An Infinite Needle in a Finite Haystack: Finding Infinite Counter-Models in Deductive Verification

NETA ELAD, Tel Aviv University, Israel
ODED PADON, VMware Research, USA
SHARON SHOHAM, Tel Aviv University, Israel

First-order logic, and quantifiers in particular, are widely used in deductive verification of programs and systems. Quantifiers are essential for describing systems with unbounded domains, but prove difficult for automated solvers. Significant effort has been dedicated to finding quantifier instantiations that establish unsatisfiability of quantified formulas, thus ensuring validity of a system's verification conditions. However, in many cases the formulas are satisfiable—this is often the case in intermediate steps of the verification process, e.g., when an invariant is not yet inductive. For such cases, existing tools are limited to finding *finite* models as counterexamples. Yet, some quantified formulas are satisfiable but only have *infinite* models, which current solvers are unable to find. Such infinite counter-models are especially typical when first-order logic is used to approximate the natural numbers, the integers, or other inductive definitions such as linked lists, which is common in deductive verification. The inability of solvers to find infinite models makes them diverge in these cases, providing little feedback to the user as they try to make progress in their verification attempts.

In this paper, we tackle the problem of finding such infinite models, specifically, finite representations thereof that can be presented to the user of a deductive verification tool. These models give insight into the verification failure, and allow the user to identify and fix bugs in the modeling of the system and its properties. Our approach consists of three parts. First, we introduce *symbolic structures* as a way to represent certain infinite models, and show they admit an efficient model checking procedure. Second, we describe an effective model finding procedure that symbolically explores a given (possibly infinite) family of symbolic structures in search of an infinite model for a given formula. Finally, we identify a new decidable fragment of first-order logic that extends and subsumes the many-sorted variant of EPR, where satisfiable formulas always have a model representable by a symbolic structure within a known family, making our model finding procedure a decision procedure for that fragment.

We evaluate our approach on examples from the domains of distributed consensus protocols and of heap-manipulating programs (specifically, linked lists). Our implementation quickly finds infinite counter-models that demonstrate the source of verification failures in a simple way, while state-of-the-art SMT solvers and theorem provers such as Z3, cvc5, and Vampire diverge or return "unknown".

CCS Concepts: • **Networks** → *Protocol testing and verification*; • **Software and its engineering** → **Software verification**; • **Theory of computation** → **Logic and verification**; **Automated reasoning**.

Additional Key Words and Phrases: deductive verification, counter-models, infinite models, Paxos

ACM Reference Format:
Neta Elad, Oded Padon, and Sharon Shoham. 2024. An Infinite Needle in a Finite Haystack: Finding Infinite Counter-Models in Deductive Verification . *Proc. ACM Program. Lang.* 8, POPL, Article 33 (January 2024), 31 pages. https://doi.org/10.1145/3632875

Authors' addresses: Neta Elad, Tel Aviv University, Tel Aviv, Israel, netaelad@mail.tau.ac.il; Oded Padon, VMware Research, Palo Alto, USA, oded.padon@gmail.com; Sharon Shoham, Tel Aviv University, Tel Aviv, Israel, sharon.shoham@cs.tau.ac.il.

ACM 2475-1421/2024/1-ART33
https://doi.org/10.1145/3632875

1 INTRODUCTION

A plethora of tools and techniques have emerged in recent years that use first-order logic (FOL) with quantifiers for verification [Ball et al. 2014; Feldman et al. 2017; Goel and Sakallah 2021a,b; Hance et al. 2021; Itzhaky et al. 2013; Karbyshev et al. 2017; Koenig et al. 2022; Löding et al. 2018; Ma et al. 2019; Mathur et al. 2020; McMillan and Padon 2018; Murali et al. 2022; Padon et al. 2017, 2022; Taube et al. 2018; Yao et al. 2022, 2021]. These works all use quantified formulas in FOL to encode the behavior of systems and specify desired properties and inductive invariants. Using FOL to model (or abstract) verification problems has the advantage of completeness, meaning that every formula has either a refutation or a satisfying model. Furthermore, FOL allows quantification, which is essential for reasoning about unbounded domains, such as the set of processes participating in a distributed protocol, the set of messages exchanged in a network, or the set of objects in the heap. Following previous work, we target FOL without theories, which focuses the verification challenge on dealing with quantifiers, while side-stepping the difficulty of combining theories with quantifiers and uninterpreted symbols.

Quantifiers provide expressivity, but pose a significant challenge for automated verification. SMT solvers [Barbosa et al. 2022; de Moura and Bjørner 2008; Dutertre and De Moura 2006] and theorem provers [Riazanov and Voronkov 1999; Weidenbach et al. 2009] often struggle and diverge while searching for a refutation or a satisfying model, in which case they time out or return "unknown".

In many cases this divergence is not simply bad luck, but a result of a more fundamental problem: formulas that are satisfiable but only have infinite models. Though infinite models typically do not represent concrete states of a system, their existence may arise from the use of FOL to approximate a stronger logic, for example least fixed-point. In such cases, the solution is to add additional axioms that eliminate spurious behaviors [Hozzová et al. 2021; Löding et al. 2018; Murali et al. 2022; Reynolds and Kuncak 2015]. Infinite counter-models could provide useful guidance on how to refine the FOL modeling with additional axioms, and ultimately allow the user to successfully complete the verification task. However, existing solvers cannot find infinite counter-models. As a result, when the FOL modeling is still missing some axiom (e.g., an induction principle), solvers diverge, leaving the user with an inconclusive result and no actionable information.

While a significant amount of research has been dedicated to developing algorithms and heuristics for finding quantifier instantiations for refutation [Bansal et al. 2015; Bradley et al. 2006; de Moura and Bjørner 2007; Detlefs et al. 2005; Dross et al. 2016; Ge and de Moura 2009; Jacobs 2009; Vazou et al. 2018], ensuring that finite models exist [Itzhaky et al. 2013; Padon et al. 2017; Taube et al. 2018], and discovering finite models [Lynch 2013; Reger et al. 2016; Reynolds et al. 2013a,b], less attention has been given to finding infinite models in the context of verification. Although there are works on proving the existence or constructing infinite models (e.g., based on a saturated set of clauses in resolution-based theorem proving [Bachmair and Ganzinger 2001; Peltier 2003], or using the model evolution calculus [Baumgartner et al. 2006; Baumgartner and Tinelli 2003]; see Sec. 8 for further discussion), finding infinite models has not (yet) made its way into common verification frameworks based on SMT solvers. In this paper, we aim for a technique for finding infinite models that can work well in the context of SMT-based deductive verification. To that end, we seek a representation of first-order structures that captures models that appear in real-world problems, admits an effective search procedure, and can be communicated to users.

This paper develops a finite representation of (certain) infinite models, and an effective search procedure for such models. Our key observation is that many of the infinite models arising in practice have repeating patterns that can be summarized in a finite way. We are motivated by examples from various domains such as heap-manipulating programs (in particular usage of linked lists) [Löding et al. 2018; Murali et al. 2022] and distributed protocols [Padon et al. 2017, 2016] where

a linearly ordered set is used to model reachability of objects in the heap or consecutive rounds of execution (respectively). Importantly, these linear orders are intended to be finite, but may be infinite under FOL semantics. Accordingly, we introduce *symbolic structures* as finite representations of possibly infinite models. Symbolic structures use finitely many *summary nodes* to capture possibly infinite sets of elements, and use the language of Linear Integer Arithmetic (LIA) to describe repeating patterns. This allows us to check if a symbolic structure satisfies an FOL formula by translating the formula into a (quantified) LIA formula, which solvers such as Z3 [de Moura and Bjørner 2008] or CVC5 [Barbosa et al. 2022] can solve quickly. Our LIA-based symbolic structures are particularly useful when verification uses FOL to (soundly) abstract common linear orders such as the natural numbers or the integers, or to model (semi-) linear data structures such as linked lists.

Given that checking if a specific symbolic structure satisfies a formula is reducible to LIA, we can obtain a procedure that searches for a symbolic model (i.e., a satisfying symbolic structure) for a given formula by enumerating symbolic structures. However, such a procedure is inefficient. Instead of enumerating all possible symbolic structures, we introduce an algorithm that can efficiently search a (possibly infinite) space of symbolic structures, given by a *template*. A template specifies the number of symbolic-structure nodes, as well as a subset of the LIA language to be used within the symbolic structures. The search algorithm avoids explicitly constructing all potential symbolic structures. Rather, it symbolically encodes the entire family of symbolic structures described by the template into a single (quantified) LIA formula that, when satisfiable, induces a symbolic model for the FOL formula at hand. This search procedure is sound for establishing the satisfiability of any formula in FOL: if a symbolic structure is found, the formula is satisfiable. Further, we suggest a simple heuristic template, parameterized by size, that captures non-trivial infinite models for a variety of examples. As a notable example, using such a template, we are able to find an infinite counter-model showing that the original inductive invariant of the Paxos distributed consensus protocol [Lamport 1998] is not inductive in its FOL formalization given in [Goel and Sakallah 2021b].

Although not every satisfiable FOL formula has a symbolic model (as expected, given that satisfiability in FOL is not recursively enumerable), we identify a new decidable fragment of FOL where that is the case, i.e., each satisfiable formula has a symbolic model. We further prove a "small symbolic model property," (1) bounding the size of potentially satisfying symbolic structures for formulas in the fragment, and (2) characterizing a finite subset of the LIA language that is sufficient for them. This makes our search algorithm a decision procedure for the fragment. Interestingly, our decidability proof departs from the widely-used approach of establishing a finite model property, as the underlying models we consider are infinite.

The new decidable fragment, called Ordered Self Cycle (OSC), is an extension of the many-sorted variant of the decidable Effectively PRopositional fragment (EPR, also known as the Bernays-Schönfinkel-Ramsey class) [Lewis 1980; Ramsey 1930][1]. Many-sorted EPR already proved useful in verifying systems from various domains, e.g., distributed protocols [Padon et al. 2017], networking systems [Ball et al. 2014], and heap-manipulating programs [Itzhaky et al. 2014, 2013]. OSC extends many-sorted EPR in that it relaxes some of the restrictions on quantifier alternations and function symbols. Namely, OSC allows one sort, equipped with a linear order, to have cyclic functions (i.e., functions mapping the sort to itself), which is strictly beyond many-sorted EPR, and can be useful in modeling verification problems.

[1]Following previous papers, e.g., [Padon et al. 2017], we would henceforth take EPR to refer to the extension of the original fragment from [Ramsey 1930] to many-sorted logic. The extension allows quantifier alternations and function symbols, as long as no cycles are introduced, see Sec. 6 for a precise definition. The proofs of decidability and small model property of EPR carry over to the many-sorted extension.

There is a rich literature on decidable fragments of first-order logic (see e.g. [Börger et al. 2001; Voigt 2019] and references therein). The OSC fragment is unique in that landscape as it extends EPR with cyclic functions, and, unlike monadic logics (e.g., [Löb 1967; Shelah 1977]), allows a linear order; to the best of our knowledge it is not subsumed by any existing decidable fragment.

In summary, this paper makes the following contributions:

- We introduce *symbolic structures*, a finite representation of infinite structures, in which functions and relations are defined using LIA terms and formulas (Sec. 4);
- We provide an efficient way to find a symbolic model across a given (possibly infinite) search space, without an explicit enumeration, by reduction to LIA (Sec. 5);
- We identify a new decidable fragment of FOL, which extends many-sorted EPR, where the satisfiability problem can be solved by restricting the search space to a set of symbolic structures of a bounded size, representing (possibly) *infinite* models. (Sec. 6);
- We implement a tool called FEST (Find and Evaluate Symbolic structures via Templates) that finds symbolic models for FOL formulas using heuristically computed templates, and evaluate it on examples from the domains of distributed protocols and of heap-manipulating programs (specifically, linked lists), which exhibit infinite counter-models. While existing tools fail to produce an answer, FEST finds symbolic models for all 21 examples in our evaluation. (Sec. 7).

The rest of this paper is organized as follows. Sec. 2 gives an informal overview of the key ideas. Sec. 3 provides necessary background. Sec. 4 to 7 present the main contributions listed above. Finally, Sec. 8 discusses related work and Sec. 9 concludes. For technical proofs, see [Elad et al. 2023b].

2 OVERVIEW

In this section we use a simple example to demonstrate our approach for representing infinite models and finding them automatically. The example also demonstrates the kind of verification problems expressible in the decidable OSC fragment.

2.1 Motivation

Quantification is prevalent when modeling systems over unbounded domains in first-order logic. As soon as quantifier alternations are introduced, infinite models, which are usually not intended, may arise. An example for the emergence of infinite models is the Paxos [Lamport 1998] distributed protocol for consensus. In [Padon et al. 2017], Paxos was verified in first-order logic, by carefully rewriting and adapting the formalization of both the protocol and its inductive invariant to ensure that the verification conditions are in EPR, avoiding problematic quantifier alternations. However, this process requires creativity and can be quite tricky. It is therefore desirable to verify Paxos using its natural first-order modeling, avoiding rewrites. Indeed, recently, [Goel and Sakallah 2021b] considered the natural FOL modeling of Paxos as a benchmark for automatically inferring inductive invariants. The invariants inferred match the natural hand-written ones [Lamport 1998], but to obtain a complete mechanically-checked proof, one needs to mechanically verify their inductiveness. (Recall that in a typical use-case of invariant inference, no reference inductive invariant is known.) Unfortunately, the inductiveness of the inferred Paxos invariants hinges on the property that in every moment in time, the sequence of rounds (also known as ballots) from the beginning of the protocol is finite, i.e., induction on rounds is required. This property is lost in the FOL modeling of the protocol. As a result, the invariants are *not* inductive in FOL, and exhibit an infinite counter-model, in which an infinite chain of rounds leads to a violation of consistency. This counter-model reveals the need for induction over the sequence of rounds as part of the inductiveness proof.

The infinite counter-model is unintended, and perhaps unnatural, but it is an unavoidable artifact of modeling in FOL. Therefore, a method to describe and find such infinite models is valuable for deductive verification in FOL. Presenting these models to the user provides a better experience than timing out, as current tools do, and allows the user to gain insight into the problem, and refine their modeling to eliminate these infinite models (e.g., by adding instances of induction axioms).

2.2 Running Example: The Echo Machine

Since Paxos is too complex to be used as a running example, we demonstrate our approach for representing and finding infinite counter-models on the verification problem of an Echo Machine, a simple system that captures concepts also found in more complex examples, such as Paxos.

The Echo Machine is a reactive system that starts by reading some value as input and echoing it, and later, upon request, recalls a previously echoed value and echoes it once more. It is designed to satisfy the safety property that any two values echoed by the machine are identical. We use the problem of verifying that the Echo Machine satisfies this property as our running example.

Modeling in First-Order Logic. We model the verification problem of the Echo Machine in first order logic, using an Ivy-like language [McMillan and Padon 2020; Padon et al. 2016], as presented in Fig. 1. The modeling introduces a **value** sort and a **round** sort, a constant `start` of sort **round** to record the start round, and a relation `echo`: **round** $\times$ **value** to record echoed values.

