A Decidable Fragment of Second Order Logic With Applications to Synthesis

P. Madhusudan
University of Illinois, Urbana Champaign, Urbana, IL, USA
madhu@illinois.edu

Umang Mathur
University of Illinois, Urbana Champaign, Urbana, IL, USA
umathur3@illinois.edu
https://orcid.org/0000-0002-7610-0660

Shambwaditya Saha
University of Illinois, Urbana Champaign, Urbana, IL, USA
ssaha6@illinois.edu

Mahesh Viswanathan
University of Illinois, Urbana Champaign, Urbana, IL, USA
vmahesh@illinois.edu

Abstract

We propose a fragment of many-sorted second order logic called EQSMT and show that checking satisfiability of sentences in this fragment is decidable. EQSMT formulae have an $\exists^*\forall^*$ quantifier prefix (over variables, functions and relations) making EQSMT conducive for modeling synthesis problems. Moreover, EQSMT allows reasoning using a combination of background theories provided that they have a decidable satisfiability problem for the $\exists^*\forall^*$ FO-fragment (e.g., linear arithmetic). Our decision procedure reduces the satisfiability of EQSMT formulae to satisfiability queries of $\exists^*\forall^*$ formulae of each individual background theory, allowing us to use existing efficient SMT solvers supporting $\exists^*\forall^*$ reasoning for these theories; hence our procedure can be seen as *effectively quantified SMT* (EQSMT) reasoning.

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

Keywords and phrases second order logic, synthesis, decidable fragment

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.30

Funding This work has been supported by NSF grants 1422798, 1329991, 1138994 and 1527395.

1 Introduction

The goal of program synthesis is to automatically construct a program that satisfies a given specification. This problem has received a lot of attention from the research community in recent years [34, 4, 14]. Several different approaches have been proposed to address this challenge (see [4, 17] for some of these). One approach to program synthesis is to reduce the problem to the satisfiability problem in a decidable logic by constructing a sentence whose existentially quantified variables identify the program to be synthesized, and the inner formula expresses the requirements that the program needs to meet.

This paper furthers this research program by identifying a decidable *second-order* logic that is suitable for encoding problems in program synthesis. To get useful results, one needs to constrain the semantics of functions and relations used in encoding the synthesis problem. Therefore our logic has a set of *background theories*, where each of the background

27th EACSL Annual Conference on Computer Science Logic (CSL 2018).
Editors: Dan Ghica and Achim Jung; Article No. 30; pp. 30:1–30:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

theories is assumed to be independently axiomatized and equipped with a solver. Finally, to leverage the advances made by logic solvers, our aim is to develop a decision procedure for our logic that makes *black-box* calls to the decision procedures (for $\exists^*\forall^*$ satisfiability) for the background theories.

With the above goal in mind, let us describe our logic. It is a *many-sorted* logic that can be roughly described as an *uninterpreted combination of theories* (UCT) [20]. A UCT has a many-sorted universe where there is a special sort σ_0 that is declared to be a *foreground* sort, while the other sorts $(\sigma_1, \ldots \sigma_n)$ are declared to be background sorts. We assume that there is some fixed signature of functions, relations, and constants over each individual background sort that is purely over that sort. Furthermore, we assume that each background sort σ_i $(i > 0)$ comes with an associated *background theory* T_i; T_i can be arbitrary, even infinite, but is constrained to formulae with functions, relations and constants that only involve the background sort σ_i. Our main contribution is a decidability result for the satisfiability problem modulo these theories for boolean combinations of sentences of the form

$$(\exists \mathbf{x})(\exists \boldsymbol{\mathcal{R}})(\exists \mathbf{F})(\forall \mathbf{y})(\forall \boldsymbol{\mathcal{P}})(\forall \mathbf{G})\psi, \tag{1}$$

- $\mathbf{x}$ is a set of existentially quantified first order variables. These variables can admit values in any of the sorts (background or foreground);
- $\boldsymbol{\mathcal{R}}$ is a set of existentially quantified relational variables, whose arguments are restricted to be over the foreground sort σ_0;
- $\mathbf{F}$ is a set of existentially quantified function variables, which take as arguments elements from the foreground sort σ_0, and return a value in any of the background sorts σ_i;
- $\mathbf{y}$ is a set of universally quantified first order variables over any of the sorts;
- $\boldsymbol{\mathcal{P}}$ is a set of universally quantified relational variables, whose arguments could be of any of the sorts; and
- $\mathbf{G}$ is a set of universally quantified function variables, whose arguments can be from any sort and could return values of any sort.

Thus our logic has sentences with prefix $\exists^*\forall^*$, allowing for quantification over both first order variables and second-order variables (relational and functional). To obtain decidability, we have carefully restricted the sorts (or *types*) of second-order variables that are existentially and universally quantified, as described above.

Our decidability result proceeds as follows. By crucially exploiting the disjointness of the universes of background theories and through a series of transformations, we reduce the satisfiability problem for our logic to the satisfiability of several pure $\exists^*\forall^*$ *first-order* logic formulas over the individual background theories $T_1, \ldots T_n$. Consequently, if the background theories admit (individually) a decidable satisfiability problem for the first-order $\exists^*\forall^*$ fragment, then satisfiability for our logic is *decidable*. Examples of such background theories include Presburger arithmetic, the theory of real-closed fields, and the theory of linear real arithmetic. Our algorithm for satisfiability makes finitely many black-box calls to the engines for the individual background theories.

Salient aspects of our logic and our decidability result

Design for decidability: Our logic is defined to carefully avoid the undecidability that looms in any logic of such power. We do not know of any decidable second-order logic fragment that supports background theories such as arithmetic and uninterpreted functions. While *quantifier-free* decidable logics can be combined to get decidable logics using Nelson-Oppen combinations [24], or local theory extensions [33], combining quantified logics is notoriously hard, and there are only few restricted classes of first-order logic that are known to be decidable.

Our design choice forces *communication* between theories using the foreground sort, keeping the universes of the different sorts *disjoint*, which allows a decidable combination of $\exists^*\forall^*$ theories. We emphasize that, unlike existing work on quantified first-order theories that are decidable by reduction to quantifier-free SMT, our logic allows existential and universal quantification over the background theories as well, and the decision procedure reduces satisfiability to $\exists^*\forall^*$ fragment of the underlying theories. Our result can hence be seen as a decidable combination of $\exists^*\forall^*$ theories that further supports second-order quantification.

Undecidable Extensions: We show that our logic is on the edge of the decidability barrier, by showing that lifting some of the restrictions we have will render the logic undecidable. In particular, we show that if we allow outer existential quantification over functions (which is related to the condition demanding that all function variables are universally quantified in the inner block of quantifiers), then satisfiability of the logic is undecidable. Second, if we lift the restriction that the underlying background sorts are pairwise disjoint, then again the logic becomes undecidable. The design choices that we have made hence seem crucial for decidability.

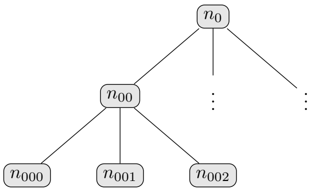
Expressing Synthesis Problems: Apart from decidability, a primary motivational design principle of our logic is to express *synthesis* problems. Synthesis problems typically can be expressed in $\exists^*\forall^*$ fragments, where we ask whether there exists an object of the kind we wish to synthesize (using the block of existential quantifiers) such that the object satisfies certain properties (expressed by a universally quantified formula). For instance, if we are synthesizing a program snippet that is required to satisfy a Hoare triple (pre/post condition), we can encode this by asking whether there is a program snippet such that for all values of variables (modeling the input to the snippet), the verification condition corresponding to the Hoare triple holds. In this context, the existentially quantified variables (first order and second order) can be used to model program snippets. Furthermore, since our logic allows second-order universal quantification over *functions*, we can model aspects of the program state that require *uninterpreted functions*, in particular pointer fields that model the heap.

Evaluation on Synthesis Problems: We illustrate the applicability of our logic for two classes of synthesis problems. The first class involves synthesizing *recursive programs* that work over inductive data-structures. Given the precise pre/post condition for the program to be synthesized, we show how to model recursive program synthesis by synthesizing only a straight-line program (by having the output of recursive calls provided as inputs to the straightline program). The verification condition of the program requires universal quantification over both scalar variables as well as heap pointers, modeled as uninterpreted functions. Since such verification-conditions are already very expressive (even for the purpose of verification), we adapt a technique in the literature called *natural proofs* [20, 29, 26], that soundly abstracts the verification condition to a decidable theory. This formulation still has universal quantification over variables and functions, and combines standard background theories such as arithmetic and theory of uninterpreted functions. We then show that synthesis of bounded-sized programs (possibly involving integer constants that can be unbounded) can be modeled in our logic. In this modeling, the universal quantification over functions plays a crucial role in modeling the pointers in heaps, and modeling uninterpreted relations that capture inductive data-structure predicates (such as `lseg`, `bstree`, etc.).

The second class of synthesis involves taking a recursive definition of a function, and synthesizing a non-recursive (and iteration free) function equivalent to it. In our modeling,

```
term := c | (x + c) | ite(pred, term, term)
pred := (term < 0) | (term = 0) | (term > 0)
```

(a) Grammar for M_{three}. `ite(·, ·, ·)` stands for `if-then-else`.



(b) Program skeleton

Figure 1 Synthesizing M_{three} using EQSMT.

the existential quantification over the foreground sort as well as the background sort of integers is utilized, as the synthesized function involves integers.

The crux of our contribution, therefore, is providing a decidable logic that can express synthesis problems succinctly. Such a logic promises to provide a useful interface between researchers working on practical synthesis applications and researchers working on engineering efficient tools for solving them, similar to the role SMT plays in verification.

