# On synthesizing Skolem functions for first order logic formulae

**S. Akshay**
Indian Institute of Technology Bombay, India

**Supratik Chakraborty**
Indian Institute of Technology Bombay, India

## Abstract

Skolem functions play a central role in logic, from eliminating quantifiers in first order logic formulas to providing functional implementations of relational specifications. While classical results in logic are only interested in their existence, the question of how to effectively compute them is also interesting, important and useful for several applications. In the restricted case of Boolean propositional logic formula, this problem of synthesizing Boolean Skolem functions has been addressed in depth, with various recent work focussing on both theoretical and practical aspects of the problem. However, there are few existing results for the general case, and the focus has been on heuristical algorithms.

In this note, we undertake an investigation into the computational hardness of the problem of synthesizing Skolem functions for first order logic formula. We show that even under reasonable assumptions on the signature of the formula, it is impossible to compute or synthesize Skolem functions. Then we determine conditions on theories of first order logic which would render the problem computable. Finally, we show that several natural theories satisfy these conditions and hence do admit effective synthesis of Skolem functions.

**2012 ACM Subject Classification** Theory of computation → Logic and verification

**Keywords and phrases** Skolem functions, computability, first-order logic

## 1 Introduction

The history of Skolem functions can be traced back to 1920, when the Norwegian mathematician, Thoralf Albert Skolem, gave a simplified proof of a landmark result in logic, now known as the *Löwenheim-Skolem* theorem. Leopold Löwenheim had already proved this theorem in 1915. However, Skolem's 1920 proof was significantly simpler and made use of a key observation: *For every first order logic formula $\exists y\, \varphi(x, y)$, the choice of $y$ that makes $\varphi(x, y)$ true (if at all) depends on $x$ in general. This dependence can be thought of as implicitly defining a function that gives the "right" value of $y$ for every value of $x$. If $F$ denotes a fresh function symbol, the second order sentence $\exists F\, \varphi(x, F(x))$ formalizes this dependence explicitly. Thus, the second order sentence $\exists F\, \forall x\, \big(\exists y\, \varphi(x, y) \Rightarrow \varphi(x, F(x))\big)$ always holds.* Since the implication trivially holds in the other direction too, we have

$$\exists F\, \forall x\, \big(\exists y\, \varphi(x, y) \Leftrightarrow \varphi(x, F(x))\big) \tag{1}$$

is valid. Note the relation between the first order formulas $\xi_1 = \exists y\, \varphi(x, y)$ and $\xi_2 = \varphi(x, F(x))$:

- While $\xi_2$ has one less existential quantifier than $\xi_1$, the signature of $\xi_2$ has one more function symbol than the signature of $\xi_1$. Thus, an existential quantifier has been traded off, so to say, for a function symbol.
- Though $\xi_1$ and $\xi_2$ are not semantically equivalent, for every assignment of the free variable $x$, the formula $\xi_1$ is satisfiable iff $\xi_2$ is.

- Every model $\mathfrak{M}$ of $\forall x\, \xi_1$ can be augmented with an interpretation of $F$ to yield a model $\mathfrak{M}'$ of $\forall x\, \xi_2$. Similarly, for every model $\mathfrak{M}'$ of $\forall x\, \xi_2$, restricting $\mathfrak{M}'$ to the signature of $\xi_1$ yields a model $\mathfrak{M}$ of $\forall x\, \xi_1$.

The process of transforming $\xi_1$ to $\xi_2$ by eliminating $\exists y$ and substituting $F(x)$ for $y$ is an instance of Skolemization. The fresh function symbol $F$ introduced in the process is called a Skolem function. Skolem functions play a very important role in logic – both in theoretical investigations and in practical applications. While it suffices in some (especially theoretical) studies to simply know that a Skolem function $F$ exists, in other (especially practically significant) cases, we require an algorithm that effectively computes $F(x)$ for every $x$.

This question of synthesizing Skolem functions has been studied in depth for the Boolean setting, i.e., quantified propositional logic, with an impressive array of recent results [10, 12, 11, 7, 14, 17, 4, 15, 3, 1, 13, 2, 8]. Called the Boolean functional synthesis problem in this setting, the problem is posed as given a Boolean relational specification, to synthesize Boolean Skolem functions that implement it. This problem has a wide-ranging set of applications [16] already from certified QBF to factorization to disjunctive decomposition of circuits. All known algorithms for the problem are exponential in the worst-case and in [3], it was also shown that the problem cannot have a sub-exponential algorithm unless some complexity-theoretic conjectures are falsified. Despite this, the practical performance of algorithms on real benchmarks has led to questions about the structure of the input that leads to this performance. This has resulted in investigations into knowledge representations that give polytime synthesis algorithms [1] and still remains an active line of research. Thus the theoretical investigations have gone hand-in-hand with improved algorithms and understanding of the problem.