At first, modeling **round** explicitly as the natural numbers (or integers) using the corresponding theory might seem attractive. However, current off-the-shelf SMT solvers do not handle combinations of uninterpreted functions with interpreted numerical sorts well in the presence of quantified formulas. Indeed, in our example, they get stuck and ultimately return "unknown".

We therefore abstract the **round** sort into an uninterpreted sort, and introduce the $\prec$ relation, axiomatized to be a strict linear order relation, to represent the $<$ order over the natural numbers. Additionally, the `start` constant is axiomatized to be the least element in **round**, according to $\prec$.

The behavior of the Echo Machine is modeled as a first-order transition system, where the state of the system is maintained by the `echo` relation, and the transitions are `echo_start` and `echo_prev`. The semantics of each transition is given by a transition relation formula over a "double vocabulary", where the pre-state of a transition is denoted by the plain `echo` relation, and the post-state is denoted by $\texttt{echo}'$. For example, the transition relation for `echo_start` is:

$$\exists \texttt{v}.\ (\forall \texttt{V}.\ \neg\texttt{echo(start,V)}) \wedge (\forall \texttt{R},\texttt{V}.\ \texttt{echo}'\texttt{(R,V)} \leftrightarrow (\texttt{echo(R,V)} \vee (\texttt{R} = \texttt{start} \wedge \texttt{V} = \texttt{v}))). \quad (1)$$

That is, the transition is only enabled when no value is echoed at `start`, as given by the **assume** statement on line 13; the transition updates the `echo` relation by adding a single pair (`start`, `v`), as given by line 14. The `echo_prev` transition is enabled if no value has been echoed at round `r` (line 16), and value `v` has been echoed at some round `R` $\prec$ `r` (see line 9, `prev_echoed` definition); it then echoes `v` at round `r` (line 17). The initial condition, that no value has been echoed, and the safety property, that all echoed values are the same, are also written in FOL (lines 11 and 18, respectively).

To establish that the system satisfies the safety property, an *inductive invariant* that implies it is specified. An inductive invariant is a property that holds in the initial states of the system and is preserved by every transition, and hence holds in all the reachable states. The verification problem is then to check that the conjunction of the provided invariants forms an inductive invariant.

Checking inductiveness as a satisfiability problem. Verifying that a formula `INV` is inductive amounts to checking unsatisfiability of the *verification condition* (VC) formulas $\texttt{INV} \wedge \texttt{TR}_i \wedge \neg\texttt{INV}'$ for every transition, where $\texttt{TR}_i$ is the transition formula and $\texttt{INV}'$ is obtained from `INV` by substituting all mutable symbols by their primed versions. When the invariant is not inductive, a model of the

```
1  sort round, value
2  immutable individual start: round
3  mutable relation echo: round × value
4  immutable relation ≺: round × round
5  axiom strict_linear_order(≺)
6  axiom ∀R. R ≠ start ⇒ start ≺ R
7
8  definition prev_echoed(r: round, v: value):
9    ∃R. R ≺ r ∧ echo(R, v)
10
11 init ∀R,V. ¬echo(R, V)
12 transition echo_start(v: value):
13   assume ∀V. ¬echo(start, V)
14   echo(start, v) ← true
15 transition echo_prev(r: round, v: value):
16   assume (∀V. ¬echo(r, V))
       ∧ prev_echoed(r, v)
17   echo(r, v) ← true
18 invariant [safety] ∀V1,V2.
     ((∃R.echo(R, V1)) ∧ (∃R.echo(R, V2)))
     ⇒ V1 = V2
19 invariant ∀R,V.
     (echo(R, V) ∧ R ≠ start)
     ⇒ prev_echoed(R, V)
```

Fig. 1. The Echo Machine in pseudo-Ivy.

VC formula for some transition is a *counterexample to induction* (CTI)—a pair of states where the pre-state satisfies the invariant but the post-state obtained after executing the transition does not.[2]

It is easy to see that the Echo Machine satisfies its safety property, but the property alone is not inductive, as it is not preserved by the `echo_start` transition: starting from a state where one value has been echoed at round $\neq$ `start`, the `echo_start` transition can echo a different value at round `start`, violating the property. (SMT solvers are able to find such a CTI.) To make the safety property inductive, the invariant in line 19 is added, stating that every value that is echoed at round $\neq$ `start` was previously echoed. This is meant to ensure that all echoed values are equal to the one echoed at `start`, which proves the safety property.[3]

Despite its simplicity, this example is beyond known decidable fragments of FOL and existing SMT solvers as well as resolution-based theorem provers are unable to prove (or disprove) it.

2.3 Infinite Counterexample to Induction

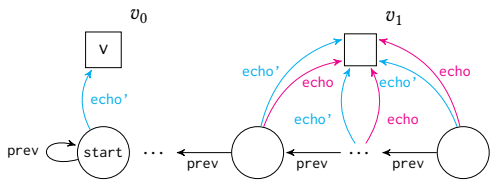
In fact, the inability of solvers to handle this problem is not incidental. In contrast to the intuition above (that holds when rounds correspond to natural numbers), the strengthened invariant is still *not* inductive in the FOL semantics, and a CTI for transition `echo_start` still exists, as we explain next. However, the CTI consists of infinitely many rounds, which existing solvers have no way of representing, let alone constructing a CTI with.

Before we explain the CTI, we note that the pre-state must satisfy the invariant of line 19, that is, for every round and value in the `echo` relation there should exist a previous round when the value was echoed. This $\forall\exists$ quantifier alternation leads to a Skolem function `prev`: **round** $\times$ **value** $\longrightarrow$ **round**, which we make explicit in the CTI. Intuitively, the Skolem function maps each pair of round `r` and value `v` in the `echo` relation to the witnessing (lower) round that made `v` safe to echo at round `r`.

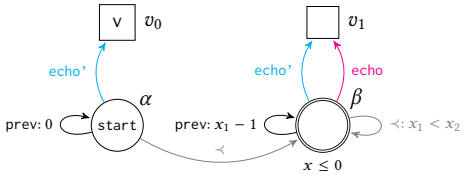
The CTI, depicted in Fig. 2a, consists of an infinite decreasing sequence of rounds, all of which are greater than the initial round `start`. The `prev` function is defined arbitrarily for `start` (and any value), and as the predecessor w.r.t. $\prec$ for other rounds (and any value). In the pre-state, all rounds except for `start` have echoed the same value v_1; we can see that this state indeed satisfies both of the invariants. However, by executing `echo_start` where a value `v` $= v_0 \neq v_1$ is chosen for `start`, we reach an unsafe state of the system.

[2]Some papers consider the pre-state of the transition violating inductiveness of the invariant as a CTI. In this paper, we follow the convention that the CTI consists of both the pre-state and the post-state.

[3]The mechanism of the Echo Machine for ensuring that a value is "safe to echo," by introducing rounds and using a previous round as a witness, is reminiscent of the consensus mechanism in Paxos [Lamport 1998], where a value is "safe to propose" if it was previously voted for by a member of a quorum and none of the quorum members voted for another value since.



(a) An infinite CTI. The interpretation of `start` and `v` are depicted as annotations of the corresponding elements. Interpretations depicted in magenta are relevant for the pre-state, cyan for the post-state, and black for both. The interpretation of `prev` is depicted as arrows from rounds to rounds, with the intention that for each round, the image of the function is the same for both values. The interpretation of $\prec$ orders the rounds from left to right.



(b) A symbolic structure representing the infinite CTI from (a). Double lines denote summary nodes, for which the subset formulas are written underneath. Terms and formulas defining functions, respectively relations, are written on the edges, using the syntax $a : b$, where a is a function or relation and b is a term or formula, respectively. The formula on the edge is omitted when it is $\top$; and the entire edge is omitted when the formula is $\bot$.

Fig. 2. A CTI for `echo_start`. Values are depicted as squares and rounds as circles.

Fixing the model based on the infinite CTI. The infinite CTI stems from the fact that, in the FOL semantics, each echoed value being echoed previously does *not* imply that it was echoed at `start`. The CTI reveals that reaching this conclusion requires induction on the rounds. Indeed, when the user adds the following instance of the induction scheme:

$$\forall \mathtt{V}.\ (\exists \mathtt{R}.\ \mathtt{echo(R,\ V)}) \Longrightarrow \exists \mathtt{R}.\ \mathtt{echo(R,\ V)} \wedge \forall \mathtt{R'}.\ \mathtt{R'} \prec \mathtt{R} \Longrightarrow \neg \mathtt{echo(R',\ V)}$$

stating that if some value is echoed, then there is also a minimal round in which it is echoed, the verification condition formulas become unsatisfiable, and existing solvers manage to prove it.

Our goal. Unfortunately, despite its simplicity, to the best of our knowledge, no existing solver is able to produce the infinite CTI. As such, the user receives no feedback from the solver, which either diverges attempting to prove the verification conditions, or gives up and returns "unknown". Changing this unfortunate outcome is the goal of our work.

2.4 Symbolic Structures for Compacting the Infinite Pattern

The key to our approach for finding infinite counter-models is observing that though infinite, the CTI in Fig. 2a can be described compactly: it has one unique `start` round, in which no value is echoed in the pre-state and v_0 is echoed in the post-state; and a repeating pattern of infinitely many rounds, each echoing v_1 (in both the pre- and post-state) and pointing to the `prev` one as a witness for satisfying the invariant at line 19. Our *symbolic structures* (Sec. 4) generalize this observation and formalize a way to express infinite models with similarly structured repetition. A symbolic structure capturing the infinite model from Fig. 2a is depicted in Fig. 2b.

Symbolic structure domain. Symbolic structures use "regular nodes" to represent single elements, such as the `start` round (α in Fig. 2b), as well as the individual v_0 and v_1 values; and "summary nodes" to represent (infinitely) repeating patterns in the model. A summary node represents a (copy of a) subset of the integers, described by some Linear Integer Arithmetic (LIA) formula. In our example, all rounds besides `start` are grouped into a summary node β, which is defined to represent the integers ≤ 0.

Each node in the symbolic structure represents a set of *explicit elements*, which together form the domain of the infinite model represented by the symbolic structure. An explicit element is

a pair comprised of a node and an integer "index". For uniformity, we use a pair with index 0 for regular nodes. For example, the explicit elements for the symbolic structure of Fig. 2b are $\langle\alpha, 0\rangle, \langle\beta, 0\rangle, \langle\beta, -1\rangle, \langle\beta, -2\rangle, \cdots$.

Interpretation. A symbolic structure defines the interpretation of functions over its nodes by specifying both the target node, and a LIA term to select one of its explicit elements. This term may also refer to the index of the source nodes. For the `prev` function in Fig. 2b:

$$\texttt{prev}(\alpha, v_0) = \texttt{prev}(\alpha, v_1) = \langle\alpha, 0\rangle \qquad \texttt{prev}(\beta, v_0) = \texttt{prev}(\beta, v_1) = \langle\beta, x_1 - 1\rangle$$

where x_1 represents the index of the first argument. This means that the `prev` of the (only) explicit element formed by the regular node α is the same node (for any value), and the `prev` of each explicit element $\langle\beta, x_1\rangle$, formed by the summary node β, is the explicit element of β whose index is smaller by 1, i.e., $\langle\beta, x_1 - 1\rangle$.

Interpretations of relations are similarly defined by specifying for each tuple of nodes, which of their explicit elements are in the relation. This is done by assigning a LIA formula over the indices of the nodes. In our example, the `echo`, `echo'`, and $\prec$ relations are defined as follows:

$$\texttt{echo}(\beta, v_1) = \top \qquad \texttt{echo'}(\alpha, v_0) = \top \qquad \alpha \prec \beta = \top$$
$$\texttt{echo'}(\beta, v_1) = \top \qquad \beta \prec \beta = x_1 < x_2$$

where x_1, x_2 represent the indices of the first and second argument, respectively, and for all other tuples the relations are defined as $\bot$. The definition $\alpha \prec \beta = \top$ means *all* explicit elements of α are $\prec$-related to *all* explicit elements of β. The definition $\beta \prec \beta = x_1 < x_2$ means that among the explicit elements associated with β, elements are only $\prec$-related to the elements whose integer indices are strictly larger than theirs, forming an infinite $\prec$-decreasing sequence.

Model checking symbolic structures. How do we know that the infinite model represented by the symbolic structure, also called the *explication of the symbolic structure*, indeed satisfies the formula of interest? It turns out that the *model checking* problem for symbolic structures reduces to checking LIA formulas (Thm. 4.7). Given the symbolic structure in Fig. 2b, we transform the original VC formula into a (quantified) LIA formula. Universal quantifiers are transformed into a combination of conjunction (over nodes) and universal LIA quantifiers (over indices), and similarly for existential quantifiers (with disjunctions and existential LIA quantifiers). Function and relation applications are transformed into the LIA terms and formulas that interpret them in the symbolic structure. Critically, the resulting LIA formula has no uninterpreted functions or relations. As we show in Sec. 4.2, the resulting LIA formula is true (in LIA) if and only if the explication of the symbolic structure satisfies the original FOL formula.

For example, in the VC formula of the Echo Machine, the anti-reflexivity axiom for "$\prec$" is expressed as $\forall x\colon \textbf{round}.\ \neg(x \prec x)$. To check that the explication of the symbolic structure satisfies this sub-formula, it is translated into $\forall x\colon \textbf{int}.\ (x = 0 \rightarrow \neg\bot) \wedge (x \leq 0 \rightarrow \neg(x < x))$, which is true in LIA. The first conjunct is generated from the regular node α, where the explicit elements are characterized by $x = 0$ and the definition of $\alpha \prec \alpha$ is $\bot$; the second conjunct is generated from the summary node β, where the explicit elements are characterized by $x \leq 0$ and the definition of $\beta \prec \beta$ results in $x < x$. In this way, the translation "bakes" the nodes into the formula and keeps only the LIA definitions from the symbolic structure to be checked in LIA.

2.5 Finding Symbolic Structures via Symbolic Search

The reducibility of the model-checking problem for symbolic structures to LIA establishes its decdiability. (Recall that LIA is decidable even with quantifiers.) Our next goal is obtaining an efficient process, by which to find a symbolic model (i.e., a satisfying symbolic structure) for a

given formula. Enumeration is an example for a sound procedure that searches for a symbolic model, but it is inefficient. Instead, we introduce the concept of a *symbolic-structure template*, as a way to describe a family of symbolic structures, by limiting the terms and formulas allowed in summary nodes to some finite subset of the LIA language. Such a template induces a search space of symbolic structures, and in Sec. 5 we develop an efficient method to symbolically consider all symbolic structures within that space and search among them for a possibly infinite satisfying model for a given formula.