2 Motivating EQSMT for synthesis applications

In program synthesis, the goal is to search for programs, typically of bounded size, that satisfy a given specification. The $\exists$-Block of an EQSMT formula can be used to express the search for the syntactic program. The inner formula, then, must interpret the semantics of this syntactic program, and express that it satisfies the specification. If the specification is a universally quantified formula, then, we can encode the synthesis problem in EQSMT.

One of the salient features of the fragment EQSMT is the ability to universally quantify over functions and relations. Often, specifications for programs, such as those that manipulate heaps, involve a universal quantification over uninterpreted functions (that model pointers). EQSMT aptly provides this functionality, while still remaining within the boundaries of decidability. Further, EQSMT supports combination of background theories/sorts; existential quantification over these sorts can thus be used to search for programs with arbitrary elements from these background sorts. As a result, the class of target programs that can be expressed by an EQSMT formula is infinite. Consequently, when our decision procedure returns unsatisfiable, we are assured that no program (from an infinite class of programs) exists, (most CEGIS solvers for program synthesis cannot provide such a guarantee.)

We now proceed to give a concrete example of a synthesis problem which will demonstrate the power of EQSMT. Consider the specification of the following function M_{three}, which is a slight variant of the classical McCarthy's 91 function [23], whose specification is given below.

$$M_{\mathsf{three}}(n) = \begin{cases} n - 30 & \text{if } n > 13 \\ M_{\mathsf{three}}(M_{\mathsf{three}}(M_{\mathsf{three}}(n + 61))) & \text{otherwise} \end{cases} \tag{2}$$

We are interested in synthesizing a straight line program that implements the recursive function M_{three}, and can be expressed as a `term` over the grammar specified in Figure 1(a).

Here, we only briefly discuss how to encode this synthesis problem in EQSMT, and the complete details can be found in Appendix A. First, let us fix the maximum height of the term we are looking for, say to be 2. Then, the program we want to synthesize can be represented as a tree of height at most 2 such that every node in the tree can have ≤ 3 child nodes (because the maximum arity of any function in the above grammar is 3, corresponding to `ite`). The skeleton of such an expression tree is shown in Figure 1(b). Every node in the tree is named according to its path from the root node.

The synthesis problem can then be encoded as the following formula

$$\begin{aligned}\phi_{M_{\mathsf{three}}} \equiv\, & (\exists n_0, n_{00}, n_{01}, \dots n_{022} : \sigma_0)\ (\exists \mathsf{Left}, \mathsf{Mid}, \mathsf{Right} : \sigma_0\sigma_0)\\ & (\exists \mathtt{ADD}, \mathtt{ITE}, \mathtt{LTZero}, \mathtt{EQZero}, \mathtt{GTZero}, \mathtt{INPUT}, C_1, C_2, C_3 : \sigma_{\mathsf{label}})\\ & (\exists c_1, c_2, c_3 : \mathbb{N})\ (\exists f_{\mathsf{label}} : \sigma_0, \sigma_{\mathsf{label}})\\ & \quad \varphi_{\mathsf{well\text{-}formed}} \wedge (\forall x : \mathbb{N})(\forall g_{\mathsf{val}} : \sigma_0, \mathbb{N})\ (\varphi_{\mathsf{semantics}} \implies \varphi_{\mathsf{spec}})\end{aligned}$$

Here, the nodes $n_0, n_{00}, \dots$ are elements of the foreground sort σ_0. The binary relations Left, Mid, Right over the foreground sort will be used to assert that a node n is the left,middle, right child respectively of node n' : $\mathsf{Left}(n', n)$, $\mathsf{Mid}(n', n)$, $\mathsf{Right}(n', n)$. The operators or *labels* for nodes belong to the background sort σ_{label}, and can be one of `ADD` $(+)$, `ITE` (`ite`), `LTZero` (< 0), `GTZero` (> 0), (`EQZero` $(= 0)$), `INPUT` (denoting the input to our program), or constants C_1, C_2, C_3 (for which we will synthesize natural constants c_1, c_2, c_3 in the (infinite) background sort $\mathbb{N}$). The function f_{label} assigns a label to every node in the program, and the formula $\varphi_{\mathsf{well\text{-}formed}}$ asserts some sanity conditions:

$$\begin{aligned}\varphi_{\mathsf{well\text{-}formed}} \equiv & \bigwedge_{\rho \neq \rho'} n_\rho \neq n_{\rho'} \ \wedge \mathsf{Left}(n_0, n_{00}) \wedge \bigwedge_{\rho \neq 00} \neg(\mathsf{Left}(n_0, n_\rho))) \wedge \cdots\\ & \wedge \neg(\mathtt{ADD} = \mathtt{ITE}) \wedge \neg(\mathtt{ADD} = \mathtt{LTZero}) \wedge \cdots \wedge \neg(C_1 = C_3) \wedge \neg(C_2 = C_3)\\ & \wedge \bigwedge_{\rho} (f_{\mathsf{label}}(n_\rho){=}\mathtt{ADD}) \vee (f_{\mathsf{label}}(n_\rho){=}\mathtt{ITE}) \vee \cdots \vee (f_{\mathsf{label}}(n_\rho){=}C_3)\end{aligned}$$

The formula $\varphi_{\mathsf{semantics}}$ asserts that the 'meaning' of the program can be inferred from the meaning of the components of the program. We will use the function g_{val}, that assigns value to nodes from $\mathbb{N}$, for this purpose :

$$\varphi_{\mathsf{semantics}} \equiv \bigwedge_{\rho, \rho_1, \rho_2} \left[\begin{array}{l}\Big(f_{\mathsf{label}}(n_\rho) = \mathtt{ADD} \wedge \mathsf{Left}(n_\rho, n_{\rho_1}) \wedge \mathsf{Mid}(n_\rho, n_{\rho_2})\Big)\\ \qquad \implies g_{\mathsf{val}}(n_\rho) = g_{\mathsf{val}}(n_{\rho_1}) + g_{\mathsf{val}}(n_{\rho_2})\Big)\\ \qquad\qquad\qquad \vdots\\ \qquad \wedge\ f_{\mathsf{label}}(n_\rho) = C_3 \implies g_{\mathsf{val}}(n_\rho) = c_3\end{array}\right]$$

Finally, the formula φ_{spec} expresses the specification of the program as in Equation (2). A complete description is provided in Appendix A.

Observe that the formula $\phi_{M_{\mathsf{three}}}$ has existential and universal quantification over functions and relations, as allowed by our decidable fragment EQSMT. The existentially quantified functions map the foreground sort σ_0 to one of the background sorts, and the existentially quantified relations span only over the foreground sort.

We encoded the above EQSMT formula in the z3 [12] SMT solver (see Section 6 for details), which synthesized the expression $\mathtt{fun(n)} = \mathtt{ite(n} > \mathtt{13}, \mathtt{n} - \mathtt{30}, -\mathtt{16)}$. In Section 6, we show that we can synthesize a large class of such programs amongst others.

3 Many-sorted Second Order Logic and the EQSMT Fragment

We briefly recall the syntax and semantics of general many-sorted second order logic, and then present the EQSMT fragment of second order logic.

Many-sorted second-order logic

A many-sorted signature is a tuple $\Sigma = (\mathcal{S}, \mathcal{F}, \mathfrak{R}, \mathcal{V}, \mathcal{V}^{\mathsf{fun}}, \mathcal{V}^{\mathsf{rel}})$ where, $\mathcal{S}$ is a nonempty finite set of sorts, $\mathcal{F}$, $\mathfrak{R}$, $\mathcal{V}$, $\mathcal{V}^{\mathsf{fun}}$, $\mathcal{V}^{\mathsf{rel}}$ are, respectively, sets of function symbols, relation symbols, first order variables, function variables and relation variables. Each variable $x \in \mathcal{V}$ is associated with a sort $\sigma \in \mathcal{S}$, represented as $x : \sigma$. Each function symbol or function variable also has an associated type $(w, \sigma) \in \mathcal{S}^* \times \mathcal{S}$, and each relation symbol and relation variable has a type $w \in \mathcal{S}^+$. We assume that the set of symbols in $\mathcal{F}$ and $\mathfrak{R}$ are either finite or countably infinite, and that $\mathcal{V}$, $\mathcal{V}^{\mathsf{fun}}$, and $\mathcal{V}^{\mathsf{rel}}$ are all countably infinite. Constants are modeled using 0-ary functions. We say that Σ is *unsorted* if $\mathcal{S}$ consists of a single sort.

Terms over a many-sorted signature Σ have an associated sort and are inductively defined by the grammar

$$t{:}\sigma \quad := \quad x{:}\sigma \quad | \quad f(t_1{:}\sigma_1, t_2{:}\sigma_2, \ldots, t_m{:}\sigma_m) \quad | \quad F(t_1{:}\sigma_1, t_2{:}\sigma_2, \ldots, t_n{:}\sigma_n)$$

where $f : (\sigma_1\sigma_2 \cdots \sigma_m, \sigma) \in \mathcal{F}$, and $F : (\sigma_1\sigma_2 \cdots \sigma_n, \sigma) \in \mathcal{V}^{\mathsf{fun}}$. Formulae over Σ are inductively defined as

$$\begin{aligned}\phi \quad := \quad & \bot \quad | \quad \phi \Rightarrow \phi \quad | \quad t{:}\sigma = t'{:}\sigma \quad | \quad R(t_1{:}\sigma_1, t_2{:}\sigma_2, \ldots, t_m{:}\sigma_m) \quad | \\ & \mathcal{R}(t_1{:}\sigma_1, t_2{:}\sigma_2, \ldots, t_n{:}\sigma_n) \quad | \quad (\exists x{:}\sigma)\,\phi \quad | \quad (\exists F{:}w, \sigma)\,\phi \quad | \quad (\exists \mathcal{R}'{:}w)\,\phi\end{aligned}$$

where $R{:}(\sigma_1\sigma_2 \cdots \sigma_m) \in \mathfrak{R}$, $\mathcal{R}, \mathcal{R}'$ are relation variables, F is a function variable, of appropriate types. Note that equality is allowed only for terms of same sort. A formula is said to be *first-order* if it does not use any function or relation variables.

