are both polynomial-sized given a
polynomial-sized Skolem function vector.

---

**Lemma 8.** Suppose \( G(X, I) \) is in SAUNF w.r.t. \( X \)
and a sequence \( S^G \), and \( H(X, I) \) is in SAUNF w.r.t. \( X \)
and \( S^H \). If there is no literal \( \ell \) over \( X \) such that \( G \)
has an \( \ell \)-labeled leaf and \( H \) has a \( \neg \ell \)-labeled leaf,
combining the outputs of \( G \) and \( H \) by an \( \land \)-gate gives a
"circuit in SAUNF w.r.t. \( X \) and \( (S^G, S^H) \). Otherwise,
an SAUNF circuit semantically equivalent to \( G \land H \)
cannot be constructed in time polynomial in \(|G|, |H| \) unless
\( P = NP \). Such a circuit cannot have size polynomial in
\(|G|, |H| \) unless \( \exists^P = \Sigma^2_2 \) (i.e. the polynomial hierarchy
collapses to the second level).

**Proof.** Since there is no literal \( \ell \) over \( X \) such that \( G \)
has an \( \ell \)-labeled leaf and \( H \) has a \( \neg \ell \)-labeled leaf, it is
easy to see that a leaf of \( G \) and a leaf of \( H \) cannot
participate together in making a literal \( \land \)-realizable in
\( G \land H \). Since \( G \) is in \( \text{SAUNFw.r.t.} \ X \) and \( S^G \),
and \( H \) is in \( \text{SAUNFw.r.t.} \ X \) and \( S^H \), it then follows that \( G \land H \)
is also in \( \text{SAUNFw.r.t.} \ X \) and the sequence \( (S^G, S^H) \)
(or alternatively, \( (S^H, S^G) \)).

The proof for the remainder of the lemma involves
certain definitions of transformations.

**Definition 6.** For a circuit \( F(X) \) with variable set
\( x_1, x_2, \ldots, x_n \), we define \( F^-(X, X') \) as the circuit obtained by replacing each instance of \( \neg x_i \) at all leaves with a corresponding new variable \( x'_i \) for all \( i \).

**Claim 1.** For any circuit \( F(X), F^-(X, X') \) is in
\( \text{SAUNFw.r.t.} \ X \cup X' \) for any sequence of literal-
consistent subsets of leaves.

**Proof.** For any \( F^-(X, X') \), there is no literal label of a
leaf, whose complement is also the label of some other
leaf in the circuit. This completely eliminates the pos-
sibility of subset of literal-consistent leaves being \( \land \)-
realizable in \( F^-(X, X') \).

**Definition 7.** For any circuit \( F(X) \) with variable
\( x_1, x_2, \ldots, x_n \), \( F'(X, X') \) is defined as \( \bigwedge_{i=1}^{n} (x'_i \land \neg x_i) \lor \ldots \lor \)
Consider the circuit by Claims 1, 2.

**Claim 2.** For any circuit $F(X)$, $F'(X, X')$ is in $\text{SAUNF} w.r.t. X \cup X'$ and the sequence of literal-consistent leaves $S = (S_1, \ldots, S_{2n}, S_1', \ldots, S_{2n}' )$, where $S_{2i-1}$ (resp. $S_{2i-1}'$) is the set of all leaves labeled $x_i$ for $i \leq n$.

By Claims 1, $F(X)$ is equivalent to $\bigwedge_{i=1}^{n} (x'_i \leftrightarrow \neg x_i)$. Hence, the circuit is in $\text{SAUNF}$. Moreover $F'|_{S_1:1, S_2:1, \ldots, S_{2n}:1} = 1$, and hence also w.r.t. $X \cup X'$ and the sequence $S = (S_1, \ldots, S_{2n}, S_1', \ldots, S_{2n}')$ as well.

**Proof.** It is easy to see that for the sequence $(S_1, S_2, \ldots, S_{2n})$, the first three conditions of Definition 3 are satisfied. Moreover $F'|_{S_1:1, S_2:1, \ldots, S_{2n}:1} = 1$.