Our search procedure encodes the entire family of symbolic structures induced by a template using auxiliary Boolean and integer variables. It transforms the original FOL formula into a (quantified) LIA formula whose structure is similar to the structure of the model checking formula except for the use of the auxiliary variables. The resulting formula is satisfiable in LIA whenever at least one of the symbolic structures in the family satisfies the original formula (Thm. 5.4), in which case the assignment to the auxiliary variables induces the sought symbolic structure.

The symbolic encoding moves the complexity of finding the right symbolic structure in a given family into the hands of existing SMT solvers, which are able to solve LIA formulas quickly. However, this symbolic search requires specifying a template in order to constrain the search space. Rather than putting the burden of defining such a template on the user, we propose a simple (yet robust, see Sec. 7) heuristic for computing a candidate template syntactically from the formula. The template we propose uses simple building blocks for function terms and relation formulas: all functions use the LIA terms 0 and $x_i \pm k$; unary relations use $\top$, $\bot$, and $x_i = 0$, binary relations additionally allow comparisons $x_i \bowtie x_j \pm k$, and ternary relations additionally allow a "between" formula $x_i \leq x_j < x_k$ (see Fig. 6 for the precise definition).

By using such a template and enumerating over the number of nodes, finding a satisfying symbolic structure for the Echo Machine takes less than one second.

2.6 Decidable Fragment

Rather than a fluke, we show that in the case of the Echo Machine, considering the proposed templates is not just a heuristic. We prove that for the verification problem of the Echo Machine (and other problems obeying a syntactic restriction) the set of symbolic structures induced by such a template is always sufficient, provided that the formula is indeed satisfiable. Note that for arbitrary satisfiable formulas it is not guaranteed that a symbolic model exists, let alone one from a simple, predefined, family of templates. This is because not every infinite model can be represented by a symbolic structure, and some satisfiable formulas are bound to have models that are not the explication of any symbolic structure.

That said, in Sec. 6 we identify a new fragment of FOL, the *Ordered Self Cycle (OSC)* fragment, where all satisfiable formulas have a (possibly infinite) model that is representable by a symbolic structure, thus making the satisfiability problem in this fragment decidable. The VC formula of the Echo machine falls into OSC, both before and after the instance of the induction scheme is added.

OSC extends the many-sorted variant of the Effectively PRopositional (EPR) decidable fragment of FOL. While many-sorted EPR forbids all function definitions that form cycles between sorts, OSC extends it to allow one "infinite" sort (**round** in our example) that is allowed to have cyclic functions (such as `prev`), any number of relations where the infinite sort appears at most once (such as the `echo` relation), and one binary relation, axiomatically defined to be a strict linear order, and conventionally denoted $\prec$. Linear order is a common primitive in many verification problems, especially when abstracting the naturals or the integers in FOL, or when verifying properties of linear data-structures such as linked lists. The ability to add cyclic functions to the vocabulary, which is strictly beyond EPR, makes the language more expressive and well-suited for a wider variety of verification problems.

We prove the decidability of checking satisfiability of a formula in OSC by showing the fragment enjoys a "small symbolic model property," bounding the number of nodes in candidate symbolic models in terms of syntactic parameters of the formula, and restricting the subset of the LIA language needed in function terms and relation formulas (Thm. 6.1). I.e., the small symbolic model property provides a cap for enumerating the number of nodes, and induces a definition of candidate templates, which is strictly covered by our heuristic: for formulas in OSC only the $\prec$ relation may use comparisons, and other relations only require $\top$ and $\bot$. This makes our symbolic search a decision procedure for OSC: when enumerating the heuristic templates up to the bound, if no symbolic model is found, the formula is necessarily unsatisfiable. A notable property of our decidability proof is that while the symbolic structures are finite (and bounded in size), the models they represent can be infinite. Thus, OSC does *not* enjoy a finite model property, as opposed to many decidable fragments, including EPR: for EPR, the syntactic restrictions imply a bound on the number of ground terms, from which a *finite* model can always be constructed. In contrast, in the case of OSC, the number of ground terms is infinite due to the cyclic functions, and the models, accordingly, are potentially infinite.

2.7 Applicability of Symbolic Structures

The "small symbolic model property" establishes that the simple templates we consider are complete for OSC. While the OSC fragment does not cover all of the examples we consider in this paper (e.g., complex distributed protocols), it does provide important theoretical justification for designing the search space of symbolic structures and using our heuristic for computing a template for a formula. These templates turn out to be valuable for a myriad of complex formulas that lie outside OSC.

We implemented our symbolic search, combined with the heuristically computed symbolic-structure templates, in our FEST tool (Find and Evaluate Symbolic structures via Templates) and successfully applied our approach to examples that admit infinite counter-models, which can be eliminated by additional FOL axioms (e.g., instances of an induction scheme). The examples are taken from the domains of distributed protocols (including five variants of Paxos [Lamport 1998]), and of properties of linked lists [Löding et al. 2018]. All 21 examples in our evaluation, are solved by FEST, which quickly finds infinite counter-models. Z3 [de Moura and Bjørner 2008], CVC5 [Barbosa et al. 2022] and Vampire [Riazanov and Voronkov 1999] all fail to solve these examples.

3 BACKGROUND

First-Order Logic. We use the usual definition of many-sorted first-order logic (FOL) with equality. A vocabulary Σ consists of a set of sorts $\mathcal{S}$, and constant, function and relation symbols, each with a sorted signature. In the cases where Σ has a single sort, we will omit $\mathcal{S}$ and the signatures of the symbols. We denote by $\mathcal{X}$ the set of variables used in FOL (we assume that each variable has a designated sort, and omit sorts when they are clear from the context). Terms t over Σ are either constants c, variables x, or well-sorted function applications of the form $f(t_1, \ldots, t_k)$. Formulas are defined recursively, where atomic formulas are either $t_1 \approx t_2$ (we use $\approx$ as the logical equality symbol) or $R(t_1, \ldots, t_k)$, where R is a relation symbol, and non-atomic formulas are built using Boolean connectives $\neg, \wedge, \vee, \rightarrow$ and quantifiers $\forall, \exists$. Given a formula φ (term t), $\mathrm{FV}(\varphi)$ ($\mathrm{FV}(t)$) denotes the set of free variables appearing in φ (t). A term t is *ground* if $\mathrm{FV}(t) = \emptyset$. A formula φ is *closed* if $\mathrm{FV}(\varphi) = \emptyset$. Models and assignments are defined in the usual way. A formula φ is satisfiable if it has a satisfying model M, denoted $M \models \varphi$. We denote by $\top$ a quantifier-free closed tautology, and by $\bot$ a contradiction.

Linear Integer Arithmetic. The interpreted theory of Linear Integer Arithmetic (LIA) is defined over a vocabulary that includes the sort of integers, and standard constants, functions and relations

of linear arithmetic $(0, 1, +, -, <)$. Many SMT solvers support extensions of LIA where all integers are considered constant symbols, functions include multiplication and division by a constant, and relations include short-hand comparisons $(>, \leq, \geq)$, and equality modulo a constant. We adopt these extensions, and denote by $\mathcal{T}_{\text{LIA}}$ the set of all LIA terms, and by QF_{LIA} the set of all quantifier-free LIA formulas. We distinguish between the LIA order $<$ and any axiomatized FOL order relation, for which we typically use $\prec$. We denote by $\models_{\text{LIA}}$ satisfiability in the standard model of integer arithmetic, and recall that checking satisfiability of (quantified) LIA formulas is decidable [Cooper 1972] and supported by SMT solvers [Barbosa et al. 2022; de Moura and Bjørner 2008].

4 SYMBOLIC STRUCTURES AS REPRESENTATIONS OF INFINITE MODELS

In this section we introduce symbolic structures as a way to represent (certain) infinite structures. We show how the model-checking problem for models represented by symbolic structures reduces to the problem of checking satisfiability of a LIA formula, and is therefore decidable.

4.1 Symbolic Structures

Given a many-sorted vocabulary Σ with a set of sorts $\mathcal{S}$, we define a *symbolic structure* as a triple $S = (\mathcal{D}, \mathcal{B}, \mathcal{I})$, where $\mathcal{D}$ is the *domain* of S, $\mathcal{B}$ is an assignment of *bound formulas* to the domain, and $\mathcal{I}$ is an *interpretation* of the symbols in Σ, defined as follows.

The *domain* of S maps each sort $\boldsymbol{s} \in \mathcal{S}$ to a finite, non-empty, set of nodes $\mathcal{D}(\boldsymbol{s})$, also denoted $\mathcal{D}_{\boldsymbol{s}}$, such that the domains of distinct sorts are disjoint, i.e., $\mathcal{D}_{\boldsymbol{s}} \cap \mathcal{D}_{\boldsymbol{s}'} = \emptyset$ for any two sorts $\boldsymbol{s} \neq \boldsymbol{s}'$. For each sort $\boldsymbol{s} \in \mathcal{S}$, the domain $\mathcal{D}_{\boldsymbol{s}}$ is partitioned into "summary nodes", denoted $\mathcal{D}^S_{\boldsymbol{s}}$, and "regular nodes", denoted $\mathcal{D}^R_{\boldsymbol{s}}$, such that $\mathcal{D}_{\boldsymbol{s}} = \mathcal{D}^S_{\boldsymbol{s}} \uplus \mathcal{D}^R_{\boldsymbol{s}}$. Intuitively, each summary node represents a (partial) copy of the integers, while each regular node represent an individual element.

The *bound* assignment maps, for every sort $\boldsymbol{s}$, every node $n \in \mathcal{D}_{\boldsymbol{s}}$ to a satisfiable LIA formula $\mathcal{B}(n) \in \text{QF}_{\text{LIA}}$ with at most one free variable, $\text{FV}(\mathcal{B}(n)) \subseteq \{x\}$. We further require that for regular nodes, the bound formula is always $x \approx 0$[4]. We think of the bound formula as representing a subset of the integers — those that satisfy $\mathcal{B}(n)$ — and use the formula and the subset of $\mathbb{Z}$ it represents interchangeably. Intuitively, $\mathcal{B}(n)$ restricts the copy of the integers represented by n to the elements in $\mathcal{B}(n)$ (for regular nodes, this is always a single integer, 0).

Given $z \in \mathcal{B}(n)$, we refer to $\langle n, z\rangle$ as an explicit element of n, and denote the union set of explicit elements for a set of nodes N by $\mathcal{E}_{\mathcal{B}}(N) = \{\langle n, z\rangle \mid n \in N, z \in \mathcal{B}(n)\}$.

The *interpretation* function maps each symbol to its interpretation. A constant symbol c of sort $\boldsymbol{s}$ is mapped to an element of some node of sort $\boldsymbol{s}$: $\mathcal{I}(c) \in \mathcal{E}_{\mathcal{B}}(\mathcal{D}_{\boldsymbol{s}})$, also denoted c^S.

The interpretation of a function symbol $f\colon \boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_k \to \boldsymbol{s}$ is a function that maps nodes of the argument sorts to an image node of the range sort, accompanied by an image LIA term: $\mathcal{I}(f)\colon \mathcal{D}_{\boldsymbol{s}_1} \times \cdots \times \mathcal{D}_{\boldsymbol{s}_k} \to \mathcal{D}_{\boldsymbol{s}} \times \mathcal{T}_{\text{LIA}}$, also denoted f^S. We further require that if $f^S(n_1, \ldots, n_k) = \langle n, s\rangle$, then $\text{FV}(s) \subseteq \{x_i \mid 1 \leq i \leq k\}$, and that $\bigwedge_{i=1}^k \mathcal{B}(n_i)[x_i/x] \models_{\text{LIA}} \mathcal{B}(n)[s/x]$. Intuitively, we assign a free variable x_i to each argument n_i, representing the specific element of node n_i to which the function is applied. The image term s selects an element of the image node for each combination of argument elements. The requirement $\bigwedge_{i=1}^k \mathcal{B}(n_i)[x_i/x] \models_{\text{LIA}} \mathcal{B}(n)[s/x]$ ensures that whenever the argument elements are elements of the corresponding nodes $n_1, \ldots, n_k$, the selected element is an element of n. In particular, if the image node n is a regular node, the term must evaluate to 0.

The interpretation of a relation symbol $R\colon \boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_k$ is a function that maps tuples of nodes of the relevant sorts to a LIA formula: $\mathcal{I}(R)\colon \mathcal{D}_{\boldsymbol{s}_1} \times \cdots \times \mathcal{D}_{\boldsymbol{s}_k} \to \text{QF}_{\text{LIA}}$, also denoted R^S. We further require that $\text{FV}\big(R^S(n_1, \ldots, n_k)\big) \subseteq \{x_i \mid 1 \leq i \leq k\}$. Intuitively, the tuples of integers that satisfy the LIA formula define the elements of nodes n_i for which the relation holds.

[4]We assign a bound formula to regular nodes only for the uniformity of the presentation.

Example 4.1. The symbolic structure of Fig. 2b is $S = (\mathcal{D}, \mathcal{B}, \mathcal{I})$, where $\mathcal{D}(\textbf{value}) = \{v_0, v_1\}$, $\mathcal{D}(\textbf{round}) = \{\alpha, \beta\}$; v_0, v_1, α are regular nodes with $\mathcal{B}(n) = x \approx 0$; β is a summary node with $\mathcal{B}(\beta) = x \leq 0$; and

$$\mathtt{start}^S = \langle \alpha, 0 \rangle \qquad \mathtt{v}^S = \langle v_0, 0 \rangle$$

$$\mathtt{prev}^S(n_1, n_2) = \begin{cases} \langle \alpha, 0 \rangle & n_1 = \alpha \\ \langle \beta, x_1 - 1 \rangle & n_1 = \beta \end{cases} \qquad n_1 \prec^S n_2 = \begin{cases} \top & n_1 = \alpha, n_2 = \beta \\ x_1 < x_2 & n_1 = n_2 = \beta \\ \bot & \text{otherwise} \end{cases}$$

$$\mathtt{echo}^S(n_1, n_2) = \begin{cases} \top & n_1 = \beta, n_2 = v_1 \\ \bot & \text{otherwise} \end{cases} \qquad \mathtt{echo'}^S(n_1, n_2) = \begin{cases} \top & n_1 = \alpha, n_2 = v_0 \\ \top & n_1 = \beta, n_2 = v_1 \\ \bot & \text{otherwise} \end{cases}$$

An example of a symbolic structure with two summary nodes can be found in [Elad et al. 2023b].

Explication. A symbolic structure $S = (\mathcal{D}, \mathcal{B}, \mathcal{I})$ represents a (possibly) infinite first-order structure, defined by its explication $M = \mathcal{E}(S)$.

The domain of sort $s \in \mathcal{S}$ in M, denoted $\mathcal{D}_s^M$, is the set of explicit elements $\mathcal{E}_{\mathcal{B}}(\mathcal{D}_s)$, i.e., it consists of all pairs $\langle n, z \rangle$ of nodes n in $\mathcal{D}_s$ and integers z that satisfy $\mathcal{B}(n)$. Thus, each summary node may contribute infinitely many elements to the explicit domain, while each regular node contributes a single element. Note that the requirements for bound formulas to be satisfiable ensures that the explicit domains of all sorts are non-empty.