The semantics of many sorted logics are described using many-sorted structures. A Σ-structure is a tuple $\mathcal{M} = (\mathcal{U}, \mathcal{I})$ where $\mathcal{U} = \{M_\sigma\}_{\sigma \in \mathcal{S}}$ is a collection of pairwise disjoint $\mathcal{S}$ indexed universes, and $\mathcal{I}$ is an interpretation function that maps each each variable $x : \sigma$ to an element in the universe M_σ, each function symbol and each function variable to a function of the appropriate type on the underlying universe. Similarly, relation symbols and relation variables are also assigned relations of the appropriate type on the underlying universe. For an interpretation $\mathcal{I}$, as is standard, we use $\mathcal{I}[c^x/x]$ to denote the interpretation that maps x to c^x, and is otherwise identical to $\mathcal{I}$. For function variable F and relation variable $\mathcal{R}$, $\mathcal{I}[f^F/F]$ and $\mathcal{I}[R^{\mathcal{R}}/\mathcal{R}]$ are defined analogously.

Interpretation of terms in a model is the usual one obtained by interpreting variables, functions, and function variables using their underlying interpretation in the model; we skip the details. The satisfaction relation $\mathcal{M} \models \phi$ is also defined in the usual sense, and we will skip the details.

A first-order theory is a tuple $T = (\Sigma_T, \mathcal{A}_T)$, where $\mathcal{A}_T$ is a set of (possibly infinite) first-order sentences. Theory T is *complete* if every sentence α or its negation is entailed by $\mathcal{A}_T$, i.e., either every model satisfying $\mathcal{A}_T$ satisfies α, or every model satisfying $\mathcal{A}_T$ satisfies $\neg\alpha$. A theory $\mathcal{A}_T$ is *consistent* if it is not the case that there is a sentence α such that both α and $\neg\alpha$ are entailed.

The logic EQSMT

We now describe EQSMT, the fragment of many-sorted second order logic that we prove decidable in this paper and that we show can model synthesis problems.

Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathfrak{R}, \mathcal{V}, \mathcal{V}^{\mathsf{fun}}, \mathcal{V}^{\mathsf{rel}})$ be a many sorted signature. Σ is a *pure signature* if (a) the type of every function symbol and every relation symbol is over a single sort (however, function variables and relation variables are allowed to mix sorts), (b) there is a special sort σ_0 (which we call the *foreground sort*, while other sorts $\sigma_1, \ldots, \sigma_k$ are called *background sorts*) and (c) there are no function or relation symbols involving σ_0.

The fragment EQSMT is the set of sentences defined over a pure signature Σ, with foreground sort σ_0 and background sorts $\sigma_1, \ldots \sigma_k$, by the following grammar

$$\phi \quad := \quad \varphi \quad | \quad \exists (x : \sigma)\, \phi \quad | \quad (\exists \mathcal{R} : w)\, \phi \quad | \quad (\exists F : w, \sigma_i)\, \phi$$

where, $\sigma \in \mathcal{S}$, $w \in \sigma_0^+$ (i.e., only foreground sort), $1 \leq i \leq k$, and φ is a universally quantified formula defined by the grammar

$$\varphi \quad := \quad \psi \quad | \quad \forall (y : \sigma)\, \varphi \quad | \quad (\forall \mathcal{R} : w')\, \varphi \quad | \quad (\forall F : w', \sigma)\, \varphi$$

where, $\sigma \in \mathcal{S}$, $w' \in \mathcal{S}^+$, and ψ is quantifier free over Σ.

The formulas above consist of an existential quantification block followed by a universal quantification block. The existential block can have first-order variables of any sort, relation variables that are over the foreground sort only, and function variables that map tuples of the foreground sort to a background sort. The inner universal block allows all forms of quantification — first-order variables, function variables, and relation variables of all possible types. The inner formula is quantifier-free. We will retrict our attention to *sentences* in this logic, i.e., we will assume that all variables (first-order/function/relation) are quantified. We will denote by $\mathbf{x}_i$ (resp. $\mathbf{y}_i$), the set of existentially (resp. universally) quantified first order variables of sort σ_i, for every $0 \leq i \leq k$.

The problem.

The problem we consider is that of deciding satisfiability of EQSMT sentences with *background theories* for the background sorts. First we introduce some concepts.

An *uninterpreted combination of theories (UCT)* over a pure signature, with $\{\sigma_0, \sigma_1, \ldots, \sigma_k\}$ as the set of sorts, is the union of theories $\{T_{\sigma_i}\}_{1 \leq i \leq k}$, where each T_{σ_i} is a theory over signature σ_i. A sentence ϕ is $\bigcup_{i=1}^{k} T_{\sigma_i}$-satisfiable if there is a multi-sorted structure $\mathcal{M}$ that satisfies ϕ and all the sentences in $\bigcup_{i=1}^{k} T_{\sigma_i}$.

The satisfiability problem for EQSMT with background theories is the following. Given a UCT $\{T_{\sigma_i}\}_{1 \leq i \leq k}$ and a sentence $\phi \in$ EQSMT , determine if ϕ is $\bigcup_{i=1}^{k} T_{\sigma_i}$-satisfiable. We show that this is a decidable problem, and furthermore, there is a decision procedure that uses a finite number of black-box calls to satisfiability solvers of the underlying theories to check satisfiability of EQSMT sentences.

For the rest of this paper, for technical convenience, we will assume that the boolean theory T_{bool} is one of the background theories. This means $\mathsf{bool} \in \mathcal{S}$ and the constants $\top : \mathsf{bool}, \bot : \mathsf{bool} \in \mathcal{F}$. The set of sentences in T_{bool} is $\mathcal{A}_{\mathsf{bool}} = \{\top \neq \bot, \forall (y : \mathsf{bool}) \cdot (y = \top \vee y = \bot)\}$. Note that checking satisfiability of a $\exists^*\forall^*$ sentence over T_{bool} is decidable.

4 The Decision Procedure for EQSMT

In this section we present our decidability result for sentences over EQSMT in presence of background theories. Let us first state the main result of this paper.

▶ **Theorem 1.** *Let Σ be a pure signature with foreground sort σ_0 and background sorts $\sigma_1, \ldots, \sigma_k$. Let $\{T_{\sigma_i}\}_{1 \leq i \leq k}$ be a UCT such that, for each i, checking T_{σ_i}-satisfiability of $\exists^*\forall^*$ first-order sentences is decidable. Then the problem of checking $\bigcup\limits_{i=1}^{k} T_{\sigma_i}$-satisfiability of EQSMT sentences is decidable.* ◀

We will prove the above theorem by showing that any given EQSMT sentence ϕ over a UCT signature Σ can be transformed, using a sequence of satisfiability preserving transformation steps, to the satisfiability of $\exists\forall$ first-order formulae over the individual theories.

We give a brief overview of the sequence of transformations (Steps 1 through 4). In Step 1, we replace the occurrence of every relation variable $\mathcal{R}$ (quantified universally or existentially) of sort w by a function variable F of sort (w, bool). Note that doing this for the outer existentially quantified relation variables keeps us within the syntactic fragment. In Step 2, we eliminate function variables that are existentially quantified. This crucially relies on the *small model property* for the foreground universe, similar to EPR [5]. This process, however, adds both existential first-order variables and universally quantified function variables. In Step 3, we eliminate the universally quantified function variables using a standard Ackermann reduction [28], which adds more universally quantified first-order variables.

The above steps result in a first-order $\exists^*\forall^*$ sentence over the combined background theories, and the empty theory for the foreground sort. In Step 4, we show that the satisfiability of such a formula can be reduced to a finite number of satisfiability queries of $\exists^*\forall^*$ sentences over *individual* theories.

Step 1 : Eliminating relation variables

The idea here is to introduce, for every relational variable $\mathcal{R}$ (with type w), a function variable $f^{\mathcal{R}}$ (with type $(w, \sigma_{\mathsf{bool}})$) that corresponds to the characteristic function of $\mathcal{R}$.

Let ϕ be EQSMT formula over Σ. We will transform ϕ to an EQSMT formula $\phi^{\mathsf{Step\text{-}1}}$ over the same signature Σ. Every occurrence of an atom of the form $\mathcal{R}(t_1\!:\!\sigma_{i_1}, \dots, t_k\!:\!\sigma_{i_k})$ in ϕ, is replaced by $f^{\mathcal{R}}(t_1\!:\!\sigma_{i_1}, \dots, t_k\!:\!\sigma_{i_k}) = \top$ in $\phi^{\mathsf{Step\text{-}1}}$. Further, every quantification $Q(\mathcal{R} : w)$ is replaced by $Q(f^{\mathcal{R}} : w, \mathsf{bool})$, where $Q \in \{\forall, \exists\}$. Thus, the resultant formula $\phi^{\mathsf{Step\text{-}1}}$ has no relation variables. Further, it is a EQSMT formula, since the types of the newly introduced existentially quantified function variables are of the form $(\sigma_0^+, \sigma_{\mathsf{bool}})$. The correctness of the above transformation is captured by the following lemma.