Surprisingly, a similar theoretical treatment beyond this restricted Boolean setting seems lacking, despite the existence of several potential applications. In this paper, we move towards closing this gap. We go beyond the Boolean setting and address the Skolem function synthesis problem in the full generality of first order logic. That is, does there always an algorithm to synthesize Skolem functions of quantified first order formulae? Unfortunately, our first result is to show that Skolem functions cannot be computed in general. We show this by giving a reduction from the classical Post's correspondance problem known to be undecidable. However, this impossibility result requires having an uninterpretted predicate. We then strengthen this result by showing that even if all predicates and function symbols are interpretted, the problem continues to be intractable, by showing a novel reduction using Diophantine sets. Given these impossibility results, we turn our attention to different subclasses of the first order logic (still beyond or incomparable to the propositional case) that are of interest. We establish sufficient conditions that classes must have for Skolem functions to be computable. We then exhibit several natural theories of first-order logic that satisfy these conditions and hence show that Skolem functions can be effectively synthesized for them. All this results in a nuanced picture of the computability landscape for the Skolem function synthesis problem. We hope that this work will be a starting point towards further research into the design of practical algorithms (when possible, i.e., within these subclasses) to synthesize Skolem functions for first order logic and its applications.

## 2 Preliminaries and Problem Statement

We start by fixing some notations. We will use lower case English letters, e.g, $x$, $y$, $z$, possibly with subscripts, to denote first order variables, and bold-faced upper case English letters, viz. $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$, to denote sequences of first order variables. Lower case Greek letters, e.g., $\varphi$, $\xi$, $\alpha$, possibly with subscripts, will be used to denote formulas. For a sequence $\mathbf{X}$, we use $|\mathbf{X}|$ to denote the count of variables in $\mathbf{X}$, and $x_1, \ldots x_{|X|}$ to denote the individual variables in the sequence. With a slight abuse of notation, we also use $|\varphi|$ to denote the size of the formula $\varphi$, represented using a suitable format (viz. as a string, syntax tree, directed acyclic graph etc.), when there is no confusion.

For a quantifier $Q \in \{\exists, \forall\}$ be a quantifier, we use $Q\mathbf{X}$ to denote the block of quantifiers $Qx_1 \ldots Qx_{|\mathbf{X}|}$. It is a standard exercise in logic to show that every well-formed first order logic formula can be transformed to a semantically equivalent *prenex normal form*, in which all quantifiers appear to the left of the quantifier-free part of the formula. Without loss of generality, let $\xi(\mathbf{X}) \equiv \exists\mathbf{Y}\ \forall\mathbf{Z}\ \exists\mathbf{U} \ldots \forall\mathbf{V}\ \exists\mathbf{W}\, \varphi(\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{U}, \ldots \mathbf{V}, \mathbf{W})$ be such a formula in prenex normal form, where $\mathbf{X}$ is a sequence of free variables and $\varphi$ is a quantifier-free formula. In case the leading (resp. trailing) quantifier in $\xi$ is universal, we consider $\mathbf{Y}$ (resp. $\mathbf{W}$) to be the empty sequence. Given such a formula $\xi$, *Skolemization* refers to the process of transforming $\xi$ to a new (albeit related) formula $\xi^\star$ without any existential quantifiers via the following steps: (i) for every existentially quantified variable, say $a$, in $\xi$, substitute $F_a(\mathbf{X}, \mathbf{S}_a)$ for $a$ in the quantifier-free formula $\varphi$, where $F_a$ is a new function symbol and $\mathbf{S}_a$ is a sequence of universally quantified variables that appear to the left of $a$ in the quantifier prefix of $\xi$, and (ii) remove all existential quantifiers from $\xi$. The functions $F_a$ introduced above are called *Skolem functions*. In case $\xi$ has no free variables, i.e. $\mathbf{X}$ is empty, the Skolem functions for variables $y_i$ in the leftmost existential quantifier block of $\xi$ have no arguments (i.e. are nullary functions), and are also called *Skolem constants*. The sentence $\xi^\star$ is said to be in *Skolem normal form* if the quantifier-free part of $\xi^\star$ is in conjunctive normal form. For notational convenience, let $\exists\mathfrak{F}$ denote the second order quantifier block $\exists F_{y_1} \ldots \exists F_{y_{|Y|}} \cdots \exists F_{w_1} \ldots \exists F_{w_{|W|}}$ that existentially quantifies over all Skolem functions introduced above. The key guarantee of Skolemization is that the second order sentence $\exists\mathfrak{F}\,\forall\mathbf{X}\big(\xi \Leftrightarrow \xi^\star\big)$ always holds. Note that substituting Skolem functions for existentially quantified variables need not always make the quantifier-free part of $\xi$, i.e. $\varphi$, evaluate to true. This can happen, for example, if there are valuations of universally quantified variables for which no assignment of existentially qualified variables renders $\varphi$ true. For every other valuation of universally quantified variables, the Skolem functions indeed provide the "right" values of existentially quantified variables so that $\varphi$ evaluates to true.