The interpretation of the symbols in M, denoted α^M for symbol α, is defined as follows. For a constant symbol c, $c^M = c^S$. For a function symbol f of arity k, $f^M(\langle n_1, z_1 \rangle, \ldots, \langle n_k, z_k \rangle) = \left\langle n, \bar{v}_{\text{LIA}}(s) \right\rangle$, where $\langle n, s \rangle = f^S(n_1, \ldots, n_k)$, v_{LIA} is an assignment such that $v_{\text{LIA}}(x_i) = z_i$, and $\bar{v}_{\text{LIA}}$ extends v_{LIA} to all LIA terms in the standard way. That is, the function maps the elements $\langle n_1, z_1 \rangle, \ldots, \langle n_k, z_k \rangle$ of the explicit domain to an element contributed by node n that is determined by evaluating the term s on the specific elements z_i of nodes n_i. For a relation symbol R, $R^M = \left\{ (\langle n_1, z_1 \rangle, \ldots, \langle n_k, z_k \rangle) \mid v_{\text{LIA}} \models_{\text{LIA}} R^S(n_1, \ldots, n_k) \text{ where } v_{\text{LIA}}(x_i) = z_i \in \mathcal{B}(n_i) \right\}$. That is, the tuples of explicit elements in the relation are determined by evaluating the LIA formula $R^S(n_1, \ldots, n_k)$ on the specific elements z_i of nodes n_i.

Example 4.2. Let us explicate the symbolic structure from Ex. 4.1. The explicit domains are $\mathcal{D}^M(\textbf{value}) = \{\langle v_0, 0 \rangle, \langle v_1, 0 \rangle\}$ and $\mathcal{D}^M(\textbf{round}) = \{\langle \alpha, 0 \rangle\} \cup \{\langle \beta, z \rangle \mid z \leq 0\}$. The explication of the constants is: $\mathtt{start}^M = \langle \alpha, 0 \rangle$ and $\mathtt{v}^M = \langle v_0, 0 \rangle$. As for the function prev and the relations:

$$\mathtt{prev}^M(\langle n, z \rangle, v) = \begin{cases} \langle \alpha, 0 \rangle & n = \alpha \\ \langle \beta, z - 1 \rangle & n = \beta \end{cases}$$

$$\prec^M = \{(\langle \alpha, 0 \rangle, \langle \beta, z \rangle) \mid z \leq 0\} \cup \{(\langle \beta, z_1 \rangle, \langle \beta, z_2 \rangle) \mid z_1 < z_2 \leq 0\}$$
$$\mathtt{echo}^M = \{(\langle \beta, z \rangle, \langle v_1, 0 \rangle) \mid z \leq 0\}$$
$$\mathtt{echo'}^M = \{(\langle \beta, z \rangle, \langle v_1, 0 \rangle) \mid z \leq 0\} \cup \{(\langle \alpha, 0 \rangle, \langle v_0, 0 \rangle)\}$$

The resulting explicit model is isomorphic to the infinite model depicted in Fig. 2a.

Remark 4.3. *In fact, for the Echo Machine, the degenerate bound formula $\mathcal{B}(\beta) = \top$ would produce a symbolic structure whose explication is a counter-model to the VC formula just as well (even though it is not isomorphic to Fig. 2a). One may wonder therefore if bound formulas are necessary, or do regular nodes (where $\mathcal{B}(n) = x \approx 0$) and unbounded summary nodes (where $\mathcal{B}(n) = \top$) suffice. An example that requires a summary node with a bound formula other than $\top$ (e.g., $x \geq 0$) is provided in [Elad et al. 2023b].*

4.2 Model Checking a Symbolic Structure

A symbolic structure S represents an explicit model $M = \mathcal{E}(S)$. In this section we address the problem of checking if $\mathcal{E}(S)$ satisfies a given FOL formula φ, in which case, we also say that S satisfies φ (with abuse of notation). This is done by a transformation that "bakes" the symbolic structure into the formula, resulting in a LIA formula that is true if and only if $\mathcal{E}(S) \models \varphi$.

We define the transformation recursively (over the inductive structure of formulas). As a result, we must consider both closed formulas and formulas with free variables. To that end, we first define assignments in symbolic structures.

Symbolic assignments. If φ has free variables, then its value in M depends also on an assignment $v\colon \mathcal{X} \to \mathcal{D}^M$ (which respects the sorts). We refer to such an assignment as an *explicit* assignment. Recall that $\mathcal{D}^M \subseteq \mathcal{D}^S \times \mathbb{Z}$, i.e., explicit assignments map each variable to a node paired with an integer. For the purpose of defining the transformation to LIA, we introduce *symbolic assignments* where explicit integers are replaced by integer variables. In the sequel, let $\mathcal{X}^{\text{LIA}} = \{x^{\text{LIA}} \mid x \in \mathcal{X}\}$ be a set of integer variables such that each variable $x \in \mathcal{X}$ has a unique *symbolically corresponding* LIA variable $x^{\text{LIA}} \in \mathcal{X}^{\text{LIA}}$.

Definition 4.4. Given a symbolic structure S, a *symbolic assignment* is a mapping $v_S\colon \mathcal{X} \to \mathcal{D}^S \times \mathcal{X}^{\text{LIA}}$ such that for every variable $x \in \mathcal{X}$, $v_S(x) = \langle n, x^{\text{LIA}}\rangle$ for some $n \in \mathcal{D}^S$.

That is, a symbolic assignment fixes the node n to which each variable x is mapped, but leaves the element of n symbolic, captured by the integer variable x^{LIA} that uniquely corresponds to x.

We denote by $\bar{v}_S$ the extension of v_S to arbitrary terms over Σ. The extended assignment $\bar{v}_S$ maps terms to nodes paired with LIA terms over $\mathcal{X}^{\text{LIA}}$. Formally, $\bar{v}_S\colon \mathcal{T} \to \mathcal{D}^S \times \mathcal{T}_{\text{LIA}}$ is defined by setting $\bar{v}_S(x) = v_S(x)$; $\bar{v}_S(c) = c^S$; and $\bar{v}_S(f(t_1, \ldots, t_k)) = \langle n, s[s_i/x_i]\rangle$, where $\bar{v}_S(t_i) = \langle n_i, s_i\rangle$ and $f^S(n_1, \ldots, n_k) = \langle n, s\rangle$. That is, if $\bar{v}_S$ interprets the terms t_i as $\langle n_i, s_i\rangle$, and the symbolic structure defines $f^S(n_1, \ldots, n_k) = \langle n, s\rangle$, then $\bar{v}_S$ interprets $f(t_1, \ldots, t_k)$ as the node n and the term obtained from s by substituting s_i for x_i.

Example 4.5. For the symbolic structure defined in Ex. 4.1, we can define an assignment v_S such that $v_S(x_1) = \left\langle \beta, x_1^{\text{LIA}} \right\rangle$. Given the term $t = \texttt{prev}(\texttt{prev}(x_1))$, $\bar{v}_S(S) = \left\langle \beta, \left(x_1^{\text{LIA}} - 1\right) - 1 \right\rangle$.

The correspondence between FOL variables and their LIA counterparts allows us to "decompose" an explicit assignment v into a symbolic assignment where all integers are replaced by variables, accompanied by a LIA assignment to $\mathcal{X}^{\text{LIA}}$, which captures the explicit integers from v.

Definition 4.6. Given an explicit assignment $v\colon \mathcal{X} \to \mathcal{D}^M$, its *symbolization* is the symbolic assignment $v_S\colon \mathcal{X} \to \mathcal{D}^S \times \mathcal{X}^{\text{LIA}}$ defined by $v_S(x) = \langle n, x^{\text{LIA}}\rangle$ whenever $v(x) = \langle n, z\rangle$.

The *residual* assignment of v is the LIA assignment $v_{\text{LIA}}\colon \mathcal{X}^{\text{LIA}} \to \mathbb{Z}$ defined by $v_{\text{LIA}}(x^{\text{LIA}}) = z$ whenever $v(x) = \langle n, z\rangle$.

We are now ready to define the recursive transformation that reduces model checking of symbolic structures in FOL formulas to checking satisfiability of LIA formulas by "baking" a symbolic structure and an explicit assignment into the FOL formula.

Model checking symbolic structures by transforming FOL formulas to LIA. Given a FOL formula φ, a symbolic structure S and an explicit assignment v, we use v_S, the symbolization of v, to translate φ into a LIA formula. When evaluated with the residual assignment, v_{LIA}, this formula captures whether $\mathcal{E}(S)$ and v satisfy φ. The transformation, $\text{trans}^S_{v_S}(\varphi)$, is defined in Fig. 3.

The transformation uses the LIA terms and formulas that interpret function and relation symbols in the symbolic structure as building blocks to transform the formula. Specifically, if v_S interprets the terms t_i as $\langle n_i, s_i\rangle$, then the atomic formula $R(t_1, \ldots, t_k)$ is transformed to the LIA formula $R^S(n_1, \ldots, n_k)$, which the symbolic structure defines as the interpretation of R on nodes $n_1, \ldots, n_k$, except that the variables $x_1, \ldots, x_k$ that may appear in $R^S(n_1, \ldots, n_k)$ are substituted by the LIA terms $s_1, \ldots, s_k$.

Similarly, the atomic formula $t_1 \approx t_2$ is transformed to the LIA formula $s_1 \approx s_2$ if the nodes n_1 and n_2 are the same, requiring that t_1, t_2 are mapped not only to the same node but also to the same element of the node, and transformed to $\bot$ otherwise.

The transformation of the Boolean connectives is straightforward.

A universal quantifier $\forall x$ is transformed into a universal quantifier over an integer variable x^{LIA}, such that for every node n in the symbolic structure (captured by a conjunction over all nodes of the

$$\begin{aligned}
\mathrm{trans}^S_{v_S}(R(t_1,\ldots,t_k)) &= R^S(n_1,\ldots,n_k)[s_i/x_i] \text{ where } \bar{v}_S(t_i) = \langle n_i, s_i\rangle \\
\mathrm{trans}^S_{v_S}(t_1 \approx t_2) &= s_1 \approx s_2 \text{ if } n_1 = n_2 \text{ and } \bot \text{ otherwise, where } \bar{v}_S(t_i) = \langle n_i, s_i\rangle \\
\mathrm{trans}^S_{v_S}(\neg\varphi) &= \neg\left(\mathrm{trans}^S_{v_S}(\varphi)\right) \\
\mathrm{trans}^S_{v_S}(\varphi \circ \psi) &= \mathrm{trans}^S_{v_S}(\varphi) \circ \mathrm{trans}^S_{v_S}(\psi) \qquad \text{for } \circ \in \{\wedge, \vee, \rightarrow\} \\
\mathrm{trans}^S_{v_S}(\forall x\colon s.\varphi) &= \forall x^{\mathrm{LIA}}.\ \bigwedge_{n\in\mathcal{D}_s}\left(\mathcal{B}(n)[x^{\mathrm{LIA}}/x] \rightarrow \mathrm{trans}^S_{v_S[\langle n, x^{\mathrm{LIA}}\rangle/x]}(\varphi)\right) \\
\mathrm{trans}^S_{v_S}(\exists x\colon s.\varphi) &= \exists x^{\mathrm{LIA}}.\ \bigvee_{n\in\mathcal{D}_s}\left(\mathcal{B}(n)[x^{\mathrm{LIA}}/x] \wedge \mathrm{trans}^S_{v_S[\langle n, x^{\mathrm{LIA}}\rangle/x]}(\varphi)\right)
\end{aligned}$$

Fig. 3. Model-checking symbolic structure S and assignment v by translation to LIA.

corresponding sort), all the elements of n, captured by assignments to x^{LIA} that satisfy the bound formula of n, satisfy the body of the quantified formula. Similarly for an existential quantifier.

Note that the quantifier structure of $\mathrm{trans}^S_{v_S}(\varphi)$ is the same as that of φ, except that quantification is over integer variables. Further, if $\mathrm{FV}(\varphi) \subseteq \{x_1,\ldots,x_k\}$ then $\mathrm{FV}(\mathrm{trans}^S_{v_S}(\varphi)) \subseteq \{x_1^{\mathrm{LIA}},\ldots,x_k^{\mathrm{LIA}}\}$. Moreover, as expected, if two symbolic assignments agree on all the free variables of φ, then they result in the same LIA formula. In particular, for a closed formula, the transformation does not depend on the symbolic assignment used. The following theorem, proven by structural induction on φ, summarizes the correctness of the transformation for model checking a symbolic structure.

Theorem 4.7. *Let S be a symbolic structure, v an explicit assignment, v_S its symbolization and v_{LIA} the residual assignment. Then for every formula φ over Σ the following holds:*

$$\mathcal{E}(S), v \models \varphi \iff v_{LIA} \models_{LIA} trans^S_{v_S}(\varphi).$$

Example 4.8. Continuing with our running example, let us consider checking whether the symbolic structure from Ex. 4.1 satisfies the formula $\varphi = \forall x_2.\neg\left(\underline{\mathtt{prev}(\mathtt{prev}(x_1))} \prec x_2\right)$ under some explicit assignment v where $v(x_1) = \langle\beta, -7\rangle$. The transformation of φ w.r.t. v_S is

$$\mathrm{trans}^S_{v_S}(\varphi) = \forall x_2^{\mathrm{LIA}}.\ \left(x_2^{\mathrm{LIA}} \approx 0 \rightarrow \neg\bot\right) \wedge \left(x_2^{\mathrm{LIA}} \leq 0 \rightarrow \neg\left(\underline{\left(x_1^{\mathrm{LIA}} - 1\right) - 1} < x_2^{\mathrm{LIA}}\right)\right).$$

(The blue color highlights the transformation of the atomic formula and the underline depicts the transformation of the corresponding term.) Recall that in this example $v_{\mathrm{LIA}}(x_1^{\mathrm{LIA}}) = -7$. Therefore, $v_{\mathrm{LIA}} \not\models \mathrm{trans}^S_{v_S}(\varphi)$, which is expected since $\mathcal{E}(S), v \not\models \varphi$ (see Ex. 4.2).

4.3 Beyond Single-Dimension Symbolic Structures

We illustrate one limitation of symbolic structures, and sketch a possible way to overcome it. The summary nodes we have seen thus far represent a single dimension of infinity (a single copy of $\mathbb{Z}$), and symbolic structures can compactly describe infinite structures with relatively simple repeating patterns using those summary nodes. However, they cannot represent structures with more complex, e.g., "nested", repeating patterns. One such example would be a structure where domain elements are linearly ordered in a fashion akin to $\omega \times \omega$.

Specifically, consider a vocabulary $\Sigma = \{f(\cdot), P(\cdot), R(\cdot,\cdot)\}$, where f is a unary function, P is a unary relation, and R is a binary relation. The formula φ^+, defined by the conjunction of the formulas in Tab. 1, requires its models to include an infinite repetition of infinitely many P's, followed by infinitely many $\neg P$'s.