▶ **Lemma 2.** *ϕ is $\bigcup\limits_{i=1}^{k} T_{\sigma_i}$-satisfiable iff $\phi^{\mathsf{Step\text{-}1}}$ is $\bigcup\limits_{i=1}^{k} T_{\sigma_i}$-satisfiable.*

Step 2: Eliminating existentially quantified function variables

We first note a small-model property with respect to the foreground sort for EQSMT sentences. This property crucially relies on the fact that existentially quantified function variables do not have their ranges over the foreground sort.

▶ **Lemma 3** (Small-model property for σ_0). *Let ϕ be an EQSMT sentence with foreground sort σ_0 and background sorts $\sigma_1, \dots \sigma_k$. Let n be the number of existentially quantified first-order variables of sort σ_0 in ϕ. Then, ϕ is $\cup_{i=1}^{k} T_{\sigma_i}$-satisfiable iff there is a structure $\mathcal{M} = (\{M_{\sigma_i}\}_{i=0}^{k}, \mathcal{I})$, such that $|M_{\sigma_0}| \leq n$, $\mathcal{M} \models \cup_{i=1}^{k} T_{\sigma_i}$ and $\mathcal{M} \models \phi$.*

Proof (Sketch). We present the more interesting direction here. Consider a model $\mathcal{M} = (\mathcal{U}, \mathcal{I})$ such that $\mathcal{M} \models \bigcup\limits_{i=1}^{k} T_{\sigma_i}$ and $\mathcal{M} \models \phi$. Let $\mathcal{I}_\exists$ be the interpretation function that *extends* $\mathcal{I}$ so that $(\mathcal{U}, \mathcal{I}_\exists) \models \varphi$, where φ is the inner universally quantified subformula of ϕ. Let $U = \{\mathcal{I}_\exists(x) \in M_{\sigma_0} \,|\, x \in \mathbf{x}_0\}$ be the restriction of the foreground universe to the interpretations of the variables $\mathbf{x}_0$. Clearly, $|U| \leq |\mathbf{x}_0|$.

Let us first show that $(\mathcal{U}|_U, \mathcal{I}_\exists|_U) \models \varphi$. For this, first see that for every extension $\mathcal{I}_{\exists\forall}$ of $\mathcal{I}_\exists$ with interpretations of all the universal FO variables, we must have have $(\mathcal{U}, \mathcal{I}_{\exists\forall}) \models \psi$, where ψ is the quantifier free part of φ (and thus also of ϕ). Now, clearly $(\mathcal{U}, \mathcal{I}_{\exists\forall}) \models \psi$ must also hold for those extensions $\mathcal{I}_{\exists\forall}^{U}$ which map all universal variables in $\mathbf{y}_0$ to the set U and maps all universally quantified function variables of range sort σ_0 to function interpretations whose ranges are limited to the set U.

Thus, it must also be the case that when we restrict the universe M_{σ_0} to the set U, we have that $(\mathcal{U}|_U, \mathcal{I}_\exists|_U) \models \forall * \psi$. This is because every universal extension $\mathcal{I}'$ of $\mathcal{I}_\exists|_U$ is also a projection of one of these $\mathcal{I}_{\exists\forall}^{U}$ interpretations. ◀

The proof of the above statement shows that if there is a model that satisfies ϕ (in Lemma 3), then there is a model that satisfies ϕ and in which the foreground universe contains *only* elements that are interpretations of the first-order variables $\mathbf{x}_0$ over the foreground sort (and hence bounded). Consequently, instead of existentially quantifying over a function F (of arity r) from the foreground sort σ_0 to some background sort σ_i, we can instead quantify over first-order variables $\mathbf{x}^F$ of sort σ_i that capture the image of these functions for each r-ary combination of $\mathbf{x}_0$.

Let $\phi^{\textsf{Step-1}}$ be the EQSMT sentence over Σ obtained after eliminating relation variables. Let $\psi^{\textsf{Step-1}}$ be the quantifier free part (also known as the *matrix*) of $\phi^{\textsf{Step-1}}$. Let $\mathcal{T}_0$ be the set of all sub-terms of sort σ_0 that occur in $\psi^{\textsf{Step-1}}$. Note that, $\mathcal{T}_0$ also includes the set of variables and $\mathbf{x}_0$ and $\mathbf{y}_0$. Now, define[††]

$$\widetilde{\psi} \equiv \psi_{\textsf{restrict}} \wedge \psi^{\textsf{Step-1}}, \text{ where, } \psi_{\textsf{restrict}} \equiv \bigwedge_{t\in\mathcal{T}_0} \Big(\bigvee_{x\in\mathbf{x}_0} t = x \Big).$$

Let $\widetilde{\phi}$ the sentence obtained by replacing the matrix $\psi^{\textsf{Step-1}}$ in $\phi^{\textsf{Step-1}}$, by $\widetilde{\psi}$. Then, the correctness of this transformation is noted below.

▶ **Lemma 4.** *$\phi^{\textsf{Step-1}}$ is $\bigcup_{i=1}^{k} T_{\sigma_i}$-satisfiable iff $\widetilde{\phi}$ is $\bigcup_{i=1}^{k} T_{\sigma_i}$-satisfiable.*

We now eliminate the existentially quantified function variables in $\widetilde{\phi}$, one by one. Let $\widetilde{\phi} = (\exists F : \sigma_0^m, \sigma)\exists^*\forall^*\,\widetilde{\psi}$, where σ is a background sort. For every m-tuple $t = (t[1], \dots, t[m])$ over the set $\mathbf{x}_0$, we introduce a variable x_t^F of sort σ. Let $\mathbf{x}^F$ be the set of all such n^m variables, where $n = |\mathbf{x}_0|$ is the number of existential first order variables of sort σ_0 in $\widetilde{\phi}$. Next, we introduce a fresh function variable G^F of sort σ_0^m, σ, and quantify it universally. G^F will be used to *emulate* the function F. Let us define

$$\psi^{\textsf{Step-2}} \equiv (\forall G^F : \sigma_0^m, \sigma)\big(\psi_{\textsf{emulate}} \implies \bar{\psi}\big)$$

where, $\psi_{\textsf{emulate}} \equiv \bigwedge_{t\in\mathbf{x}_0^m} \big(G^F(t[1], \dots, t[m]) = x_t^F\big)$ and $\bar{\psi}$ is obtained by replacing all occurrences of F in $\widetilde{\psi}$ by G^F. Now define $\phi^{\textsf{Step-2}}$ to be the sentence

$$\phi^{\textsf{Step-2}} \equiv (\exists \mathbf{x}^F : \sigma)\exists^*\forall^*(\forall G^F : \sigma_0^m, \sigma)\,\psi^{\textsf{Step-2}}.$$

The following lemma states the correctness guarantee of this transformation.

▶ **Lemma 5.** *$\phi^{\textsf{Step-2}}$ is $\bigcup_{i=1}^{k} T_{\sigma_i}$-satisfiable iff $\phi^{\textsf{Step-1}}$ is $\bigcup_{i=1}^{k} T_{\sigma_i}$-satisfiable.*

Step 3 : Eliminating universal function variables

The recipe here is to perform Ackermann reduction [2] for every universally quantified function variable.

Let $\phi^{\textsf{Step-2}} \equiv \exists^*\forall^*(\forall F : w, \sigma)\,\psi^{\textsf{Step-2}}$, where $\psi^{\textsf{Step-2}}$ is the quantifier free part of $\phi^{\textsf{Step-2}}$, and let $|w| = m$. For every term t of the form $F(t_1, \dots, t_m)$ in $\psi^{\textsf{Step-2}}$, we introduce a fresh first order variable $y^F_{(t_1,t_2,\dots,t_m)}$ of sort σ, and replace every occurrence of the term t in $\psi^{\textsf{Step-2}}$ with $y^F_{(t_1,t_2,\dots,t_m)}$. Let $\widehat{\psi}$ be the resulting quantifier free formula. Let $\mathbf{y}^F$ be the collection of all the newly introduced variables. Let us now define $\psi^{\textsf{Step-3}} \equiv \big(\psi_{\textsf{ack}} \implies \widehat{\psi}\big)$. Here, $\psi_{\textsf{ack}} \equiv$

[††] This is a correction from the previous version of the paper [21].

$\bigwedge\limits_{y_t^F, y_{t'}^F \in \mathbf{y}^F} \Big[(\bigwedge\limits_{j=1}^{m} t_j = t'_j) \implies (y_t^F = y_{t'}^F)\Big]$ where, $t = F(t_1, \ldots t_m), t' = F(t'_1, \ldots, t'_m)$.
Then, the transformed formula $\phi^{\textsf{Step-3}} \equiv \exists^*\forall^*(\forall \mathbf{y}^F : \sigma)\, \psi^{\textsf{Step-3}}$ is correct:

▶ **Lemma 6.** *$\phi^{\textsf{Step-2}}$ is $\bigcup\limits_{i=1}^{k} T_{\sigma_i}$-satisfiable iff $\phi^{\textsf{Step-3}}$ is $\bigcup\limits_{i=1}^{k} T_{\sigma_i}$-satisfiable.*

Step-4: Decomposition and black box calls to $\exists^*\forall^*$ Theory solvers

The EQSMT sentence $\phi^{\textsf{Step-3}}$ obtained after the sequence of steps 1 through 3 is a first order $\exists^*\forall^*$ sentence over Σ. This sentence, however, may possibly contain occurrences of variables of the foreground sort σ_0. Intuitively, the objective of this step is to decompose $\phi^{\textsf{Step-3}}$ into $\exists^*\forall^*$ sentences, one for each sort, and then use decision procedures for the respective theories to decide satisfiability of the decomposed (single sorted) sentences. Since such a decomposition can result into $\exists^*\forall^*$ sentences over the foreground sort, we must ensure that there is indeed a decision procedure to achieve this. For this purpose, let us define T_{σ_0} be the empty theory (that is $\mathcal{A}_\sigma = \varnothing$). Checking satisfiability of $\exists^*\forall^*$ sentences over T_{σ_0} is decidable. Also, satisfiability is preserved in the presence of T_{σ_0} in the following sense.