Theorem 5. 1. First, we show that identifying whether a given circuit is in $\text{SAUNF}$ for a given sequence of subsets of leaves is in coNP. This is equivalent to asking whether the complement problem, i.e. if the given circuit is not in $\text{SAUNF}$ for the given sequence of subsets of leaves, is in NP. We will define a non-deterministic polynomial-time machine $M$ that solves this complement problem. The machine $M$ first checks whether the input circuit (say $C$) is in NNF. This can be done by checking in each internal node of $C$ is labeled either $\land$ or $\lor$, and if all negations (if any) are on the labels of leaves. Clearly, this check can be done in time polynomial in the size of $C$. If the circuit is found to be not in NNF, the machine $M$ accepts, since $C$ cannot be in $\text{SAUNF}$ in this case. Otherwise (i.e. if $C$ is in NNF), the machine $M$ non-deterministically chooses a subset $S_i$ of literal-consistent leaves in the given sequence $S$ and executes the following operations.}

Suppose the literal labeling of the subset $S_i$ is $\ell_i$. The machine $M$ (a) constructs the circuit $C' = C|_{S_1:1, \ldots, S_{i-1}:1}$, (b) sets all leaves of $C'$ that are not in $S_i$ but are labeled $\ell_i$ to 0, (c) replaces all remaining labels $\ell$ on leaves by $w$ and all labels $\neg \ell$ on leaves by $w'$, (d) guesses an assignment $\sigma$ to all variables other than $w$ and $w'$ labeling leaves in the resulting circuit, and (e) checks if the resulting circuit represents the Boolean function $w \land w'$ for the assignment $\sigma$ to other variables. Note that after step d, the resulting circuit represents a function of only $w$ and $w'$ (at most). Hence the check in step (e) can be performed by setting $(w, w')$ to each of (1,1), (1,0), (0,1) and (0,0) and checking if the resulting circuit evaluates to 1, 0, 0 and 0 respectively. Clearly, all the steps from (a) through (e) above can be done in time polynomial in the size of the circuit $C$. If after step (e), the resulting circuit is found to represent $w \land w'$, then machine $M$ accepts. In this case, $C$ is not in $\text{SAUNF}$ w.r.t. $X$ and the given sequence $S$ of subsets of leaves. Conversely, if the circuit is not in $\text{SAUNF}$ w.r.t. $X$ and the given sequence $S$ of subsets of leaves, it is not $\text{SAUNF}$ for any sequence $S$ of literal-consistent leaves in the assignment described above. Hence the problem of identifying whether a circuit is not in $\text{SAUNF}$ for a given sequence of subsets of leaves is in NP. Therefore, the problem of identifying whether $C$ is in $\text{SAUNF}$ for a given sequence of subsets of leaves is in coNP.

Next, we show that the problem is co-NP hard. We reduce the problem of identifying whether a propositional formula represented by a CNF circuit is unsatisfiable to identifying whether an appropriately constructed circuit is in $\text{SAUNF}$ for a specific sequence of subsets of literal-consistent leaves. For this reduction, we consider the specification represented by the circuit $C \land \neg x$, where $x$ is the sole output of the specification, and the inputs are the variables labeling leaves of $C$. Since there is only one output variable, there are only two (equivalent) ordering of subsets of
leaves labeled by output literals. It is easy to see that $x$ (equivalently, $\neg x$) is $\land$-realizable if and only if $C$ is satisfiable. Hence, identifying whether a problem is in SAUNF for a given sequence of subsets of leaves is coNP-hard.

2. To prove the second part of the theorem, we use the following result from the polynomial hierarchy: $\Sigma_2^P = NP^{NP^P}$.

Specifically, we show that checking whether a given circuit $C$ is in SAUNF w.r.t. some (unspecified) sequence of subsets of output literal-consistent leaves can be solved by an non-deterministic polynomial-time Turing machine $M$ that invokes an NP oracle, i.e., $\in NP^{NP^P}$. Given $C(X, I)$, the machine $M$ operates as follows.

(a) Guesses an sequence $S$ of subsets of literal-consistent leaves.

(b) Reduces the problem of deciding whether $C(X, I)$ is not in SAUNF w.r.t $X$ and the sequence $S$ to checking the satisfiability of an appropriately constructed propositional formula $\varphi$. This reduction is similar to what we discussed above in the proof of part (1).