Table 1. Conjuncts of φ^+, which has a model akin to $\omega \times \omega$.

Conjunct	Meaning
$\forall x, y, z.(R(x, y) \wedge R(y, z)) \rightarrow R(x, z)$	R is transitive
$\forall x.\neg R(x, x)$	R is anti-reflexive
$\forall x, y.R(x, y) \vee R(y, x) \vee x \approx y$	R is linear
$\forall x.R(x, f(x))$	f is "increasing"
$\forall x, y.R(x, y) \rightarrow (R(f(x), y) \vee f(x) \approx y)$	f is a "successor"
$\forall x.P(x) \leftrightarrow P(f(x))$	f preserves P
$\forall x.\exists y, z.R(x, y) \wedge R(x, z) \wedge P(y) \wedge \neg P(z)$	P alternates infinitely

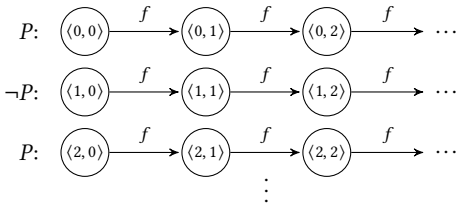


Fig. 4. Model M for φ^+, which is linearly ordered in a fashion akin to $\omega \times \omega$. The relation R is reflected by the ordering of the nodes, which are ordered top to bottom, left to right. P alternates between the rows, i.e., either P holds for an entire row or $\neg P$ holds for an entire row (as noted to the left). The successor function f always stays within the same row, mapping each node to its immediate neighbor to the right.

The conjunction φ^+ is satisfiable by the infinite model $M = \left(\mathcal{D}^M, \mathcal{I}^M\right)$, depicted in Fig. 4, where $\mathcal{D}^M = \mathbb{N} \times \mathbb{N}$ and $\mathcal{I}^M$ is defined as

$$f^M(\langle a, b\rangle) = \langle a, b+1\rangle \quad \vdots \quad P^M = \{\langle a, b\rangle \mid a \equiv 0 \mod 2\}$$
$$R^M = \{(\langle a_1, b_1\rangle, \langle a_2, b_2\rangle) \mid a_1 < a_2 \vee (a_1 = a_2 \wedge b_1 < b_2)\}.$$

I.e., the domain consists of pairs of integers, f^M increments the second integer in each pair (along the horizontal dimension in Fig. 4), P^M holds whenever the first integer in a pair is even (alternating along the vertical dimension in Fig. 4), and R^M is the lexicographical order over pairs of integers.

In order to represent this structure using a symbolic structure, infinitely many summary nodes would be needed. However, symbolic structures can be naturally generalized to allow representation of such structures by (1) letting each summary node represent a subset of $\mathbb{Z} \times \mathbb{Z}$ instead of $\mathbb{Z}$, and, accordingly, (2) attaching to each summary node n two LIA variables, x^1 and x^2, to be used in bound formulas, and in the LIA terms and formulas that interpret the function and relation symbols. The intention is that an assignment $\left[x^1 \mapsto z_1, x^2 \mapsto z_2\right]$ represents the explicit element $\langle n, z_1, z_2\rangle$ of n.

With this generalization, the structure M from above, which satisfies φ^+, can be represented by a symbolic structure with a single summary node, $T = (\mathcal{D} = \{n\}, \mathcal{B}, \mathcal{I})$, where $\mathcal{B}$ and $\mathcal{I}$ are defined as

$$\mathcal{B}(n) = x^1 \geq 0 \wedge x^2 \geq 0 \quad \vdots \quad f^T(n) = \left\langle n, x_1^1, x_1^2 + 1\right\rangle$$
$$P^T(n) = x_1^1 \equiv 0 \mod 2 \quad \vdots \quad R^T(n, n) = x_1^1 < x_2^1 \vee (x_1^1 = x_2^1 \wedge x_1^2 < x_2^2).$$

where x_i^j is the variable representing the j'th copy of $\mathbb{Z}$ associated with the node used as the i'th argument of a function / relation. That is, the lower indices refer to different arguments, as before, while the upper indices refer to one of the copies of $\mathbb{Z}$ within a summary node. T represents an infinite model, which is isomorphic to the model M above. (The domain $\mathcal{D}^M = \mathbb{N} \times \mathbb{N}$ maps straightforwardly to the explicit domain of T, $\{\langle n, z_1, z_2\rangle \in \{n\} \times \mathbb{Z} \times \mathbb{Z} \mid z_1 \geq 0 \wedge z_2 \geq 0\} = \{n\} \times \mathbb{N} \times \mathbb{N}$.)

We can further generalize and let the multiplicity of the copies of $\mathbb{Z}$ represented by each node be a parameter of either the symbolic structure (as a whole) or each summary node (regular nodes can be thought of as 0-dimension summary nodes). Both the explication and the translation at the heart of the model checking procedure for symbolic structures can be generalized to multi-dimensional symbolic structures such that the model-checking theorem (Thm. 4.7) still holds. However, note that with any generalization, the fact that symbolic structures are encoded in a finite way and that their model-checking is decidable means that some formulas will not have a symbolic structure: any

recursively enumerable set of models, for which model-checking is decidable, cannot be complete since FOL satisfiability is not in RE.

For the remainder of the paper, we exclusively use single-dimension symbolic structures, since they effectively handle the examples we consider.

5 CHECKING SATISFIABILITY BY FINDING SYMBOLIC STRUCTURES

We now present an algorithm for finding a satisfying symbolic structure for a given FOL formula within a given search space. Broadly, given a FOL formula, we leverage an SMT solver to *symbolically* search for a symbolic model in a (possibly infinite) family of symbolic structures given by a *template* (defined below). The idea is that given some FOL formula φ and a template, we construct a LIA formula where a satisfying assignment induces a symbolic model for φ. This process is sound for any formula, i.e., whenever a symbolic model is found, its explication satisfies the formula. In Sec. 6 we present a class of formulas, for which this process is also complete: i.e., for any formula in that class, there is a computable template, such that if the formula is satisfiable, a symbolic model is guaranteed to be found when searching using this template.

Symbolic-structure templates. To define the search space of symbolic structures, i.e., a symbolic structure family, we use the notion of a *symbolic-structure template*. A symbolic-structure template, similarly to a symbolic structure, fixes a finite set of nodes for each sort's domain, but uses *finite sets* of LIA terms and formulas with additional free variables as *candidates* for the bound formulas and the interpretation of functions and relations. Formally:

Definition 5.1. A *symbolic-structure template* for a vocabulary Σ is a triple $S^\sharp = (\mathcal{D}, \mathcal{B}^\sharp, \mathcal{I}^\sharp)$ where $\mathcal{D}$ is a symbolic structure domain, $\mathcal{B}^\sharp$ is a *finite* set of bound formulas for summary nodes, and $\mathcal{I}^\sharp$ defines the space of interpretations for function and relation symbols: for every function symbol $f \in \Sigma$, $\mathcal{I}^\sharp(f) \subseteq \mathcal{T}_{\text{LIA}}$ and for every relation symbol $R \in \Sigma$, $\mathcal{I}^\sharp(R) \subseteq \text{QF}_{\text{LIA}}$, such that $\mathcal{I}^\sharp(f)$ and $\mathcal{I}^\sharp(R)$ are finite.

We distinguish two kinds of free variables in bound formulas, function terms and relation formulas in $S^\sharp$. *Symbolic-structure free variables* are the same as the free variables in symbolic structures, used to identify explicit elements in nodes. Namely, they consist of x in bound formulas and of $x_1, x_2, \ldots$ in function terms and relation formulas, according to their arity (see Sec. 4). All other free variables are considered *template free variables*, conventionally denoted $d, d^1, d^2, \ldots$. These variables are not part of a symbolic structure definition; they are used to capture infinitely many similar LIA terms and formulas, and are replaced by an interpreted LIA constant (integer) when the template is instantiated.

A symbolic-structure template $S^\sharp = (\mathcal{D}, \mathcal{B}^\sharp, \mathcal{I}^\sharp)$ represents the set of all symbolic structures S over the domain $\mathcal{D}$, where bound formulas are taken from $\mathcal{B}^\sharp$ for summary nodes and are $x \approx 0$ for regular nodes, constants are interpreted as regular nodes, the terms and formulas in the interpretations of function and relation symbols are taken from $\mathcal{I}^\sharp$, with all template free variables substituted by integers. Note that these integers may be different in each use of a term or formula. For example, two nodes may use the same bound formula $x \geq d \in \mathcal{B}^\sharp$ with different substitutions, resulting in bounds $x \geq 0, x \geq 1$. The formal definition follows, where $SFV(a)$ denotes the set of template free variables appearing in a term or formula a.

Definition 5.2. A symbolic structure $S = (\mathcal{D}, \mathcal{B}, \mathcal{I})$ is *represented by* a symbolic structure skeleton $S^\sharp = \left(\mathcal{D}, \mathcal{B}^\sharp, \mathcal{I}^\sharp\right)$ if (i) for every regular node $n \in \mathcal{D}$, $\mathcal{B}(n) = x \approx 0$, and for every summary node $n \in \mathcal{D}$, $\mathcal{B}(n) = \psi\left[z_1/d^1, \ldots, z_m/d^m\right]$, where $\psi \in \mathcal{B}^\sharp$, $z_1, \ldots, z_m \in \mathbb{Z}$, and $SFV(\varphi) = \left\{d^1, \ldots, d^m\right\}$; similarly, (ii) for every $n_1, \ldots, n_k \in \mathcal{D}$ and $f \in \Sigma$ (resp. $R \in \Sigma$), $f^T(n_1, \ldots, n_k) =$

$\langle n, s[z_1/d^1, \ldots, z_m/d^m]\rangle$ (resp. $R^T(n_1, \ldots, n_k) = \psi[z_1/d^1, \ldots, z_m/d^m]$) for some node $n \in \mathcal{D}$, integers $z_1, \ldots, z_m$ and $s \in \mathcal{I}^\sharp(f)$ (resp. $\psi \in \mathcal{I}^\sharp(R)$) s.t. $SFV(t) = \{d^1, \ldots, d^m\}$ (resp. $SFV(\psi) = \{d^1, \ldots, d^m\}$).

In the sequel, we refer to the template and the set of symbolic structures it represents interchangeably, and write $S \in S^\sharp$ when S is represented by $S^\sharp$.

Example 5.3. To capture a family of symbolic structures where all summary nodes represent a full copy of $\mathbb{Z}$, we can define a template $S_1^\sharp$ where $\mathcal{B}_1^\sharp = \{\top\}$. The bound formula $\top$ has neither template free variables nor symbolic-structure free variables. A richer family of symbolic structures, where summary nodes may also represent the sets of positive or negative integers, can be captured by a template $S_2^\sharp$ where $\mathcal{B}_2^\sharp = \{\top, x \geq 1, x \leq -1\}$. The bound formulas $x \geq 1, x \leq -1$ contain the symbolic-structure free variable x and no template free variables. Finally, to capture a family of symbolic structures where summary nodes may represent any half-space of the integers, we can use instead the bound formulas $x \geq d, x \leq d$, where d is a template free variable (and x is still a symbolic-structure free variable). Specific symbolic structures within this template would replace d with specific LIA (integer) constants; for example, $x \geq 3$ or $x \leq -2$.

Symbolic structure finding through LIA encoding. To check if a symbolic-structure template $S^\sharp$ includes a symbolic model for a FOL formula, we use an encoding similar to the one used in Sec. 4.2 for model checking an individual symbolic structure. The key difference is that the bound formulas and the interpretation are not (fully) specified in $S^\sharp$, and, instead, need to be found. To account for this, the encoding introduces Boolean auxiliary variables that keep track of which interpretation and bound formulas are used, and LIA auxiliary variables that keep track of the LIA constants assigned to template free variables, such that each symbolic structure in $S^\sharp$ is induced by some assignment to the auxiliary variables.

Auxiliary variables. We use the following Boolean auxiliary variables: a variable $b_{n:\,\psi}$ for any summary node n and any bound formula $\psi \in \mathcal{B}^\sharp$, represents the decision to use ψ as the bound formula for n; a variable $b_{c \to n}$ for any constant $c \in \Sigma$ and any regular node n, represents the decision to define c^T as $\langle n, 0\rangle$; a variable $b_{f(n_1,\ldots,n_k)\to\langle n,s\rangle}$ for any function $f \in \Sigma$, any term $s \in \mathcal{I}^\sharp(f)$, and any nodes $n_1, \ldots, n_k, n$, represents the decision to define $f^T(n_1, \ldots, n_k)$ as $\langle n, s\rangle$, modulo the substitutions of the template free variables in s; and a variable $b_{R(n_1,\ldots,n_k):\,\psi}$ for any relation $R \in \Sigma$, any formula $\psi \in \mathcal{I}^\sharp(R)$, and any nodes $n_1, \ldots, n_k$, represents the decision to define $R^T(n_1, \ldots, n_k)$ as ψ, modulo the substitutions of the template free variables appearing in ψ.

To encode the possible substitutions of template free variables, allowing different substitutions in different contexts, the following LIA auxiliary variables are used: a variable d_n for any summary node $n \in \mathcal{D}$, and any template free variable d appearing in any bound formula $\psi \in \mathcal{B}^\sharp$; a variable $d_{f(n_1,\ldots,n_k)}$ for any function $f \in \Sigma$, nodes $n_1, \ldots, n_k \in \mathcal{D}$, and any template free variable d appearing in any term $s \in \mathcal{I}^\sharp(f)$; and a variable $d_{R(n_1,\ldots,n_k)}$ for any relation $R \in \Sigma$, nodes $n_1, \ldots, n_k \in \mathcal{D}$, and any template free variable d appearing in any formula $\psi \in \mathcal{I}^\sharp(R)$.

Encoding. Given a FOL formula φ and a symbolic-structure template $S^\sharp$, the "symbolic structure finding" LIA formula is $\varphi_{S^\sharp} \wedge \varphi_{aux}$, where $\varphi_{S^\sharp}$ encodes satisfaction of φ, and φ_{aux} ensures that the assignment to the Boolean auxiliary variables induces a well-defined symbolic structure. We obtain $\varphi_{S^\sharp}$ by transforming φ in a similar way to $\mathrm{trans}^T_{v_T}(\varphi)$ (see Fig. 3). However, here we use the Boolean auxiliary variables as guards and the LIA auxiliary variables as parameters in order to consider different possibilities for bounds, constants, and function and relation applications. This affects the transformation of terms, atomic formulas and quantifiers. For example, for $\mathcal{B}^\sharp = \{x \geq d, \top\}$,

the encoding of quantifiers will use $\left(b_{n:\, x \geq d} \rightarrow x \geq d_n\right) \wedge \left(b_{n:\, \top} \rightarrow \top\right)$ as the bound formula for a summary node n, and simply $x \approx 0$ for regular nodes. Atomic formulas use a conjunction of Boolean auxiliary variables, accumulating the decisions for the constants, functions and relations that appear in them, with the LIA auxiliary variables substituting the template free variables. For example, transforming $R(f(c))$ involves case splits using conjunctions of auxiliary variables for c, f and R. Lastly, if φ has any free variables, the transformation adds a disjunction over all possible symbolic assignments v_S.