▶ **Lemma 7.** *$\phi^{\textsf{Step-3}}$ is $\bigcup\limits_{i=1}^{k} T_{\sigma_i}$-satisfiable iff $\phi^{\textsf{Step-3}}$ is $\bigcup\limits_{i=0}^{k} T_{\sigma_i}$-satisfiable.*

We first transform the quantifier free part $\psi^{\textsf{Step-3}}$ of $\phi^{\textsf{Step-3}}$ into an equivalent CNF formula $\psi_{\textsf{CNF}}$. Let $\phi_{\textsf{CNF}}$ be obtained by replacing $\psi^{\textsf{Step-3}}$ by $\psi_{\textsf{CNF}}$. Let $\phi_{\textsf{CNF}} \equiv \exists^*\forall^*\psi_{\textsf{CNF}}$, where $\psi_{\textsf{CNF}} \equiv \bigwedge\limits_{i=1}^{r} \psi_i$ and each ψ_i is a disjunction of atoms. Since $\phi_{\textsf{CNF}}$ is a first order formula over a pure signature, all atoms are either of the form $R(\cdots)$ or $t = t'$ (with possibly a leading negation). Now, equality atoms are restricted to terms of the same sort. Also since Σ is pure, the argument terms of all relation applications have the same sort. This means, for every atom α, there is a unique associated sort $\sigma \in \mathcal{S}$, which we will denote by $\textsf{sort}(\alpha)$.

For a clause ψ_i in $\psi_{\textsf{CNF}}$, let $\textsf{atoms}(\psi_i)$ be the set of atoms in ψ_i. Let $\textsf{atoms}^\sigma(\psi_i) = \{\alpha \in \textsf{atoms}(\psi_i) \,|\, \textsf{sort}(\alpha) = \sigma\}$, and let $\psi_i^\sigma \equiv \bigvee\limits_{\alpha \in \textsf{atoms}^\sigma(\psi_i)} \alpha$. Then, we have the identity $\psi_{\textsf{CNF}} \equiv \bigwedge\limits_{j=1}^{r} \bigvee\limits_{\sigma \in \mathcal{S}} \psi_j^\sigma$. We now state our decomposition lemma.

▶ **Lemma 8.** *$\phi_{\textsf{CNF}}$ is $\bigcup\limits_{i=0}^{k} T_{\sigma_i}$-satisfiable iff there is a mapping $L : \{1, \ldots, r\} \to \mathcal{S}$ such that for each $0 \leq i \leq k$, the formula $\phi_i^L \equiv (\exists \mathbf{x}_i : \sigma_i)(\forall \mathbf{y}_i : \sigma_i) \bigwedge\limits_{j \in L^{-1}(\sigma_i)} \psi_j^{\sigma_i}$ is T_{σ_i}-satisfiable.*

Proof (Sketch). We present the more interesting direction here. Let $\phi_{\textsf{Skolem}}$ be an equi-satisfiable Skolem norm form of $\phi_{\textsf{CNF}}$. That is, $\phi_{\textsf{Skolem}} = \forall^*\psi_{\textsf{Skolem}}$, where $\psi_{\textsf{Skolem}}$ is obtained from $\psi_{\textsf{CNF}}$ by replacing all existential variables $\mathbf{x}_0, \mathbf{x}_1 \ldots, \mathbf{x}_k$ by *Skolem* constants. We will use the same notation ψ_i for the i^{th} clause of $\psi_{\textsf{Skolem}}$. Then, consider a structure $\mathcal{M}$ such that $\mathcal{M} \models \bigcup\limits_{i=0}^{k} T_{\sigma_i}$ and $\mathcal{M} \models \phi_{\textsf{Skolem}}$. Now, suppose, on the contrary, that there is a clause ψ_j such that for every sort σ_i, we have $\mathcal{M} \not\models \forall(\mathbf{y}_i : \sigma_i)\psi_j$. This means, for every sort σ_i, there is a interpretation $\mathcal{I}_i$ (that extends $\mathcal{I}$ with valuations of $\mathbf{y}_i$), such that either $\mathcal{I}_i$ leads to falsity of T_{σ_i} or the clause ψ_j. Let $c_1^{\sigma_i}, c_2^{\sigma_i}, \ldots c_{|\mathbf{y}_i|}^{\sigma_i}$ be the values assigned to the universal variables $\mathbf{y}_i$ in $\mathcal{I}_i$. Then, construct an interpretation $\mathcal{I}'$ by extending $\mathcal{I}$ with the variables $\mathbf{y}_i$ interpreted with c^{σ_i}'s . This interpretation $\mathcal{I}'$ can be shown to either violate one of the theory axioms or the formula ψ_j. In either case, we have a contradiction. ◀

The *contract* L above identifies, for each clause ψ_j, one sort σ_i such that the restriction $\psi_j^{\sigma_i}$ of ψ_j to σ_i can be set to true. Thus, in order to decide satisfiability of ϕ_{CNF}, a straightforward decision procedure involves enumerating all contracts, $L \in [\{1, \ldots, r\} \to \mathcal{S}]$. For each contract L and for each sort σ_i, we construct the sentence ϕ_i^L, and make a black-box call to the $\exists^*\forall^*$ theory solver for T_{σ_i}. If there is a contract L for which each of these calls return 'SATISFIABLE', then ϕ_{CNF} (and thus, the original formula ϕ) is satisfiable. Otherwise, ϕ is unsatisfiable.

5 Undecidability Results

The logic that we have defined was carefully chosen to avoid undecidability of the satisfiability problem. We now show that natural generalizations or removal of restrictions in our logic renders the satisfiability problem undecidable. We believe our results are hence not simple to generalize any further.

One restriction that we have is that the functions that are existentially quantified cannot have σ_0 as their range sort. A related restriction is that the *universal* quantification block quantifies all uninterpreted function symbols, as otherwise they must be existentially quantified on the outside block.

Let us now consider the fragment of logic where formulas are of the form $(\exists \mathbf{x}_0)\,(\exists \mathbf{F})(\forall \mathbf{y}_0)\,\psi$ where in fact we do not even have any background theory. Since the formula is over a single sort, we have dropped the sort annotations on the variables. It is not hard to see that this logic is undecidable.

▶ **Theorem 9.** *Consider signature with a single sort σ_0 (and no background sorts). The satisfiability problem for sentences of the following form is undecidable.*

$$(\exists \mathbf{x}_0)\,(\exists \mathbf{F})(\forall \mathbf{y}_0)\,\psi$$

Proof (Sketch). We can show this as a mild modification of standard proofs of the undecidability of first-order logic. We can existentially quantify over a variable $Zero$ and a function $succ$, demand that for any element y, $succ(y)$ is not $Zero$, and for every y, y', if $succ(y) = succ(y')$, then $y = y'$. This establishes an infinite model with distinct elements $succ^n(Zero)$, for every $n \geq 0$. We can then proceed to encode the problem of *non-halting* of a 2-counter machine using a relation $R(t, q, c_1, c_2)$, which stands for the 2CM is in state q at time t with counters c_1 and c_2, respectively. It is easy to see that all this can be done using only universal quantification (the relation R can be modeled as a function easily). ◀

The theorem above has a simple proof, but the theorem is not new; in fact, even more restrictive logics are known to be undecidable (see [8]).

Based on Theorem 9, when the only sort we have is the foreground sort, quantifying functions existentially results in undecidability of the satisfiability problem. On the other hand, if we allow only relation symbols (over the foreground sort) to be existentially quantified, the satisfiability problem becomes decidable [5]. However, for certain theories different from the empty theory (such as Presburger arithmetic), existential quantification over relation symbols (without any quantified function symbols) can lead to undecidability, despite the decidability of checking satisfiability of the entire first order fragment. This is the content of Theorem 10 and sheds light on the importance of the restriction that the foreground sort is disjoint from the various background sorts in EQSMT .

▶ **Theorem 10.** *Consider a signature with a single sort σ_1 and let T_{σ_1} be the theory of Presburger arithmetic. The satisfiability problem is decidable for sentences of the form*

$$(\exists \mathbf{x}_1)\,(\exists \mathbf{R})\,(\forall \mathbf{y}_1)\,\psi$$

Proof (Sketch). We can use a similar proof as the theorem above, except now that we use the successor function available in Presburger arithmetic. We can again reduce non-halting of Turing machines (or 2-counter machines) to satisfiability of such formulas. ◀

Stepping further back, there are very few subclasses of first-order logic with equality that have a decidable satisfiability problem, and the only standard class that admits $\exists^*\forall^*$ prefixes is the Bernays-Schönfinkel-Ramsey class (see [5]). Our results can be seen as an extension of this class with background theories, where the background theories admit locally a decidable satisfiability problem for the $\exists^*\forall^*$ fragment.

6 Applications to Synthesis

6.1 Synthesis: Validity or Satisfiability?

Though we argued in Section 2 that synthesis problems can be modeled using satisfiability of EQSMT sentences, there is one subtlety that we would like to highlight. In synthesis problems, we are asked to find an expression such that the expression satisfies a specification expressed as a formula in some logic. Assuming the specification is modeled as a universally quantified formula over background theories, we would like to know if $\forall y \varphi(e, y)$ holds for the synthesized expression e. However, in a logical setting, we have to qualify what "holds" means; the most natural way of phrasing this is that $\forall y \varphi(e, y)$ is valid over the underlying background theories, i.e., holds in all models that satisfy the background theories. However, the existential block that models the existence of an expression is clearly best seen as a satisfiability problem, as it asks whether there is *some foreground model* that captures an expression. Requiring that it holds in *all* foreground models (including those that might have only one element) would be unreasonable.