▶ **Example 1.** Consider $\xi \equiv \exists y \forall x \exists z \forall u \exists v\, \varphi(x, y, z, u, v)$. On Skolemizing, we get $\xi^\star \equiv \forall x \forall u\, \varphi(x, C_y, F_z(x), u, F_v(x, u))$, where $C_y$ is a Skolem constant for $y$, and $F_z(x)$ and $F_v(x, u)$ are Skolem functions for $z$ and $v$ respectively.

## 2.1 Relative computation and the problem statement

As mentioned earlier, the focus of this article is on effective computation of Skolem functions. In other words, given a first order formula $\xi$, does there always exist a halting Turing machine that computes each Skolem function appearing in a Skolemized version of $\xi$? In general, such a Turing machine (or algorithm) may need to evaluate predicate and function symbols that appear in the signature of $\xi$ as part of its computation. Therefore, the most appropriate notion of computation in our context is that of *relative computation* or *computation by oracle machines*[1]. Formally, we define our problem of interest as follows:

▶ **Definition 2.** *Let $\mathcal{P}_\xi$ and $\mathcal{F}_\xi$ denote the set of predicate and function symbols respectively*

[1] See [5] for a detailed exposition on relative computability.

*in the signature of a first order logic formula $\xi$. Given oracles for interpretations of predicate symbols in $\mathcal{P}_\xi$ and of function symbols in $\mathcal{F}_\xi$, the* Skolem function synthesis problem *asks if every Skolem function $F$ in a Skolemized version of $\xi$ can be computed by a halting Turing machine, say $M_\xi^F$, with access to these oracles.*

Note that we require $M_\xi^F$ to depend only on $\xi$ and $F$. However, the oracles that $M_\xi^F$ accesses can depend on specific interpretations of predicate and function symbols.

## 3 Uncomputability results

Our first result is that $M_\xi^F$ does not always exist for every $\xi$ and $F$. In other words, Skolem functions cannot be effectively computed in general, even in the relative sense mentioned above.

▶ **Theorem 3.** *The Skolem function synthesis problem for first order logic is uncomputable, even if the signature has only a single unary uninterpreted predicate.*

**Proof.** We show a reduction from *Post's Correspondence Problem* (PCP, in short) [9] – a well-known undecidable problem to solvability of the Skolem function synthesis problem. An instance $\pi$ of PCP consists of a finite set $\Gamma = \{(\alpha_1, \beta_1), \ldots (\alpha_k, \beta_k)\}$, where $\alpha_i$ and $\beta_i$ are finite (possibly empty) strings over $\{0, 1\}$. Solving the PCP instance $\pi$ requires determining if there exists a finite sequence of indices $i_1 i_2 \ldots i_r$ with $1 \leq i_j \leq k$ for all $i_j$, such that $\alpha_{i_1} \cdot \cdots \cdot \alpha_{i_r} = \beta_{i_1} \cdot \cdots \cdot \beta_{i_r}$, where '$\cdot$' denotes string concatenation. To reduce PCP to relative computation of Skolem functions, we consider the first-order sentence $\xi \equiv \exists y\, P(y)$, where $P$ is a unary predicate symbol. Skolemizing $\xi$, we obtain $P(c)$, where $c$ is a Skolem constant. Now suppose, if possible, there exists a Turing machine $M_\xi^c$ with access to an oracle for $P$, that always computes the value of $c$ correctly. Given an instance $\pi$ of PCP, we consider an interpretation of $P$ over the set of all finite strings over $\{0, 1\}$. The corresponding oracle, denoted $P_\pi$, returns "Yes" (or true) on input string $u$ iff there exists a sequence of indices $i_1, \ldots i_r$ with $1 \leq i_j \leq i_k$ for each $i_j$, such that $\alpha_{i_1} \cdot \cdots \cdot \alpha_{i_r} = \beta_{i_1} \cdot \cdots \cdot \beta_{i_r} = u$.