We use φ_{aux} to enforce the following constraints: (i) for every constant, function and relation symbol, exactly one Boolean auxiliary variable is true for each combination of argument nodes; (ii) for each node, exactly one of the Boolean bound variables is true; (iii) for every constant, the Boolean auxiliary variables map it to a regular node; and (iv) for every function and every combination of argument nodes, their image according to the Boolean auxiliary variables satisfies the selected bound formula of the image node, according to the Boolean bound variables. There are no constraints for the LIA auxiliary variables. (Details in [Elad et al. 2023b].)

Ultimately, we can check if φ is satisfied by some symbolic structure in $S^\sharp$ (and if so find the satisfying symbolic structure) by checking the satisfiability of $\varphi_{S^\sharp} \wedge \varphi_{aux}$, which can be done using an SMT solver.

Theorem 5.4. *Given a formula φ over Σ and a symbolic-structure template $S^\sharp$, there exists a symbolic structure S in $S^\sharp$ and an explicit assignment v for $\mathcal{E}(S)$ such that $\mathcal{E}(S), v \models \varphi$ iff there exists an assignment v' such that $v' \models_{LIA} \varphi_{S^\sharp} \wedge \varphi_{aux}$. Furthermore, we can efficiently extract S and v from v'.*

6 DECIDABILITY VIA SYMBOLIC STRUCTURES

In this section we introduce the *Ordered Self Cycle (OSC)* fragment, a new decidable fragment of FOL, where satisfiable formulas always have symbolic models. OSC extends the decidable many-sorted EPR fragment. We establish decidability of OSC by proving a novel "small symbolic model property", showing that every satisfiable OSC formula has a (possibly infinite) model representable by a symbolic structure of a bounded size that uses a small subset of LIA. The proof of the "small symbolic model property" identifies a computable template, with which the symbolic search algorithm of Sec. 5 is guaranteed to find a symbolic model for formulas in the fragment. Combined with soundness of the symbolic search, this yields a decision procedure for formulas in OSC.

This section is organized as follows: we start by giving necessary background on the EPR fragment, followed by a formal definition of the OSC fragment; we then continue with the proof of decidability which is done by first considering single-sort formulas (Sec. 6.1), and then generalizing to the many-sorted case (Sec. 6.2).

EPR. The Effectively PRopositional (EPR) fragment of many-sorted FOL consists of formulas whose *quantifier-alternation graph* is acyclic. The quantifier-alternation graph of a formula φ is a directed graph (V, E), where the set of vertices V is the set of sorts and the set of edges E includes an edge from $\boldsymbol{s}_1$ to $\boldsymbol{s}_2$ whenever (i) φ includes a function symbol where $\boldsymbol{s}_1$ is the sort of one of the arguments and $\boldsymbol{s}_2$ is the range sort, or (ii) φ includes an $\exists y\colon \boldsymbol{s}_2$ quantifier in the scope of a $\forall x\colon \boldsymbol{s}_1$ quantifier (assuming φ is in negation normal form; otherwise the definition is extended according to the standard translation to negation normal form).

Acyclicity of the quantifier-alternation graph ensures that after Skolemization, there are finitely many ground terms. That is, the Herbrand universe, as used in the proof of Herbrand's theorem, is finite.[5] This leads to a small model property: every satisfiable EPR formula has a model whose

[5]Herbrand's theorem considers FOL without equality, however it extends to equality through its axiomatization, leading to the same bounds on domain sizes.

domain size is bounded by the number of ground terms of its Skolemized version, which also makes satisfiability checking decidable. Moreover, the number of ground terms can be effectively computed.

The OSC fragment. OSC extends the EPR fragment by allowing a designated sort, $\boldsymbol{s}^\infty$, thought of as an "infinite" sort, which may have self-loops in the quantifier-alternation graph and whose domain does not have a finite model property. That is, some satisfiable OSC formulas are only satisfied by models where the domain of $\boldsymbol{s}^\infty$ is infinite. However, OSC restricts the use of $\boldsymbol{s}^\infty$ such that models can always be represented using symbolic structures from a restricted family, essentially by restricting the use of $\boldsymbol{s}^\infty$ to be monadic except for one (optional) binary relation $\prec$ that is axiomatized to be a strict linear order over $\boldsymbol{s}^\infty$. Formally, an OSC formula is $A_\prec \wedge \psi$,[6] where:

(1) $A_\prec$ encodes the axioms of a strict linear order for $\prec$ (anti-reflexivity $x \not\prec x$, transitivity $x \prec y \wedge y \prec z \rightarrow x \prec z$, and linearity $x \prec y \vee y \prec x \vee x \approx y$).
(2) ψ uses only one logical variable, x, of sort $\boldsymbol{s}^\infty$. (Variables of other sorts are not restricted.)
(3) In the quantifier alternation graph of ψ the only cycles are self-loops at $\boldsymbol{s}^\infty$, and the only outgoing edges from $\boldsymbol{s}^\infty$ are to $\boldsymbol{s}^\infty$.
(4) Functions and relations other than $\prec$ used in ψ have at most one argument of sort $\boldsymbol{s}^\infty$.
(5) The only non-ground term of sort $\boldsymbol{s}^\infty$ that appears in ψ as an argument to functions is x. (That is, functions whose range sort is $\boldsymbol{s}^\infty$ cannot be nested in non-ground terms.)

For example, the Echo Machine's invariants (Fig. 1) and transition formulas (e.g., Eq. (1)) are in OSC, while the following are not: $\forall x, y\colon \boldsymbol{s}^\infty.\ x \prec y \rightarrow f(x) \prec f(y)$ (violates 2); $\forall x\colon \boldsymbol{s}^\infty.\ \exists y\colon \boldsymbol{s}.\ R(x, y)$ (violates 3); $\forall x\colon \boldsymbol{s}^\infty.\ Q(x, x)$ (violates 4); and $\forall x\colon \boldsymbol{s}^\infty.\ x \prec f(f(x))$ (violates 5).

The $\mathbb{S}$ template family. To establish decidability, we prove that satisfiable OSC formulas have symbolic models, and that all symbolic structures needed to satisfy formulas in OSC can be represented by templates from the following family, denoted $\mathbb{S}$, where: (i) bound formulas for all summary nodes are $\top$, (ii) all function terms are 0 or $x_1 + k$ for some $k \in \mathbb{Z}$, and (iii) all relation formulas are degenerate ($\top$ or $\bot$), except for $\prec$, where they can also be $x_1 < x_2$ or $x_1 \leq x_2$. Furthermore, given an OSC formula, we can compute a sufficient *finite subset* of $\mathbb{S}$, which we can enumerate. Formally:

Theorem 6.1. *There exists a computable function $\mathcal{F}\colon \mathrm{OSC} \rightarrow \mathcal{P}(\mathbb{S})$, such that for every formula $\varphi \in \mathrm{OSC}$, $\mathcal{F}(\varphi)$ is finite, and φ is satisfiable iff there exists a template $S^\sharp \in \mathcal{F}(\varphi)$, and a symbolic structure $S \in S^\sharp$ therein, such that $\mathcal{E}(S) \models \varphi$.*

Corollary 6.2. *Satisfiability is decidable for formulas in OSC.*

We prove Thm. 6.1 in two steps: firstly, we discuss the case where there are no sorts other than $\boldsymbol{s}^\infty$ (Sec. 6.1), and secondly, we generalize our construction to many sorts (Sec. 6.2).

6.1 Step 1: Single Sort Construction

In this first step of the proof, we consider the simpler case of OSC formulas where the only sort appearing in a formula is the infinite sort. Let $\varphi = A_\prec \wedge \psi$ be an OSC formula where the only sort is $\boldsymbol{s}^\infty$. (Note that in the case of a single sort, all function and relation symbols in φ, except for $\prec$, must be unary.) The single-variable requirement ensures that ψ can be Skolemized into an equi-satisfiable formula ψ' by introducing a Skolem constant for every existential quantifier (in negation normal form), even if it is in the scope of a universal quantifier.

[6] If ψ does not include $\boldsymbol{s}^\infty$ and/or $\prec$, then $A_\prec$ may be omitted. Still, in that case, any model of ψ can be extended to a model of $A_\prec$. Hence, adding $A_\prec$ does not restrict the set of models. For simplicity of the presentation, we therefore assume that $\prec$ and its axiomatization are always included in OSC formulas.

Let G denote the set of ground terms, ℓ the number of function symbols, and m the number of unary relations that appear in ψ'. We show that if φ is satisfiable, there exists a template $S^\sharp \in \mathbb{S}$ and a symbolic structure $S \in S^\sharp$ therein, such that $\mathcal{E}(S) \models \varphi$. Further, $S^\sharp$ (and S) has at most $|G|$ regular nodes and $b(\ell+1) \cdot 2^{m(\ell+1)} \cdot (|G|+1)^{\ell+1}$ summary nodes, where $b(n)$ is the n'th ordered Bell number counting the number of weak orderings on a set of n elements. In addition, all function terms are 0 or $x_1 + k$ for some $-\ell \le k \le \ell$. The bounds on the number of nodes and on k restrict the set of relevant templates in $\mathbb{S}$ to a finite subset, which defines $\mathcal{F}(\varphi)$ in Thm. 6.1.

Skolemization ensures that the models of $\varphi = A_\prec \wedge \psi$ are the same as the models of $A_\prec \wedge \psi'$ when the latter are projected onto the original vocabulary. It therefore suffices to reason about models of $\varphi' = A_\prec \wedge \psi'$. Let $M = \left(\mathcal{D}^M, \mathcal{I}^M\right)$ be a model for φ'. We construct a symbolic structure $S = \left(\mathcal{D}^S, \mathcal{B}^S, \mathcal{I}^S\right)$ such that $\mathcal{E}(S) \models \varphi'$ and therefore $\mathcal{E}(S) \models \varphi$. (Note that $\mathcal{E}(S)$ need not be isomorphic to M.)

Domain. To define the domain of S, we first partition the elements of $\mathcal{D}^M$ to equivalence classes according to whether or not they satisfy the following atoms in M:

($\mathcal{D}$-1) *Element atoms*: these consist of *order* element atoms, of the form $x \bowtie g$ where $\bowtie \in \{\prec, \approx, \succ\}$[7] and $g \in G$; and *relation* element atoms, of the form $R(x)$, where R is a unary relation symbol.

($\mathcal{D}$-2) *Image atoms*: these consist of *order* image atoms, of the form $f(x) \bowtie g$; and *relation* image atoms, of the form $R(f(x))$, where f is a unary function symbol.

($\mathcal{D}$-3) *Mixed atoms*: $t \bowtie t'$, where $t, t' \in \{x\} \cup \{f(x) \mid f \text{ is a unary function symbol}\}$.

Each class is labeled with the set of atoms satisfied by its elements. We introduce a node in the domain of the symbolic structure, $\mathcal{D}^S$, for each class in the partition. Classes labeled by an atom of the form $x \approx g$ for some ground term g are regular nodes (i.e., with bound formula $x \approx 0$), and the rest are summary nodes with bound formula $\top$. Note that for every ground term $g \in G$, exactly one node is labeled $x \approx g$. This means that each ground term corresponds to a single regular node (a regular node may correspond to multiple ground terms that are equal in M). In particular, the number of regular nodes is bounded by $|G|$.

The number of summary nodes is bounded by the number of feasible combinations of atoms: combinations of mixed atoms are bounded by $b(\ell+1)$, combinations of relation (element and image) atoms are bounded by $2^{m(\ell+1)}$, and combinations of order (element and image) atoms are bounded by $(|G|+1)^{\ell+1}$ (see discussion of *segments* delimited by ground terms below).

For each element $e \in \mathcal{D}^M$ we use $\tau(e)$ for the node in $\mathcal{D}^S$ that corresponds to its class, and for each node $n \in \mathcal{D}^S$ we use $E(n) = \left\{e \in \mathcal{D}^M \mid \tau(e) = n\right\}$ for the set of $\mathcal{D}^M$ elements in its class. (For regular nodes, $E(n)$ is a singleton.) We assume an arbitrary strict linear order on the nodes, denoted $\ll$, which we use as a tiebreaker when defining the interpretation of $\prec$ and the function symbols.

Note that the partitioning takes into account all the atoms that may appear as non-ground atomic formulas in ψ'. Thus, grouping elements to classes according to atoms and defining a node for each class allows us to define a symbolic structure where the explicit elements of n in $\mathcal{E}(S)$ satisfy exactly the same atomic formulas as $E(n)$ in M. This, in turn, ensures that $\mathcal{E}(S)$ satisfies φ'.

Interpretation of constants. For a constant c, we define c^S to be $c^S = \langle n, 0\rangle$, where n is the regular node labeled $x \approx c$. Note that $n = \tau\left(c^M\right)$.

Interpretation of unary relations. Unary relations are determined to be either $\top$ or $\bot$ for each node based on the relation element atoms that label it. Namely, $R^S(n)$ is $\top$ if $R(x)$ labels n and $\bot$ otherwise, for every unary relation R and node n.

Interpretation of $\prec$. Defining $\prec^S$ is not as straightforward, as it is not fully determined by the labeling. Since each ground term in G has a corresponding regular node, the labeling by order

[7] We write $x \succ g$ for $g \prec x$.

element atoms essentially dictates an order on all the ground terms (and accordingly the regular nodes), and places each summary node in a *segment* delimited by ground terms. For example, if there are 3 ground terms g_1, g_2, g_3 and the labeling dictates $g_1 \prec g_2 \approx g_3$, then one regular node corresponds to g_1, another to $g_2 \approx g_3$, and each summary node is either smaller than g_1, between g_1 and $g_2 \approx g_3$, or greater than $g_2 \approx g_3$. Therefore, the only remaining question is how to order summary nodes that are in the same segment. We use this observation to distinguish between the following cases in the definition of $n \prec^S n'$:

(O-1) If either of n, n' is a regular node, then it corresponds to a ground term g and $n \prec^S n'$ is determined to be either $\top$ or $\bot$ by the order induced by the labeling. For example, if n is labeled $x \approx g$ and n' is labeled $x \succ g$ then $n \prec^S n'$ is $\top$.

(O-2) If both n and n' are summary nodes that belong to different segments, $n \prec^S n'$ is determined by the labeling via transitivity. That is, if there exists some ground term g such that n is labeled $x \prec g$ and n' is labeled $x \succ g$ (or vice versa), then $n \prec^S n'$ is $\top$ (respectively, $\bot$).

(O-3) For summary nodes n and n' that are in the same segment, we use the standard $<$ order over the integers to determine the order between $\langle n, z\rangle$ and $\langle n', z'\rangle$, and use $\ll$ as a tiebreaker when $z = z'$. Formally, we define $n \prec^S n'$ to be $x_1 \le x_2$ if $n \ll n'$ and $x_1 < x_2$ otherwise.