To summarize, the synthesis problem is most naturally modeled as a logical problem where we ask whether there is *some* foreground model (capturing a program expression) such that *all* background models, that satisfy their respective background theories, also satisfy the quantifier free formula expressing that the synthesized expression satisfies the specification. This is, strictly speaking, neither a satisfiability problem nor a validity problem!

We resolve this by considering only *complete and consistent* background theories. Hence validity of a formula under a background theory T is *equivalent* to T-satisfiability. Consequently, synthesis problems using such theories can be seen as asking whether there is a foreground universe (modeling the expression to be synthesized) *and* some background models where the specification holds for the expression. We can hence model synthesis purely as a satisfiability problem of EQSMT, as described in Section 2.

Many of the background theories used in verification/synthesis and SMT solvers are complete theories (like Presburger arithmetic, FOL over reals, etc.). One incomplete theory often used in verification is the theory of *uninterpreted functions*. However, in this case, notice that since the functions over this sort are uninterpreted, validity of formulas can be modeled using a *universal quantification* over functions, which is supported in EQSMT! The only other adjustment is to ensure that this background theory has only infinite models (we can choose this background theory to be the theory of $(\mathbb{N}, =)$, which has a decidable satisfiability problem). Various scenarios such as modeling pointers in heaps, arrays, etc., can be naturally formulated using uninterpreted functions over this domain.

The second issue in modeling synthesis problems as satisfiability problems for EQSMT is that in synthesis, we need to construct the expression, rather than just know one exists. It is easy to see that if the individual background theory solvers support finding concrete values for the existentially quantified variables, then we can pull back these values across our reductions to give the values of the existentially quantified first-order variables (over all sorts), the existentially quantified function variables as well as the existentially quantified relation variables, from which the expression to be synthesized can be constructed.

6.2 Evaluation

We illustrate the applicability of our result for solving synthesis problems.

Synthesis of recursive programs involving lists.

We model the problem of synthesizing *recursive* programs with lists, that will meet a pre/post contract C assuming that recursive calls on smaller data-structures satisfy the same contract C. Though the programs we seek are recursive, we can model certain classes of programs using straight-line programs.

To see this, let us take the example of synthesizing a program that finds a particular key in a linked list (list-find). We can instead ask whether there is a straight-line program which takes an additional input which models the return value of a possible recursive call made on the tail of the list. The straight-line program must then work on the head of the list and this additional input (which is assumed to satisfy the contract C) to produce an output that meets the same contract C.

For this problem, we modeled the program to be synthesized using existential quantification (over a grammar that generates bounded length programs) as described in Section 2. The pointer `next` and recursive data structures `list, lseg` in the verification condition were modeled using *universal quantification* over function variables and relation variables, respectively. Moreover, in order to have a tractable verification condition, we used the technique of *natural proofs* [20, 26, 29] that soundly formulates the condition in a decidable theory. We used z3 [12] to ackermanize the universally quantified functions/relations (`lseg`, `list` and `next`). We encoded the resulting formula as a synthesis problem in the SyGuS format [4] and used an off-the-shelf enumerative counter-example guided synthesis (CEGIS) solver. A program was synthesized within 1s, which was manually verified to be correct.

We also encoded other problems involving lists : list-length (calculating the length of a list), list-sum (computing sum of the keys in a list), list-sorted (checking if the sequence of keys in the list is sorted) and list-count-occurrence (counting the number of occurrences of a key in the list), using a CEGIS solver, and report the running times and the number of programs explored in Table 1.

We are convinced that EQSMT can handle recursive program synthesis (of bounded size) against separation logics specifications expressed using natural proofs (as in [26]).

Synthesis of straight-line programs equivalent to given recursive programs.

In the second class of examples, we turn to synthesizing straight-line programs given a recursive function as their specification. For example, consider Knuth's generalization of the recursive McCarthy 91 function:

$$M(n) = \begin{cases} n - b & \text{if } n > a \\ M^c(n + d)) & \text{otherwise} \end{cases}$$

for every integer n, and where $(c - 1)b < d$. For the usual McCarthy function, we have $a = 100$, $b = 10$, $c = 2$, and $d = 11$.

Program	# Programs Explored in SyGuS	Time(s)
list-find	∼5k	0.5
list-length	∼40k	5
list-sum	∼160k	15
list-sorted	∼206k	45
list-count-occurrence	∼1.3 million	134
Knuth : $(a = 100,\ b = 10,\ c = 2,\ d = 11)$	-	2
Knuth : $(a = 15,\ b = 30,\ c = 3,\ d = 61)$	-	6
Knuth : $(a = 3,\ b = 20,\ c = 4,\ d = 62)$	-	27
Knuth : $(a = 9,\ b = 11,\ c = 5,\ d = 45)$	-	49
Knuth : $(a = 99,\ b = 10,\ c = 6,\ d = 51)$	-	224
Takeuchi	-	100

Table 1 Synthesis of list programs and recursive programs.

Consider the problem of synthesizing an equivalent recursion-free expression. The programs we consider may have if-then-else statements of nesting depth 2, with conditionals over linear expressions having unbounded constants. Existential quantification over the background arithmetic sort allowed us to model synthesizing these unbounded constants. Our specification demanded that the value of the expression for n satisfy the recursive equations given above.

We modeled the foreground sort *inside* arithmetic, and converted our synthesis problem to a first-order $\exists^*\forall^*$ sentence over Presburger arithmetic and Booleans. We experimented with several values for a, b, c, d (with $(c-1)b < d$), and interestingly, solutions were synthesized only when $(d-(c-1)b) = 1$. Given Knuth's result that a closed form expression involves taking remainder modulo this expression (and since we did not have the modulo operation in our syntax), it turns out that simple expressions do not exist otherwise. Also, whenever the solution was found, it matched the recursion-free expression given by Knuth (see Theorem 1 in [19]). In Table 1, we provide the running times of our implementation on various parameters. We also compared our implementation with the popular synthesis tool Sketch [34] on these examples. For the purpose of comparison, we used the same template for both Sketch and our implementation. Further, since Sketch does not allow encoding integers with unbounded size (unlike our encoding in integer arithmetic), we represented these constants, to be synthesized, using bitvectors of size 8. Sketch does not return an answer within the set time-limit of 10 minutes for most of these programs.

We also modeled the Tak function (by Takeuchi) given by the specification below.

$$t(x, y, z) = \begin{cases} y & \text{if } x \leq y \\ t(t(x-1, y, z), t(y-1, z, x), t(z-1, x, y)) & \text{otherwise} \end{cases}$$

Our implementation synthesized the program $\texttt{t(x,y,z)} = \texttt{ite}(\texttt{x} \leq \texttt{y}, \texttt{y}, \texttt{ite}(\texttt{y} \leq \texttt{z}, \texttt{z}, \texttt{x}))$ in about 100s.

7 Related Work

There are several logics known in the literature that can express synthesis problems and are decidable. The foremost example is the monadic second-order theory over trees, which can express Church's synthesis problem [10] and other reactive synthesis problems over *finite data domains*, and its decidability (Rabin's theorem [31]) is one of the most celebrated theorems in logic that is applicable to computer science. Reactive synthesis has been well studied and applied in computer science (see, for example, [7]). The work reported in [22] is a tad closer to program synthesis as done today, as it synthesizes syntactically restricted programs with recursion that work on finite domains.

Caulfield et al [11] have considered the decidability of *syntax-guided synthesis* (SyGuS) problems, where the synthesized expressions are constrained to belong to a grammar (with operators that have the usual semantics axiomatized by a standard theory such as arithmetic) that satisfy a universally quantified constraint. They show that the problem is undecidable in many cases, but identify a class that asks for expressions satisfying a regular grammar with uninterpreted function theory constraints to be decidable.

The $\exists^*\forall^*$ fragment of pure predicate logic (without function symbols) was shown to be decidable by Bernays and Schönfinkel (without equality) and by Ramsey (with equality) [5], and is often called Effectively Propositional Reasoning (EPR) class. It is one of the few fragments of first-order logic known to be decidable. The EPR class has been used in program verification [16, 25], and efficient SMT solvers supporting EPR have been developed [27].

The work by [1] extends EPR to stratified typed logics, which has some similarity with our restriction that the universes of the foreground and background be disjoint. However, the logic therein does not allow background SMT theories unlike ours and restricts the communication between universally and existentially quantified variables via equality between existential variables and terms with universally quantified variables as arguments. In [15], EPR with simple linear arithmetic (without addition) is shown to be decidable.

Theory extensions [33] and model theoretic and syntactic restrictions theoreof [32] have been explored to devise decidable fragment for quantified fragments of first order logic. Here, reasoning in *local* theory extensions of a base theory can be reduced to the reasoning in the base theory (possibly with an additional quantification). Combination of theories which are extensions of a common base theory can similarly be handled by reducing the reasoning to a decidable base theory. Similar ideas have been employed in the context of combinations of linear arithmetic and the theory of uninterpreted functions with applications to construct interpolants [18] and invariants [6] for program verification. EQSMT does not require the background theories to be extensions of a common base theory.