Since the length of $u$ is finite, it is an easy exercise to show that the oracle $P_\pi$ can be simulated by a halting Turing machine, say $M_\pi^P$, without access to any oracle. Therefore, $M_\xi^c$ with access to oracle $P_\pi$ can be simulated by a halting Turing machine, denoted $M_{\xi,\pi}^c$, that needs no access to any oracle. We now design a Turing machine $M_{\mathsf{PCP}}$ that takes as input an instance $\pi$ of PCP and works as follows: $M_{\mathsf{PCP}}$ first writes the encoding of $M_{\xi,\pi}^c$ on a working tape and then runs a universal Turing machine to simulate $M_{\xi,\pi}^c$. Since $M_{\xi,\pi}^c$ is a halting machine, the universal Turing machine must stop after computing a binary string, say $c_\pi$, that serves as the value of the Skolem constant $c$. Subsequently, $M_{\mathsf{PCP}}$ writes the encoding of $M_\pi^P$ on a working tape and runs a universal Turing machine to simulate a run of $M_\pi^P$ on the string $c_\pi$. If the universal machine halts with output "Yes", we know that there exists a sequence of indices $i_1, \ldots i_r$ such that $\alpha_{i_1} \cdot \cdots \cdot \alpha_{i_r} = \beta_{i_1} \cdot \cdots \cdot \beta_{i_r} = c_\pi$. Otherwise, i.e. if the universal machine halts with output "No", then since $c_\pi$ is the correct value of the Skolem constant $c$ when the interpretation of $P$ corresponds to the oracle $P_\pi$, we know that there does not exist any sequence of indices $i_1, \ldots i_r$ such that $\alpha_{i_1} \cdot \cdots \cdot \alpha_{i_r} = \beta_{i_1} \cdot \cdots \cdot \beta_{i_r}$. Therefore, the Turing machine $M_{\mathsf{PCP}}$ decides PCP – a mathematical impossibility! This implies that our assumption was wrong, i.e. the machine $M_\xi^c$ cannot exist. ◀

The above argument shows that Skolem functions cannot be computed in general, even if the signature has only a single unary uninterpreted predicate in the signature. But one may then ask, what would happen if all predicates and functions are interpretted, viz. in the theory of natural numbers with multiplication and addition. This seems a significantly simpler and a natural question to consider. Our second result is that even in this case, Skolem functions cannot be computed.

▶ **Theorem 4.** *The Skolem function synthesis problem is uncomputable, even for first-order logic formulae with all predicates and functions being interpretted.*