(c) Feeds $\varphi$ to the NP oracle

(d) Accepts if and only if the NP oracle rejects

It follows that $M$ accepts if and only if there exists an sequence $S$ of subsets of output literal-consistent leaves for which $C$ is in SAUNF w.r.t. $X$ and $S$. Hence, we have proved that our problem is contained in $NP^{NP^P}$.

The proof that the problem is coNP hard is almost the same as the corresponding proof with a given sequence of literal-consistent subsets of leaves. We reduce the problem of identifying whether a formula represented by a CNF circuit is unsatisfiable to identifying whether the circuit is in SAUNF for some sequence of subsets of literal-consistent leaves. However, recall from the coNP-hardness proof of part (1) that there are only two (equivalent) sequences of subsets of output-literal-consistent leaves of the circuit $C \land x \land \neg x$, where $C$ is a CNF circuit representing a propositional formula whose unsatisfiability we wish to check. Hence $C \land x \land \neg x$ is in SAUNF w.r.t. some (there are only two possibilities in our case) sequence $S$ of subsets of output literal-consistent leaves iff $C$ is unsatisfiable. This proves coNP-hardness of the problem.

\[ \text{Theorem 6.} \quad \text{For } i \leq j \leq 2n, j - i < n, R[i, j] \text{ is representable by a polynomial (in } n \text{) sized SAUNF circuit.} \]

\[ \text{Proof.} \quad \text{By Theorem 4, we can generate a polynomial-sized SAUNF circuit in polynomial time given an explicit polynomial-sized Skolem function vector. Therefore, we focus on obtaining an explicit polynomial-sized Skolem function vector for } R[i, j], i \leq j \leq 2n, j - i < n. \]

We use the notation $X_i$ to denote the $i^{th}$ least significant bit of $X$ and $X_{i,j}$ denotes the bit-slice of $X$ from bits $i$ to $j$ (both included). We use similar notations for any other bit vectors $Y, I$. Now, we give explicit polynomial-sized Skolem function vectors for $R[i, j], i \leq j \leq 2n, j - i < n$. We consider the following two cases:

- $i \leq n$: The Skolem function vector for $X, Y$ is given by
  \[ \Psi_{X_i} = \begin{cases} 0 & \text{if } k \neq i-1 \\ 1 & \text{if } k = i-1 \end{cases} \]
  \[ \Psi_{Y_i} = \begin{cases} 0 & \text{if } k \leq j + 1 - i \\ 1 & \text{if } k > j + 1 - i \end{cases} \]

Intuitively, $X = 2^{i-1}, Y = I_{i,j}$. Clearly this results in the matching of bits $i$ to $j$ as required.

- $i > n$: Following a similar logic as above, the Skolem function vector for $X, Y$ is given by
  \[ \Psi_{X_i} = \begin{cases} 0 & \text{if } k \neq n \\ 1 & \text{if } k = n \end{cases} \]
  \[ \Psi_{Y_i} = \begin{cases} 0 & \text{if } k \leq i - n \\ 1 & \text{if } k = i - n \end{cases} \]

Intuitively, $X = 2^{i-1}, Y = 2^{i-n} \times I_{i,j}$. Clearly this results in the matching of bits $i$ to $j$.

Having generated a Skolem function vector for each specification $R[i, j], j - i < n$, we can generate a corresponding SAUNF circuit. The size of the Skolem function vector is clearly polynomial in $n$. By Theorem 4, this proves the theorem.