Verification of programs with arrays and heaps can be modeled using second order quantification over the arrays/heaps and quantifier alternation over the elements of the array/heaps which belong to the theory of Presburger arithmetic. While such a logic is, in general, undecidable, careful syntactic restrictions such as limiting quantifier alternation [9] and *flatness* restrictions [3]. We do not restrict the syntax of our formulae, but ensure decidability via careful sort restrictions. A recent paper [20] develops sound and complete reasoning for a so-called *safe* FO fragment of an uninterpreted combination of theories. However, the logic is undecidable, in general, and also does not support second-order quantification.

The SyGuS format has recently been proposed as a language to express *syntax guided* synthesis problems, and there have been several synthesis engines developed for various tracks of SyGuS [4]. However, the syntax typically allows unbounded programs, and hence the synthesis problem is not decidable. In [13], the candidate program components are 'decorated' with annotations that represent transformers of the components in a sound abstract domain. This reduces the synthesis problem ($\exists^*\forall^*$) to the search for a proof ($\exists^*\exists^*$) in the abstract domain.

When expressing synthesis problems for programs that manipulate heaps, we rely on natural-proofs style sound abstraction of the verification conditions. Natural synthesis [30] extends this idea to an inductive synthesis procedure.

8 Conclusions and Future Work

The logic EQSMT defined herein is meant to be a decidable logic for communication between researchers modeling program synthesis problems and researchers developing efficient logic solvers. Such liaisons have been extremely fruitful in verification, where SMT solvers have served this purpose. We have shown the logic to be decidable and its efficacy in modeling synthesis problems. However, the decision procedure has several costs that should not be paid *up front* in any practical synthesis tool. Ways to curb such costs are known in the literature of building efficient synthesis tools. In particular, searching for foreground models is similar to EPR where efficient engines have been developed [27], and the search can also be guided by CEGIS-like approaches [4]. And the exponential blow-up caused by guessing contracts between solvers (in Step 4 of our procedure) is similar to *arrangements* agreed upon by theories combined using the Nelson-Oppen method, again for which efficient solvers have been developed. Our hope is that researchers working on logic engines will engineer an efficient decision procedure for EQSMT that can solve synthesis problems.

References

1 Abadi, A., Rabinovich, A., Sagiv, M.: Decidable fragments of many-sorted logic. In: Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence and Reasoning. pp. 17–31. LPAR'07, Springer-Verlag, Berlin, Heidelberg (2007), `http://dl.acm.org/citation.cfm?id=1779419.1779423`

2 Ackermann, W., Ackermann, F.W.: Solvable cases of the decision problem (1954)

3 Alberti, F., Ghilardi, S., Sharygina, N.: Decision procedures for flat array properties. Journal of Automated Reasoning 54(4), 327–352 (Apr 2015), `https://doi.org/10.1007/s10817-015-9323-7`

4 Alur, R., Bodík, R., Dallal, E., Fisman, D., Garg, P., Juniwal, G., Kress-Gazit, H., Madhusudan, P., Martin, M.M.K., Raghothaman, M., Saha, S., Seshia, S.A., Singh, R., Solar-Lezama, A., Torlak, E., Udupa, A.: Syntax-guided synthesis. In: Dependable Software Systems Engineering, pp. 1–25. IOS Press (2015), `https://doi.org/10.3233/978-1-61499-495-4-1`

5 Bernays, P., Schönfinkel, M.: Zum entscheidungsproblem der mathematischen logik. Mathematische Annalen (1928)

6 Beyer, D., Henzinger, T.A., Majumdar, R., Rybalchenko, A.: Invariant synthesis for combined theories. In: Cook, B., Podelski, A. (eds.) Verification, Model Checking, and Abstract Interpretation. pp. 378–394. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)

7 Bloem, R., Galler, S., Jobstmann, B., Piterman, N., Pnueli, A., Weiglhofer, M.: Interactive presentation: Automatic hardware synthesis from specifications: A case study. In: Proceedings of the Conference on Design, Automation and Test in Europe. DATE '07 (2007)

8 Börger, E., Grädel, E., Gurevich, Y.: The classical decision problem. Springer Science & Business Media (2001)

9 Bradley, A.R., Manna, Z., Sipma, H.B.: What's decidable about arrays? In: Emerson, E.A., Namjoshi, K.S. (eds.) Verification, Model Checking, and Abstract Interpretation. pp. 427–442. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)

10 Buchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. Transactions of the American Mathematical Society (1969)

11 Caulfield, B., Rabe, M.N., Seshia, S.A., Tripakis, S.: What's decidable about syntax-guided synthesis? CoRR abs/1510.08393 (2015)

12 De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: TACAS (2008)

13 Gascón, A., Tiwari, A., Carmer, B., Mathur, U.: Look for the proof to find the program: Decorated-component-based program synthesis. In: Computer Aided Verification (2017)

14 Gulwani, S.: Dimensions in program synthesis. In: Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming. pp. 13–24. PPDP '10, ACM, New York, NY, USA (2010), `http://doi.acm.org/10.1145/1836089.1836091`

15 Horbach, M., Voigt, M., Weidenbach, C.: On the combination of the Bernays–Schönfinkel–Ramsey fragment with simple linear integer arithmetic. In: Proceedings of the International Conference on Automated Deduction. pp. 202–219 (2017)

16 Itzhaky, S., Banerjee, A., Immerman, N., Nanevski, A., Sagiv, M.: Effectively-propositional reasoning about reachability in linked data structures. In: International Conference on Computer Aided Verification (2013)

17 Jha, S., Gulwani, S., Seshia, S.A., Tiwari, A.: Oracle-guided component-based program synthesis. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1. pp. 215–224. ACM (2010)

18 Kapur, D., Majumdar, R., Zarba, C.G.: Interpolation for data structures. In: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software

Engineering. pp. 105–116. SIGSOFT '06/FSE-14, ACM, New York, NY, USA (2006), http://doi.acm.org/10.1145/1181775.1181789

19 Knuth, D.E.: Textbook examples of recursion. Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy (1991)

20 Löding, C., Madhusudan, P., Peña, L.: Foundations for natural proofs and quantifier instantiation. Proc. ACM Program. Lang. 2(POPL), 10:1–10:30 (Dec 2017), http://doi.acm.org/10.1145/3158098

21 Madhusudan, P., Mathur, U., Saha, S., Viswanathan, M.: A Decidable Fragment of Second Order Logic With Applications to Synthesis. In: Ghica, D., Jung, A. (eds.) 27th EACSL Annual Conference on Computer Science Logic (CSL 2018). Leibniz International Proceedings in Informatics (LIPIcs), vol. 119, pp. 31:1–31:19. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018), http://drops.dagstuhl.de/opus/volltexte/2018/9698

22 Madhusudan, P.: Synthesizing Reactive Programs. In: Computer Science Logic (CSL'11) - 25th International Workshop/20th Annual Conference of the EACSL (2011)

23 Manna, Z., McCarthy, J.: Properties of programs and partial function logic. Tech. rep. (1969)

24 Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Transactions on Programming Languages and Systems (TOPLAS) 1(2), 245–257 (1979)

25 Padon, O., McMillan, K.L., Panda, A., Sagiv, M., Shoham, S.: Ivy: safety verification by interactive generalization. ACM SIGPLAN Notices (2016)

26 Pek, E., Qiu, X., Madhusudan, P.: Natural proofs for data structure manipulation in c using separation logic. In: ACM SIGPLAN Notices (2014)

27 Piskac, R., de Moura, L., Bjørner, N.: Deciding effectively propositional logic with equality. Tech. rep., Technical Report MSR-TR-2008-181, Microsoft Research (2008)

28 Pnueli, A., Rodeh, Y., Strichman, O., Siegel, M.: The small model property: How small can it be? Information and computation (2002)

29 Qiu, X., Garg, P., Ştefănescu, A., Madhusudan, P.: Natural proofs for structure, data, and separation. ACM SIGPLAN Notices (2013)

30 Qiu, X., Solar-Lezama, A.: Natural synthesis of provably-correct data-structure manipulations. Proc. ACM Program. Lang. 1(OOPSLA), 65:1–65:28 (Oct 2017), http://doi.acm.org/10.1145/3133889

31 Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Transactions of the american Mathematical Society (1969)

32 Sofronie-Stokkermans, V.: Hierarchic reasoning in local theory extensions. In: Nieuwenhuis, R. (ed.) Automated Deduction – CADE-20. pp. 219–234. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

33 Sofronie-Stokkermans, V.: On combinations of local theory extensions. In: Programming Logics: Essays in Memory of Harald Ganzinger. pp. 392–413 (2013)

34 Solar-Lezama, A., Tancau, L., Bodik, R., Seshia, S., Saraswat, V.: Combinatorial sketching for finite programs. ACM SIGOPS Operating Systems Review (2006)

A Encoding M_{three} in EQSMT

We are interested in synthesizing a straight line program that implements the function M_{three}, and can be expressed as a `term` over the grammar in Figure 1(a).

Let us see how to encode this synthesis problem in EQSMT. First, let us fix the maximum height of the term we are looking for, say to be 2. Then, the program we want to synthesize can be represented as a tree of height at most 2 such that every node in the tree can have ≤ 3 child nodes (because the maximum arity of any function in the above grammar is 3, corresponding to `ite`). A skeleton of such a expression tree is shown in Figure 1(b). Every node in the tree is named according to its path from the root node.