**Proof.** The proof in this case appeals to the Matiyasevich-Robinson-Davis-Putnam (MRDP) theorem [6] that equates Diophantine sets with recursively enumerable sets. Formally, it states: *A set of natural numbers is Diophantine if and only if it is recursively enumerable.* Recall that a set $\mathcal{S}$ of natural numbers is Diophantine if there exists a polynomial $P(x, y_1, \dots y_k)$ with integer coefficients such that the Diophantine equation $P(x, y_1, \dots y_k) = 0$ has a solution in the unknowns $y_1, \dots y_k$ iff $x \in \mathcal{S}$. Recall also that a set $\mathcal{S}$ is recursively enumerable if there exists a (potentially non-halting) Turing machine that outputs every element of $\mathcal{S}$ in some order, and only those elements. Now consider the set $S_{halt}$ of natural number encodings of all Turing machines that halt on the empty tape. It is a well-known result in computability theory [9] that $S_{halt}$ is recursively enumerable, although there is no Turing machine that takes a natural number $x$ as input and halts and correctly reports whether $x \in S_{halt}$. By the MRDP theorem, recursive enumerability of $S_{halt}$ implies the existence of a polynomial $P_{halt}(x, y_1, \dots y_k)$ such that $x \in S_{halt}$ iff $\exists y_1, \dots y_k \in \mathbb{N}^k\, P_{halt}(x, y_1, \dots y_k) = 0$. We now consider the first order sentence $\xi_{halt} = \forall x \exists y_1 \dots \exists y_k\, P_{halt}(x, y_1, \dots y_k) = 0$. Note that since $P_{halt}(x, y_1, \dots y_k)$ is a polynomial, it can be written as a term in the first order theory of natural numbers with signature $\{\times, +, 0, 1\}$. Furthermore, on Skolemizing, we get $\xi^\star_{halt} = \forall x\, P_{halt}(x, f_1(x), \dots f_k(x))$, where $f_1, \dots f_k$ are unary Skolem functions. Suppose, if possible, there exist Turing machines $M^1_{halt}$ through $M^k_{halt}$ that take $x \in \mathbb{N}$ as input, and always halt and compute the values of $f_1(x)$ through $f_k(x)$ respectively. Given $x \in \mathbb{N}$, we can then use $M^1_{halt}$ through $M^k_{halt}$ to compute the values of $f_1(x), \dots f_k(x)$, and determine if $P_{halt}(x, f_1(x), \dots f_k(x)) = 0$. If so, we know that $x \in S_{halt}$; otherwise $x \notin S_{halt}$. This gives an algorithm (or halting Turing machine) to determine if any natural number $x \in S_{halt}$ – an impossibility! Hence, there cannot exist Turing machines $M^1_{halt}$ through $M^k_{halt}$ that compute the Skolem functions $f_1(x)$ through $f_k(x)$, even when the domain and interpretation of all predicates and symbols is pre-determined. ◀

## 4 Computable subclasses

In light of the above results, we cannot hope to have generic algorithms that synthesize Skolem functions for first order logic formula unlike for propositional formula. However, it turns out that Skolem functions can indeed be computed for formulas in several interesting first order theories. To do this, we identify the properties that a theory must have in order to allow for algorithms that synthesize Skolem functions effectively.

▶ **Theorem 5.** *Every first order theory that is (i) decidable, (ii) has a recursively enumerable domain, and (iii) has computable interpretations of predicates and functions, admits effective computation of Skolem functions.*

**Proof.** Fix a first order theory $\mathcal{T}$ satisfying the above premises and consider a well-formed formula of the form $\xi \equiv \forall Z \exists y\, \xi(X, Z, y)$ in this theory, where $X$ is a sequence of free variables, and $Z$ is a sequence of universally quantified variables. On Skolemizing, we get $\xi^\star \equiv \forall Z\, \xi(X, Z, f(X, Z))$, where $f$ is a Skolem function of arity $|X| + |Z|$. We can now design a Turing machine (or algorithm) $M$ that takes any $|X| + |Z|$-tuple of elements from $\mathcal{D}$, say $\sigma$, as input and halts after computing $f(\sigma)$. The Turing machine $M$ works as follows:

(a) It first determines if $\exists y\, \xi(\sigma, y)$ holds. Since the theory $\mathcal{T}$ is decidable, this is indeed possible.

(b) If the answer to the above question is "Yes", the machine $M$ recursively enumerates the elements of $\mathcal{D}$, and for each element $n$ thus enumerated, it checks if $\xi(\sigma, n)$ evaluates to true. Once again, decidability of $\mathcal{T}$ ensures that the latter check can be effectively done. The machine $M$ outputs the first (in recursive enumeration order) element $n$ of $\mathcal{D}$ for which $\xi(\sigma, n)$ is true, and halts.

(c) If the answer to the question in (a) is "No", i.e. there is no $n \in \mathcal{D}$ such that $\xi(\sigma, n)$ is true, the Turing machine outputs the first (in recursive enumeration order) element of $\mathcal{D}$ and halts.

It is easy to verify that the function $f$ computed by $M$ satisfies $\forall Z \big(\exists y\, \xi(X, Z, y) \Leftrightarrow \xi(X, Z, f(X, Z))\big)$ for every valuation of the free variables $X$ in $\mathcal{D}^{|X|}$. As we will see shortly, the ability to compute Skolem functions arising from first order sentences of the form $\forall Z \exists y\, \xi(X, Z, y)$ suffices to compute Skolem functions arising from arbitrary first order sentences. Therefore, the above argument shows that Skolem functions can be effectively computed for all sentences in decidable first order theories with recursively enumerable domains. ◀

And from this we can derive the following corollary.