(O-3) means that within each segment that includes summary nodes, the ordering of the (infinitely many) elements in $\mathcal{E}(S)$ always alternates between elements from different summary nodes. For example, if n_1 and n_2 are two summary nodes in the same segment with $n_1 \ll n_2$, then the ordering in $\mathcal{E}(S)$ is $\cdots \prec^{\mathcal{E}(S)} \langle n_1, -1\rangle \prec^{\mathcal{E}(S)} \langle n_2, -1\rangle \prec^{\mathcal{E}(S)} \langle n_1, 0\rangle \prec^{\mathcal{E}(S)} \langle n_2, 0\rangle \prec^{\mathcal{E}(S)} \langle n_1, 1\rangle \prec^{\mathcal{E}(S)} \langle n_2, 1\rangle \prec^{\mathcal{E}(S)} \cdots$. For an illustration of $\prec^{\mathcal{E}(S)}$ see [Elad et al. 2023b]. We note that this ordering is independent of M. That is, the ordering would be alternating as above even if $\forall e \in E(n_1), e' \in E(n_2).\, e \prec^M e'$.

Interpretation of unary functions. The most intricate step in our construction is showing that it is possible to define an interpretation of the functions that is consistent with the labeling. For a unary function symbol f, we define the node in the image of f^S according to the image atoms of f and the corresponding term according to the mixed atoms.

Specifically, for a node n labeled with a set L of image atoms for f, let n' be a node labeled with a set L' of element atoms, such that (i) $\{a[f(x)/x] \mid a \in L'\} = L$ (i.e., the element atoms of n' agree with the f-image atoms of n), and (ii) if $x \approx f(x)$ labels n, then $n' = n$; otherwise, n' is the least according to $\ll$ among all nodes that satisfy (i).[8][9] Furthermore, to define the term in $f^S(n)$ while ensuring consistency with the mixed atoms that label n (e.g., $f(x) \prec g(x)$, $f(x) \approx g(x)$), we observe that the mixed atoms partition $S = \{x\} \cup \{h(x) \mid h \text{ is a unary function symbol}\}$ into linearly ordered classes of "equal" terms. This order is isomorphic to $\{0, \ldots, N\}$ for some natural number $N < |S|$, and we let μ be the mapping that assigns to each $t \in S$ the number in $\{0, \ldots, N\}$ isomorphic to its class. For example, if $S = \{x, g(x), h(x)\}$ and n is labeled with $g(x) \prec x$ and $h(x) \approx x$ then $\mu(g(x)) = 0$, and $\mu(x) = \mu(h(x)) = 1$. Finally, we let $k = \mu(f(x)) - \mu(x)$ be the (possibly negative) "distance" between the classes of $f(x)$ and x according to the aforementioned order, and define $f^S(n)$ using k as follows:

$$f^S(n) = \begin{cases} \langle n', 0\rangle & \text{if either } n \text{ or } n' \text{ (or both) is a regular node} \quad (\mathcal{F}\text{-1}) \\ \langle n', x_1 + k\rangle, & \text{if both } n, n' \text{ are summary nodes} \quad (\mathcal{F}\text{-2}) \end{cases}$$

The use of k ensures function interpretations are consistent with n's mixed atoms. Specifically, if $x \approx f(x)$ labels n then $n' = n$ and $k = 0$, and if $g(x) \prec h(x)$ labels n then the k used for g is smaller than the k used for h, which is crucial when $g^S(n)$ and $h^S(n)$ are in the same segment.

[8] Such a node must exist (consider $\tau(f^M(e))$ for some $e \in E(n)$).

[9] We use the minimal according to $\ll$ only as a way to get a deterministic choice, since the choices made for different functions should be consistent (e.g., if n is labeled $f(x) \approx g(x)$). This is not related to the role of $\ll$ as a tiebreaker for $\prec^S$.

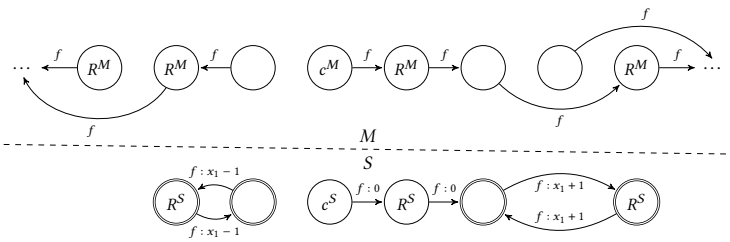


Fig. 5. An infinite model M and its constructed symbolic structure S. The interpretation of $\prec$ orders elements from left to right; adjacent summary nodes are ordered alternatingly. Bound formulas for summary nodes are $\top$.

The following lemma summarizes the correctness of the construction (see proof in [Elad et al. 2023b]).

Lemma 6.3. *For any OSC formula φ where the only sort is $\boldsymbol{s}^\infty$, and any model thereof, if S is the symbolic structure obtained by the above construction, then $\mathcal{E}(S) \models \varphi$.*

Example 6.4. Consider the model M for $\varphi = A_\prec \wedge \neg R(c) \wedge R(f(c)) \wedge \forall x.(R(x) \leftrightarrow \neg R(f(x))) \wedge (c \preccurlyeq x \leftrightarrow x \prec f(x)) \wedge x \not\approx f(x)$ depicted in Fig. 5 (top). The construction above yields the symbolic structure S shown in Fig. 5 (bottom). The formula has two ground terms, c and $f(c)$, so S has two regular nodes. As for summary nodes, observe that in M elements fall into two segments (M has no elements "between" c and $f(c)$); moreover, the elements in each segment fall into one of two equivalence classes, yielding a total of four summary nodes in S.

Note that $\mathcal{E}(S)$ and M are not isomorphic. In particular, in M there are two elements, adjacent according to $\prec^M$, that both satisfy $R(x)$ (e.g., the two top-left elements drawn in Fig. 5), whereas S completely regularizes the pattern of alternating $R(x)$ and $\neg R(x)$ elements.

Remark 6.5. *The restriction OSC imposes on nesting of $\boldsymbol{s}^\infty$ functions is crucial for the construction, but it is only essential for mixed atoms, where it disallows atoms like $f(f(x)) \prec x$. Such atoms would invalidate the construction that defines $f^S(n)$ solely according to the labeling of n. Note that an atom like $R(f(f(x)))$ can be simulated in OSC as $R'(f(x))$ where R' is a fresh unary relation symbol axiomatized by $\forall x.R'(x) \leftrightarrow R(f(x))$. Atoms like $f(f(x)) \prec g$ can be similarly encoded.*

6.2 Step 2: Generalizing to Many Sorts

In the remainder of this section, we generalize the construction of the previous section to many sorts, thus proving decidability of the OSC fragment. The generalization is done by refining the partitions induced by the atoms defined in ($\mathcal{D}$-1), ($\mathcal{D}$-2) and ($\mathcal{D}$-3).

Given $\varphi = A_\prec \wedge \psi$, we first transform ψ to ψ' as in Sec. 6.1. In particular, all existential quantifiers over sort $\boldsymbol{s}^\infty$ are Skolemized by functions from sorts other than $\boldsymbol{s}^\infty$ to $\boldsymbol{s}^\infty$. We observe that for each sort $\boldsymbol{s} \neq \boldsymbol{s}^\infty$ only finitely many ground terms can be constructed, since the quantifier-alternation graph of ψ' is acyclic except for self-loops at $\boldsymbol{s}^\infty$ and $\boldsymbol{s}^\infty$ has no outgoing edges to other sorts. Hence, given a model $M \models \varphi' = A_\prec \wedge \psi'$, we can apply Herbrand's construction (adapted to handle equality) to obtain a model $M' \models \varphi'$ where the domain of each sort $\boldsymbol{s} \neq \boldsymbol{s}^\infty$ is bounded by the number of ground terms (as in many-sorted EPR). Each element in these domains will become a regular node in the symbolic structure S, setting bounds on the domain sizes of all sorts $\boldsymbol{s} \neq \boldsymbol{s}^\infty$ in S. The interpretation of all constant, relation and function symbols that do not include $\boldsymbol{s}^\infty$ in S will be exactly as in M' (using only $\top$ and $\bot$ for relation formulas and 0 for function terms).

Next, we address the elements of sort $\boldsymbol{s}^\infty$ in M'. These are grouped into finitely many summary and regular nodes in S by a variant of the construction from Sec. 6.1, where we use the ground terms of the other sorts to refine the partition of the elements of $\boldsymbol{s}^\infty$. Specifically:

(1) We treat all (finitely many) instantiations of multi-ary functions with (one) argument and range sort s^∞ on the ground terms of sorts $s \neq s^\infty$ as unary functions from s^∞ to s^∞, and extend the sets of image- and mixed atoms to refer to them. For example, given $f\colon s^\infty \times s \to s^\infty$, we add the atoms $x \prec f(x,g)$, $x \approx f(x,g)$, and $x \succ f(x,g)$ as mixed atoms for each ground term g of sort s.

(2) We treat all (finitely many) instantiations of functions with range s^∞ and no argument of sort s^∞ on the ground terms of the other sorts as additional ground terms of sort s^∞, and extend the set of order (element and image) atoms to refer to them. For example, given $h\colon s' \to s^\infty$ and f as before, we add the atoms $x \bowtie h(g')$ and $f(x,g) \bowtie h(g')$ for all ground terms g', g of sorts s', s, resp.

(3) We treat all (finitely many) instantiations of relation symbols other than $\prec$ that include s^∞ on the ground terms of the other sorts as additional unary relation symbols of sort s^∞, and add the corresponding relation (element and image) atoms. For example, given $R\colon s^\infty \times s'$, we add the atoms $R(x,g')$ and $R(f(x,g),g')$ for all ground terms g', g of sorts s', s, resp.

The interpretation of the constant, relation and function symbols that include s^∞ in S is defined similarly to Sec. 6.1. Hence, we obtain a symbolic structure S, such that $\mathcal{E}(S) \models \varphi$; the domain of each sort $s \neq s^\infty$ consists only of regular nodes, whose number is bounded by the number of ground terms of the corresponding sort; and the domain of s^∞ consists of summary and regular nodes, whose number is bounded by the equivalence classes of the refined partition. The function terms and relation formulas used in S are restricted as in Sec. 6.1. Thus, we have a computable finite subset of $\mathbb{S}$ which includes a template $S^\sharp$ such that $S \in S^\sharp$. Relating back to *Thm.* 6.1, for any formula φ, we define $\mathcal{F}(\varphi)$ to be the finite subset of $\mathbb{S}$ that meets the restrictions above.

7 IMPLEMENTATION AND EVALUATION

We implemented the ideas presented in this paper in a tool named FEST (Find and Evaluate Symbolic structures via Templates).[10] FEST allows the user both to specify symbolic structures manually and model-check them against FOL formulas, and to automatically find a symbolic model for a given formula. The symbolic model finding procedure uses automatically-computed "heuristic" templates, or a user-provided template. FEST is implemented in Python, takes input in SMTLIB syntax [Barrett et al. 2017], and uses cvc5 [Barbosa et al. 2022] as the backend for checking LIA formulas (per Thm. 4.7 and 5.4).

We evaluated FEST's symbolic model finding capabilities with the heuristic templates on a host of examples with very positive results — FEST found symbolic models for all of the examples, and many of the symbolic models are captured by the template family $\mathbb{S}$ of OSC. We compare against the SMT solvers cvc5 and Z3 [de Moura and Bjørner 2008], and the resolution-based theorem prover Vampire [Riazanov and Voronkov 1999], all of which diverge when run on these examples.

$$\mathcal{B}^\sharp_{\mathbb{H}} = \{\top, x \geq d, x \leq d\} \qquad \mathcal{I}^\sharp_{\mathbb{H}}(R) = \left\{ \begin{array}{c} \top, \bot, x_1 < x_2, x_1 \leq x_2, x_1 \geq x_2, x_1 > x_2, \\ x_1 \approx 0, x_1 \approx x_2, x_1 \approx x_2 + 1, x_1 \approx x_2 - 1, \\ x_1 \leq x_2 < x_3, x_3 \leq x_2 < x_1, x_1 \approx x_2 \approx x_3 \end{array} \right\}$$
$$\mathcal{I}^\sharp_{\mathbb{H}}(f) = \{0, x_1, x_1 + 1, x_1 - 1\}$$

Fig. 6. The heuristic template family $\mathbb{H}$, parameterized by domain $\mathcal{D}$.

Heuristic templates. FEST uses the heuristic template family $\mathbb{H}$ defined in Fig. 6, and a domain enumeration strategy for the number of regular and summary nodes. The design of $\mathbb{H}$ was motivated by the $\mathbb{S}$ template family of OSC, extended[11] to support functions and relations of greater arity than what the syntactic restrictions of OSC allow. The $\mathbb{H}$ family uses bound formulas with template free variables ($x \geq d, x \leq d$) to represent half-space summary nodes, but avoids them in function terms and relation formulas since they lead to harder LIA queries for the solver, and did not provide

[10] We plan to open-source the implementation once the paper is published.

[11] Templates for OSC may include function terms $x \pm k$ where $1 < k \leq \ell$ (see Sec. 6.1). However, in our benchmarks $\ell = 1$.

a benefit for the examples we considered. The enumeration strategy prioritizes templates with homogeneous domain sizes and smaller total domain size.

Optimizations. FEST implements the translations to LIA developed in Sec. 4 and 5, with several optimizations. First, when searching for a satisfying symbolic structure for a given formula, the interpretation of one constant from each sort is arbitrarily fixed to the first regular node of that sort, under an arbitrary node ordering. This symmetry breaking reduces the search space but does not hurt completeness since any symbolic structure is isomorphic to one of the symbolic structures considered. Second, since $\prec$ is assumed to be a strict linear order, we have that for $x \neq y$, $x \prec y$ iff $y \not\prec x$. Thus, for every two nodes we define auxiliary variables for $\prec$ only for one of the two possible permutations (and derive the other one). Third, for the interpretation of functions and relations, we consider a term or formula with a symbolic-structure free variable x_i only for arguments $n_1, \ldots, n_k$ where n_i is a summary node. For example, for a unary function f the terms $x_1, x_1 + 1, x_1 - 1$ will only be considered when the argument is a summary node; for regular nodes the term 0 will always be used. This is in line with what OSC guarantees in symbolic models, and indeed works for examples both in and outside of OSC.

Evaluation. We evaluated FEST in comparison to Z3, cvc5 and Vampire. Current solvers do not detect infinite models, and prior work on SMT-based verification circumvented them in various ways. For example, [Feldman et al. 2017] avoids infinite counter-models by manually adding axioms, and [Goel and Sakallah 2021b] checks only bounded instances and compares inferred invariants to hand-written ones. As there is no standard benchmark suite for infinite models, we curated a set of 21 examples of distributed protocols and programs manipulating linked lists, based on prior work where FOL is used as an abstraction and infinite models may arise. We run the tools on these examples using a Macbook Pro with an Apple M1 Pro CPU and 32 GB RAM, with Z3 version 4.12.2, cvc5 version 1.0.8, and Vampire version 4.5.1. The results are summarized in Tab. 2. For each example, the table indicates if the example is in OSC, the number of regular and summary nodes in the infinite sort, the numbers of nodes in other sorts, and the run time for finding the symbolic model. Running time was averaged across 10 runs.