\[ \text{Theorem 7.} \quad \text{The relation } X = I / Y, \text{ with inputs } I, Y \text{ restricted to odd numbers (i.e. the relation evaluates to 0 if } I \text{ or } Y \text{ is even), is representable as a polynomial (in } n \text{) sized SAUNF circuit.} \]

\[ \text{Proof.} \quad \text{We give a skolem function vector for division, which will generate a SAUNF form by Theorem 4. Now, suppose we have inputs } I, Y \text{ and we have to find } X \text{ such that } X \times Y = I. \text{ We first show that for odd inputs, if } (X \times Y) \text{ mod } 2^n = I \text{ mod } 2^n \text{ and if } I \text{ is divisible by } Y \text{ then } X \times Y = I. \text{ Suppose there are two values } X_1, X_2 \text{ such that } (X_1 \times Y) \text{ mod } 2^n = I \text{ mod } 2^n \text{ and } (X_2 \times Y) \text{ mod } 2^n = I \text{ mod } 2^n. \text{ Then } (X_1 - X_2) \times Y = 0 \text{ mod } 2^n. \text{ However, } Y \text{ is odd, and therefore co-prime to } 2^n, (X_1 - X_2) \equiv 0 \text{ mod } 2^n. \text{ Since, } X_1, X_2 < 2^n, X_1 = X_2. \text{ Therefore the generated } X \text{ from the skolem function is correct if there exists a solution that matches the least significant n bits of } I. \text{ Using the notation defined above, denote } X_i \text{ to be the } i^{th} \text{ least significant bit of } X \text{ and } X_{i,j} \text{ to denote the bit vector from } X_i \text{ to } X_j \text{ (both included)). Now, since the input is odd, } Y_1 = 1, I_1 = 1. \text{ Therefore } \Psi_{X_1} = 1. \]

Now note that $(X \times Y)_i = (X_{i,1} \times Y_{1,i}) = (X_{1,i-1} \times Y_{1,i} + X_{1,j} \times Y_{1,i} \times 2^{i-1})_i = X_{1,i} \oplus (X_{1,i-1} \times Y_{1,i})_i \text{ (using the structure of multiplication). Therefore } I_i = X_i \oplus (X_{1,i-1} \times Y_{1,i})_i. \text{ Therefore, } X_i = I_i \oplus (X_{1,i-1} \times Y_{1,i})_i.
Therefore, given the skolem function for $\Psi_{X_1} = 1$ to $X_1$, giving us the skolem function vector for $X$.

Note that while we use particular skolem functions in generating the SAUNF form, the form can generate other skolem functions as well (from Algorithm 4).

Guarantees in Algorithms 2, 3: We formally give statements on Algorithms 2, 3 and give the proofs, elaborating on the general description given.

Lemma 9. Algorithm 2 returns an $\land$–unrealizable subset of $\ell$–leaves and takes a worst-case exponential (in $|C|$) time.

Proof. First, we show that it always returns an $\land$–unrealizable subset. We give a proof by contradiction. Suppose that the set $T$ is $\land$–realizable for an assignment $\sigma'$ in $C$ ($T = \text{GetClausesEvaluatingToL}(C, \sigma', \ell)$). We show that $D$ will also be $\land$–realizable for $\ell$. Under the assignment $\sigma'$, all clauses that contain leaves in $\text{HitS}$ evaluate to 1 irrespective of $\ell$ as $\text{HitS} \cap T = \emptyset$ and a clause containing $\ell$ can evaluate to only $\ell$ or 1. Therefore, for all $S \in \text{AllS}$, $\text{DisjoinWithoutLit}(S, \ell)$ evaluates to 1 as at the clause corresponding to the leave in $\text{HitS}$ evaluates to 1. Therefore, for an assignment $\sigma'$, $D$ will also be $\land$–realizable if $C$ is also $\land$–realizable for the same assignment. However, this is a contradiction to the breaking condition of the loop that $D$ is not $\land$–realizable.

Now, we look at the worst case running time of the Algorithm. Suppose that a set $S$ is pushed into $\text{AllS}$. Now, $D$ is conjuncted with $\text{DisjoinWithoutLit}(S, \ell)$. For any assignment $\sigma'$, if $S = \text{GetClausesEvaluatingToL}(C, \sigma', \ell)$, then $\text{DisjoinWithoutLit}(S, \ell)$ evaluates to 0. Therefore, any set $\text{CurrS} \neq S$ at line 5 as $D$ cannot be $\land$–realizable for such an assignment. Therefore, the number of iterations is the number of subsets of $\ell$–leaves in $C$, which is exponential in $|C|$. Each iteration involves a SAT call. Therefore, the worst case complexity is exponential in $|C|$.