▶ **Corollary 6.** *The Skolem function synthesis problem is effectively computable for following theories:*

1. *the theory of dense linear order without endpoints,*
2. *Presburger arithmetic,*
3. *linear rational arithmetic,*
4. *first order theories with bounded domain (of which the Boolean case is a special case).*

**Proof.** We only show 1. here. The others follow easily from known results. For this, we start by observing that every countable dense linear order without endpoints is isomorphic to $(\mathbb{Q}, <)$. The domain is clearly countable and the interpretations of predicates is computable.

To show decidability, we note that the theory of dense linear order without endpoints is characterized by the following axioms:

- Linear order: $\forall x \forall y \big((x < y) \vee (y < x) \vee (x = y)\big)$
- Non reflexive: $\forall x\, \neg(x < x)$
- Transitivity: $\forall x \forall y \forall z \big((x < y) \wedge (y < z) \Rightarrow (x < z)\big)$
- No high end point: $\forall x \exists y\, (x < y)$
- No low end point: $\forall x \exists y\, (y < x)$
- Dense: $\forall x \forall y \big((x < y) \Rightarrow \exists z\, ((x < z) \wedge (z < y))\big)$

This theory is $\aleph_0$-categorical and therefore by Lós-Vaught test, it is complete. Therefore, for every sentence $\xi$ in the theory, either $\xi$ or $\neg\xi$ (but not both) is a logical consequence of the axioms. Given a sentence $\xi$, we can therefore decide if the axioms entail $\xi$ or $\neg\xi$ (interleave two proofs, one for entailment of $\xi$ and the other for entailment of $\neg\xi$ – one of them must halt and provide a proof). Also, $(\mathbb{Q}, <)$ is the only countable model of the theory. ◀

Whenever Skolem functions are computable, e.g., in all the above theories, one can further ask: *Can Skolem functions be represented as terms in the underlying logical theory?* It is easy to see that a positive answer to this question implies an effective procedure for quantifier elimination. We also know that some theories, viz. (countable) dense linear order without endpoints, do not admit quantifier elimination. Therefore, we obtain:

▶ **Proposition 7.** *There exist first order theories for which Skolem functions can be effectively computed, but are not expressible as terms in the underlying logical theory.*

## 5 Conclusion

The study of algorithmic computation of Skolem functions is highly nuanced. In this note, we observed that for first-order logic it is in fact uncomputable even when all predicates and functions are interpretted. However, our sufficient conditions of computability mean that there are large subclasses, i.e., first order theories where computability itself is not an issue. However, this does not necessarily translate to expressibility of Skolem functions in the underlying logic. Neither does it automatically imply existence of *efficient* algorithms. indeed, even in the Boolean setting several computational hardness results are known even though the problem is easily computable. We leave the characterization of these issues and development of efficient algorithms for Skolem function synthesis for these theories of first order logic as intriguing directions for future research.

## References

1 S. Akshay, J. Arora, S. Chakraborty, S. Krishna, D. Raghunathan, and S. Shah. Knowledge compilation for boolean functional synthesis. In *Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA*, 2019.

2 S. Akshay, S. Chakraborty, S. Goel, S. Kulal, and S. Shah. Boolean functional synthesis: hardness and practical algorithms. *Form Methods Syst Des.*, 2020.

3 S. Akshay, Supratik Chakraborty, Shubham Goel, Sumith Kulal, and Shetal Shah. How hard is boolean functional synthesis. In *In CAV 2018 Proceedings*, 2018.

4 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.

5 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.

6 Martin Davis, Yuri Matijasevic, and Julia Robinson. Hilbert's tenth problem. diophantine equations: positive aspects of a negative solution. In *Proceedings of symposia in pure mathematics*, volume 28, pages 323–378, 1976.

7 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.

8 Priyanka Golia, Subhajit Roy, and Kuldeep S. Meel. Manthan: A data-driven approach for boolean function synthesis. In *Proceedings of International Conference on Computer-Aided Verification (CAV)*, 7 2020.

9 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA, 2006.

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

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

12 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.

13 Markus N. Rabe. Incremental determinization for quantifier elimination and functional synthesis. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II*, pages 84–94, 2019.

14 Markus N. Rabe and Sanjit 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.

15 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.

16 A. Shukla, A. Bierre, M. Siedl, and L. Pulina. A survey on applications of quantified boolean formula. In *Proceedings of the Thirty-First International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 78–84, 2019.

17 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.