The synthesis problem can then be encoded as the formula

$$\begin{aligned}
\phi_{M_{\mathsf{three}}} \equiv\ & (\exists n_0, n_{00}, n_{01}, \ldots n_{022} : \sigma_0)\ (\ \underbrace{\exists \mathsf{Left}, \mathsf{Mid}, \mathsf{Right} : \sigma_0, \sigma_0}_{\textbf{Existentially quantified relations}}\ )\\
& (\exists \mathtt{ADD}, \mathtt{ITE}, \mathtt{LTZero}, \mathtt{EQZero}, \mathtt{GTZero}, \mathtt{INPUT}, C_1, C_2, C_3 : \sigma_{\mathsf{label}})\\
& (\exists c_1, c_2, c_3 : \mathbb{N})\ (\ \underbrace{\exists f_{\mathsf{label}} : \sigma_0, \sigma_{\mathsf{label}}}_{\textbf{Existentially quantified functions}}\ )\\
& \quad \varphi_{\mathsf{well\text{-}formed}}\\
& \quad \wedge (\forall x : \mathbb{N})(\ \underbrace{\forall g^0_{\mathsf{val}}, g^1_{\mathsf{val}}, g^2_{\mathsf{val}}, g^3_{\mathsf{val}} : \sigma_0, \mathbb{N}}_{\textbf{Universally quantified functions}}\ )\ (\varphi_{\mathsf{semantics}} \implies \varphi_{\mathsf{spec}})
\end{aligned} \tag{3}$$

Here, the nodes are elements of the foreground sort σ_0. The binary relations Left, Mid, Right over the foreground sort will be used to assert that a node n is the left,middle, right child respectively of node n' : $\mathsf{Left}(n', n)$, $\mathsf{Mid}(n', n)$, $\mathsf{Right}(n', n)$. The operators or *labels* for nodes belong to the background sort σ_{label}, and can be one of `ADD` $(+)$, `ITE` $(\mathtt{ite})$, `LTZero` (< 0), `GTZero` (> 0), (`EQZero` $(= 0)$), `INPUT` (denoting the input to our program), or constants C_1, C_2, C_3 (for which we will synthesize natural constants c_1, c_2, c_3 in the (infinite) background sort $\mathbb{N}$). The function f_{label} assigns a label to every node in the program, and the formula $\varphi_{\mathsf{well\text{-}formed}}$ asserts some sanity conditions:

$$\begin{aligned}
\varphi_{\mathsf{well\text{-}formed}} \equiv & \bigwedge_{\rho \neq \rho'} n_\rho \neq n_{\rho'}\ \wedge \mathsf{Left}(n_0, n_{00}) \wedge \bigwedge_{\rho \neq 00} \neg(\mathsf{Left}(n_0, n_\rho))) \wedge \cdots\\
& \wedge \neg(\mathtt{ADD} = \mathtt{ITE}) \wedge \neg(\mathtt{ADD} = \mathtt{LTZero}) \wedge \cdots \wedge \neg(C_1 = C_3) \wedge \neg(C_2 = C_3)\\
& \wedge \bigwedge_{\rho} (f_{\mathsf{label}}(n_\rho) {=} \mathtt{ADD}) \vee (f_{\mathsf{label}}(n_\rho) {=} \mathtt{ITE}) \vee \cdots \vee (f_{\mathsf{label}}(n_\rho) {=} C_3)
\end{aligned} \tag{4}$$

The formula $\varphi_{\mathsf{semantics}}$ asserts that the 'meaning' of the program can be inferred from the meaning of the components of the program. The functions $g^0_{\mathsf{val}}, g^1_{\mathsf{val}}, g^2_{\mathsf{val}}, g^3_{\mathsf{val}}$, will assigns value to nodes from $\mathbb{N}$, for this purpose :

$$\varphi_{\mathsf{semantics}} \equiv \varphi_{\mathtt{ADD}} \wedge \varphi_{\mathtt{ITE}} \wedge \varphi_{\mathtt{LTZero}} \wedge \varphi_{\mathtt{EQZero}} \wedge \varphi_{\mathtt{GTZero}} \wedge \varphi_{\mathtt{INPUT}} \wedge \varphi_{C_1} \wedge \varphi_{C_2} \wedge \varphi_{C_3} \tag{5}$$

where each of the formulae $\varphi_{\mathtt{ADD}}, \cdots, \varphi_{C_3}$ specify the semantics of each node when labeled with these operations:

$$\begin{aligned}
\varphi_{\mathtt{ADD}} \equiv & \bigwedge_{\rho, \rho_1, \rho_2} \big(f_{\mathsf{label}}(n_\rho) = \mathtt{ADD} \wedge \mathsf{Left}(n_\rho, n_{\rho_1}) \wedge \mathsf{Mid}(n_\rho, n_{\rho_2})\\
& \implies \bigwedge_{i=0,1,2,3} g^i_{\mathsf{val}}(n_\rho) = g^i_{\mathsf{val}}(n_{\rho_1}) + g^i_{\mathsf{val}}(n_{\rho_2})\big)
\end{aligned} \tag{6}$$

$$\begin{aligned}\varphi_{\texttt{ITE}} \equiv \bigwedge_{\rho,\rho_1,\rho_2,\rho_3} \Big[& f_{\mathsf{label}}(n_\rho) = \texttt{ITE} \wedge \mathsf{Left}(n_\rho, n_{\rho_1}) \wedge \mathsf{Mid}(n_\rho, n_{\rho_2}) \wedge \mathsf{Right}(n_\rho, n_{\rho_3}) \\ & \implies \bigwedge_{i=0,1,2,3} \big(g^i_{\mathsf{val}}(n_{\rho_1}) = 1 \implies g^i_{\mathsf{val}}(n_\rho) = g^i_{\mathsf{val}}(n_{\rho_2}) \\ & \qquad \wedge\, g^i_{\mathsf{val}}(n_{\rho_1}) = 0 \implies g^i_{\mathsf{val}}(n_\rho) = g^i_{\mathsf{val}}(n_{\rho_3})\big)\Big]\end{aligned} \tag{7}$$

$$\begin{aligned}\varphi_{\texttt{LTZero}} & \equiv \bigwedge_{\rho,\rho_1} \Big[f_{\mathsf{label}}(n_\rho) = \texttt{LTZero} \wedge \mathsf{Left}(n_\rho, n_{\rho_1}) \\ & \implies \bigwedge_{i=0,1,2,3} \big(g^i_{\mathsf{val}}(n_{\rho_1}) < 0 \implies g^i_{\mathsf{val}}(n_\rho) = 1 \\ & \qquad \wedge\, g^i_{\mathsf{val}}(n_{\rho_1}) \geq 0 \implies g^i_{\mathsf{val}}(n_\rho) = 0\big)\Big]\end{aligned} \tag{8}$$

$$\begin{aligned}\varphi_{\texttt{EQZero}} & \equiv \bigwedge_{\rho,\rho_1} \Big[f_{\mathsf{label}}(n_\rho) = \texttt{LTZero} \wedge \mathsf{Left}(n_\rho, n_{\rho_1}) \\ & \implies \bigwedge_{i=0,1,2,3} \big(g^i_{\mathsf{val}}(n_{\rho_1}) = 0 \implies g^i_{\mathsf{val}}(n_\rho) = 1 \\ & \qquad \wedge\, g^i_{\mathsf{val}}(n_{\rho_1}) \neq 0 \implies g^i_{\mathsf{val}}(n_\rho) = 0\big)\Big]\end{aligned} \tag{9}$$

$$\begin{aligned}\varphi_{\texttt{GTZero}} & \equiv \bigwedge_{\rho,\rho_1} \Big[f_{\mathsf{label}}(n_\rho) = \texttt{LTZero} \wedge \mathsf{Left}(n_\rho, n_{\rho_1}) \\ & \implies \bigwedge_{i=0,1,2,3} \big(g^i_{\mathsf{val}}(n_{\rho_1}) > 0 \implies g^i_{\mathsf{val}}(n_\rho) = 1 \\ & \qquad \wedge\, g^i_{\mathsf{val}}(n_{\rho_1}) \leq 0 \implies g^i_{\mathsf{val}}(n_\rho) = 0\big)\Big]\end{aligned} \tag{10}$$

The formula $\varphi_{\texttt{INPUT}}$ states that for a node labeled `INPUT`, the value of that node is the input to M_{three}. Hence, such a node n_ρ evaluates to x, $x+61$, $g^1_{\mathsf{val}}(n_0)$ and $g^2_{\mathsf{val}}(n_0)$ respectively under g^0_{val}, g^1_{val}, g^2_{val} and g^3_{val}:

$$\begin{aligned}\varphi_{\texttt{INPUT}} \equiv \bigwedge_{\rho} \Big[& f_{\mathsf{label}}(n_\rho) = \texttt{INPUT} \implies \\ & \qquad g^0_{\mathsf{val}}(n_\rho) = x \\ & \qquad \wedge g^1_{\mathsf{val}}(n_\rho) = x + 61 \\ & \qquad \wedge g^2_{\mathsf{val}}(n_\rho) = g^1_{\mathsf{val}}(n_0) \\ & \qquad \wedge g^3_{\mathsf{val}}(n_\rho) = g^2_{\mathsf{val}}(n_0) \\ \Big] &\end{aligned} \tag{11}$$

Finally we have the semantics of constant labels:

$$\varphi_{C_1} \equiv \bigwedge_{\rho} \Big[f_{\mathsf{label}}(n_\rho) = C_1 \implies \bigwedge_{i=0,1,2,3} g^i_{\mathsf{val}}(n_\rho) = c_1\Big] \tag{12}$$

The formulae φ_{C_2} and φ_{C_3} are similar and thus skipped.

Last, the formula φ_{spec} expresses the specification of the program as in Equation (2).

$$\begin{aligned}\varphi_{\mathsf{spec}} \equiv & \big(x > 13 \implies g^0_{\mathsf{val}}(n_0) = x - 30\big) \\ & \wedge\big(x \leq 13 \implies g^0_{\mathsf{val}}(n_0) = g^3_{\mathsf{val}}(n_0)\big)\end{aligned} \tag{13}$$