However, note that if we can modify the breaking condition of the loop to involve a TimeOut. In such a case, we return the 0. This reduces the worst case complexity, providing a tradeoff between the running time and precision of the algorithm.

We now state the theorem associated with Algorithm 3.

Lemma 10. Algorithm 3 returns a SAUNF circuit.

Proof. We follow a proof similar to the argument given in Section 6. In the circuit $F$ constructed in line 11 of Algorithm 3, the set containing only the leaf $\ell$ (call it $L_1$) conjoined with $C_1$ is $\land$–unrealizable as it meets up $\sim\ell$ at a $\lor$ gate. In $F_{L_1}$, we can show that set $S'\ell$ of $\ell$–leaves of $C'$ is $\land$–unrealizable as it was already $\land$–unrealizable in the circuit $C$. For $F_{L_1}$ to be $\land$–realizable for an assignment $C'$ should evaluate to $\ell$, $C_1$ to 0 and $C_2$ to 1. We show that if such a case is possible for an assignment, then $S$ would have been $\land$–realizable in $C$ for that assignment, which is not possible by Lemma 9. We divide the clauses in $C$ into 4 sets: (i) Clauses in $S$, (ii) Clauses that have positive $\ell$–leaves not in $S$, (iii) Clauses with the negative literals $\sim\ell$ (iv) Clauses that have no dependence on $\ell$. We look at assignments to each of these clauses in $C$ for the particular assignment. $C'$ represents the circuit for clauses in (i) which evaluates to $\ell$ for $\land$–realizability in $F_{L_1}$. Since $C_2$ evaluates to 1, all clauses in $C_2$ in evaluate to 1. Therefore, the clauses in which $\ell = 0$ ($C_2$ contains only those clauses with $\ell$–leaves not in $S$) or have no dependence to $\ell$ evaluate to 1. Therefore, circuits corresponding to clauses in (ii) and (iv) evaluate to 1. Since (iv) evaluates to 1, but $C_1$ is 0, at least one with $\sim\ell$ set to 0 evaluates to 0. Therefore, in circuit $C$, it will evaluate to $\sim\ell$ and therefore, circuit correspond to (iii) evaluates to $\sim\ell$. This shows that $C$ would have been $\land$–realizable in $C$, which is not possible by Lemma 9. We set $S'\ell$ of $\ell$–leaves of $C'$ to 1.

In $F_{L_1, S'\ell}$, all positive leaves of $\ell$ have been set to 1, and therefore we can set the only $\sim\ell$ (denote $L' = \{\sim\ell\}$ to 1. $F_{L_1, S'\ell, L' \lor 1} = C_1 \lor C_2$, which is in SAUNF assuming the recursive calls return correct representations and using Lemma 6. The base condition is when $C$ is independent of $X$, for which it already returns the correct value.

This algorithm represents a significant reduction in size over Shannon expansion, as we expand the circuit $D$ obtained after setting certain clauses of $C$ to 1 (line 6 of Algorithm 3). In the worst-case, however, this reduces to a Shannon-expansion based conversion of $C$.

A Pseudocode
Algorithm 4: SkGen($C, S, r$)

Input: $C(X, Y)$: Relational spec, $\Psi_C(Y)$: Skolem function for $X$ in $C$; $n = |X|$
$r$: Recursion level
Output: $\Psi_C(Y)$: Skolem function vector for $C$

1. $SF = \text{GetSAUNF}(\Psi, C)$;
2. if $r = 2n$ then
   3. $\Psi_C(Y) := \neg\Psi_C(Y)$;
3. else
   4. $S := \land \neg$-unrealizable Literal label of leaves;
   5. $D := \text{CPropSimp}(C |_{S=1})$;
      // CPropSimp propagates constants and eliminates gates with constant outputs
   6. $\Psi_D(Y) = \text{SkGen}(D, S, r + 1)$;
   7. $\text{temp} := \text{CPropSimp}(E(\Psi_D(Y), \ell = \neg\psi(Y), Y))$;
   8. $\psi(Y) = (\neg\psi(Y) \land \text{temp}) \lor (\psi(Y) \land \neg\text{temp})$;
10. return $\Psi_C(Y)$;