FEST is able to effectively find symbolic models for all 21 examples. In contrast, Z3, cvc5 and Vampire are unable to conclude that these examples are satisfiable, and consistently time out after 10 minutes or return "unknown" on all of these examples. (We do not expect the results to be different for any higher timeout threshold.) To demonstrate the general usefulness of symbolic structures for representing and finding infinite models we also include an anecdotal example where we find a non-standard model of non-commutative addition. We briefly describe each example in the sequel; the formal modeling and depictions of some of the symbolic structures are provided in [Elad et al. 2023b].

Echo Machine. The running example from Sec. 2, inspired by the Paxos protocol.

Paxos protocol and variants. These are distributed consensus protocols for agreeing on a value among sets of "acceptors" and "values" of unknown size. Our FOL modeling follows [Goel and Sakallah 2021b]. We present 5 variants of Paxos: abstract Paxos ("Voting" protocol [Lamport 2019]), simple and implicit Paxos [Goel and Sakallah 2021b], regular Paxos [Lamport 1998], and flexible Paxos, which considers a weaker quorum intersection assumption [Howard et al. 2016]. The infinite counter-models resemble that of the Echo Machine, where rounds play a similar role.

Line Leader. A simple distributed protocol to elect a leader for a set of nodes ordered in a line.

Ring Leader. This example models a leader election protocol for a set of nodes ordered in a ring topology [Chang and Roberts 1979], and tries to prove a simplified form of termination following [Feldman et al. 2017; Padon et al. 2016]. The infinite counterexample essentially represents

Table 2. Evaluation results for FEST. We report the mean and standard deviation of run time across 10 runs with different random seeds. "-" denotes timeout after 10 minutes.

Example	In OSC	s^{∞} Size (reg./sum.)	Others' Sizes	Time (s)
Echo Machine	Yes	1/1	2	0.18 ± 0.01
Voting protocol	No	2/1	3, 2, 2	45.28 ± 6.95
Simple Paxos	No	2/1	2, 2, 2	12.72 ± 0.14
Implicit Paxos	No	2/1	3, 2, 2	133.09 ± 41.43
Paxos	No	2/1	3, 3, 2	79.69 ± 22.22
Flexible Paxos	No	2/1	3, 3, 1, 1	27.51 ± 0.20
Ring Leader	No	0/1	n/a	0.08 ± 0.00
Line Leader	No	2/1	n/a	1.59 ± 0.08

Example	In OSC	s^{∞} Size (reg./sum.)	Time (s)
List length †	No	1/1	0.09 ± 0.00
List segment (const)	No	1/1	0.17 ± 0.01
List segment (var)	No	1/1	0.25 ± 0.01
List segment (order)	No	2/1	0.97 ± 0.04
List segment (rev)	No	1/1	0.54 ± 0.07
Doubly-linked list	No	1/1	0.16 ± 0.01
Doubly-linked length	No	1/1	0.16 ± 0.01
Doubly-linked segment	No	1/1	0.35 ± 0.02
Reverse list	No	2/1	0.86 ± 0.02
Sorted list length †	No	1/1	0.09 ± 0.00
Sorted list	No	1/1	0.10 ± 0.00
Sorted list segment	No	1/1	0.20 ± 0.01
Sorted list max	Yes	1/1	0.06 ± 0.00

an infinite ring. While the partial models presented in [Feldman et al. 2017] suggest to the user that an infinite counterexample exists, FEST is able to find one explicitly.

Linked Lists. The right-hand side of Tab. 2 contains examples of verification of linked lists properties (e.g., that reversing a list does not change the set of elements within it). The first 12 examples consist of all of the linked lists examples from [Löding et al. 2018]. There, linked lists are defined inductively, and the least-fixed-point semantics is approximated in FOL by manually adding induction axioms. We remove the added axioms, and show that indeed infinite counter-models arise. For two of the examples which mix LIA and uninterpreted FOL together (example 1 and 10, marked by †), we model the LIA sort as an uninterpreted sort with a single, unbounded summary node. This hints at the future possibility of combining symbolic structures with theories. The last example ("Sorted list max") was inspired by [Löding et al. 2018], but proves a different property of sorted linked lists: that the last element is larger than the first.

Axiomatic Arithmetic. For the standard axiomatization of Presburger arithmetic with successor and addition ($s(x) \neq 0$, $s(x){=}s(y) \rightarrow x{=}y$, $x + 0 = x$, $x + s(y) = s(x + y)$), it is well known that proving commutativity of addition (by induction) requires auxiliary lemmas. Interestingly, a counterexample demonstrating this can be represented by a symbolic structure. For this example, the user must provide two additional function terms $x_1 + x_2, x_1 + x_2 + 1$. Using these, FEST finds a symbolic structure that has *two* summary nodes in about 5 seconds (see [Elad et al. 2023b]).

8 RELATED WORK

Reasoning over specific infinite structures. Several seminal works [Büchi 1990; Büchi and Landweber 1969; Rabin 1969] consider monadic second-order logic (MSO) over the specific (infinite) structures of lines (S1S) and trees (S2S). These works use finite state automata to represent infinite models. Linear Temporal Logic (LTL) [Gerth et al. 1995; Rozier and Vardi 2007; Vardi 1995] also considers verification in the context of the specific infinite structure of the naturals. [Klin and Szynwelski 2016] develops a verification framework over the naturals and the rationals. In contrast, we aim to discover infinite counter-models for VCs that are meant to be valid over all structures.

Representing infinite and unbounded structures. [Bárány et al. 2011; Blumensath and Grädel 2004; Khoussainov and Nerode 1994] introduce automatic structures, which represent infinite structures using automata and regular languages. However, they do not automatically find a satisfying structure for a given formula, nor identify a decidable fragment where representable satisfying structures are guaranteed to exist. The finite representation of infinite terms of co-datatypes introduced in [Reynolds and Blanchette 2017] resemble summary nodes. However, summary nodes represent

infinitely many different elements while an infinite term represents a single element. Symbolic structures are inspired by TVLA [Lev-Ami and Sagiv 2000; Sagiv et al. 1999], but whereas in TVLA summary nodes represent an unbounded *finite* number of elements, we use summary nodes to represent *infinitely* many elements. The use of LIA terms and formulas in symbolic structures resembles symbolic automata [D'Antoni and Veanes 2021], where transitions carry predicates over rich theories. Our construction of a symbolic structure in Sec. 6 bears some resemblances to predicate abstraction [Graf and Saïdi 1997]. However, here it is only part of the decidability proof.

Constructing infinite models. Existing SMT solvers [Barbosa et al. 2022; de Moura and Bjørner 2008; Dutertre and De Moura 2006] and resolution-based theorem provers [Riazanov and Voronkov 1999; Weidenbach et al. 2009] cannot construct infinite models. Resolution can sometimes terminate when the clause set is saturated even if the counter-model is infinite. [Horbach and Sofronie-Stokkermans 2013] uses constrained clauses in order to represent possibly infinite sets of saturated clauses. [Bachmair and Ganzinger 2001; Peltier 2003] present theoretical constructions of counter-models from saturated sets of clauses, but these constructions are not implemented in existing tools. Recently, [Lynch and Miner 2023] showed that under certain restrictions, trigger-based quantifier instantiation can be complete and used to prove satisfiability without an explicit construction of the model. The model evolution calculus [Baumgartner et al. 2006; Baumgartner and Tinelli 2003] constructs Herbrand models based on a generalization of DPLL, where the interpretation of relations is implicitly defined by a finite set of possibly non-ground literals. In contrast, our approach represents the domain and the interpretation explicitly, using LIA formulas and terms. Our approach is therefore complementary, and can give the user insight when other tools fail. In the context of higher-order logic and inductive datatypes, [Blanchette and Claessen 2010] describes a way to obtain finite *fragments* of nonstandard, infinite models, via a specific first-order abstraction. Our approach finds entire infinite models for arbitrary first-order formulas, not tied to a specific abstraction of inductive definitions, and yields a decidable fragment. Recently, [Parsert et al. 2023] experimented with using syntax-guided synthesis [Alur et al. 2013] in order to construct infinite models. Our use of templates resembles syntax-guided synthesis, but the partition of the domain via summary nodes allows us to define more complex structures in a simpler way (using simpler LIA terms).

Decidable fragments of FOL in Verification. Existing verification approaches based on FOL either restrict verification conditions to have finite models [Ge and de Moura 2009; Padon et al. 2017; Taube et al. 2018], or use finite partial models that do not fully satisfy the formula but satisfy some quantifier instantiations [Feldman et al. 2017; Neider et al. 2018; Preiner et al. 2017]. Complementary to these approaches, we aim to explicitly find certain infinite models via a finite representation. Coherent uninterpreted programs [Mathur et al. 2019, 2020] are another FOL-based approach where programs are restricted such that safety verification is decidable. In contrast, we target a richer formalism (in fact, Turing complete), focusing on VC checking rather than safety verification. There is a rich literature on classical decidable fragments of FOL [Ackermann 1928; Börger et al. 2001; Gödel 1932; Gurevich 1976; Lewis 1980; Löb 1967; Mortimer 1975; Shelah 1977; Voigt 2019]. Most of these fragments have a finite model property, and many cannot axiomatize a linear order; to the best of our knowledge, OSC is not subsumed by any known decidable fragment. Also related are decidability results for finite satisfiability (e.g., [Danielski and Kieronski 2019; Shelah 1977]); these fragments are similarly incomparable to OSC. Moreover, symbolic structures are useful beyond OSC and extend the applicability of SMT solvers, which use the standard semantics of FOL.

9 CONCLUSION

This paper introduced symbolic structures as an efficient way to represent and discover infinite models of FOL formulas. We presented a symbolic search algorithm to efficiently traverse vast (possibly infinite) families of symbolic structures, encoded as symbolic-structure templates, and designed a robust heuristic to construct such templates. Naturally, by restricting symbolic structures via a template to use a finite subset of LIA, we are severely limiting the kind of symbolic structures that can be found. Using empirically-proven templates strikes a balance between efficiency and expressivity. Our evaluation of FEST demonstrates the effectiveness of the technique for finding infinite counter-models. Beyond the empirical evaluation, we also made a theoretical contribution: we identified a new decidable fragment of FOL, which extends EPR, where satisfiable formulas always have a (possibly infinite) model representable by a symbolic model of a bounded size that is captured by the aforementioned heuristic templates. Our technique can also be extended and fine-tuned to different domains using specialized symbolic-structure templates. We consider this a first step, and posit that automatic methods for finding infinite counter-models can have wide uses in verification, in particular for guiding the user (or as part of a CEGAR-style algorithm) towards amending the modeling, e.g., by adding instantiations of induction or other axiom schemes. can use induction to find a refutation [Hajdú et al. 2021; Hozzová et al. 2021; Leino 2012; Reynolds and Kuncak 2015; Schoisswohl and Kovács 2021], these mechanisms are incomplete and require heuristics [Kaufmann et al. 2013; Kaufmann and Moore 1997]. Infinite counter-models obtained by integrating with FEST could provide hints for instantiating the induction scheme.

DATA-AVAILABILITY STATEMENT

An extended version of this paper, with proofs and additional examples can be found at [Elad et al. 2023b]. An artifact for reproducing the results is available at [Elad et al. 2023a], and the latest version of the tool can be obtained at [Elad et al. [n. d.]].

ACKNOWLEDGMENTS

We thank the anonymous reviewers and the artifact evaluation committee for comments which improved the paper. We thank Raz Lotan and Yotam Feldman for insightful discussions and comments. The research leading to these results has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement No [759102-SVIS]). This research was partially supported by the Israeli Science Foundation (ISF) grant No. 2117/23.

REFERENCES

Wilhelm Ackermann. 1928. Über die Erfüllbarkeit gewisser Zählausdrücke. *Math. Ann.* 100, 1 (1928), 638–649. https://doi.org/10.1007/BF01448869

Rajeev Alur, Rastislav Bodík, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. 2013. Syntax-guided synthesis. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*. IEEE, 1–8. https://doi.org/10.1109/FMCAD.2013.6679385

Leo Bachmair and Harald Ganzinger. 2001. Resolution Theorem Proving. In *Handbook of Automated Reasoning*. Elsevier and MIT Press, 19–99.

Thomas Ball, Nikolaj S. Bjørner, Aaron Gember, Shachar Itzhaky, Aleksandr Karbyshev, Mooly Sagiv, Michael Schapira, and Asaf Valadarsky. 2014. VeriCon: towards verifying controller programs in software-defined networks. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, Michael F. P. O'Boyle and Keshav Pingali (Eds.). ACM, 282–293. https://doi.org/10.1145/2594291.2594317

Kshitij Bansal, Andrew Reynolds, Tim King, Clark W. Barrett, and Thomas Wies. 2015. Deciding Local Theory Extensions via E-matching. In *CAV (2) (Lecture Notes in Computer Science, Vol. 9207)*. Springer, 87–105. https://doi.org/10.1007/978-3-319-21668-3_6

Vince Bárány, Erich Grädel, and Sasha Rubin. 2011. Automata-based presentations of infinite structures. In *Finite and Algorithmic Model Theory*. London Mathematical Society Lecture Note Series, Vol. 379. Cambridge University Press, 1–76. https://doi.org/10.1017/CBO9780511974960.002

Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. 2022. cvc5: A Versatile and Industrial-Strength SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13243)*, Dana Fisman and Grigore Rosu (Eds.). Springer, 415–442. https://doi.org/10.1007/978-3-030-99524-9_24

Clark Barrett, Pascal Fontaine, and Cesare Tinelli. 2017. *The SMT-LIB Standard: Version 2.6*. Technical Report. Department of Computer Science, The University of Iowa. Available at `www.SMT-LIB.org`.

Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. 2006. Implementing the Model Evolution Calculus. *Int. J. Artif. Intell. Tools* 15, 1 (2006), 21–52. https://doi.org/10.1142/S0218213006002552

Peter Baumgartner and Cesare Tinelli. 2003. The Model Evolution Calculus. In *CADE (Lecture Notes in Computer Science, Vol. 2741)*. Springer, 350–364. https://doi.org/10.1007/978-3-540-45085-6_32

Jasmin Christian Blanchette and Koen Claessen. 2010. Generating Counterexamples for Structural Inductions by Exploiting Nonstandard Models. In *LPAR (Yogyakarta) (Lecture Notes in Computer Science, Vol. 6397)*. Springer, 127–141. https://doi.org/10.1007/978-3-642-16242-8_10

Achim Blumensath and Erich Grädel. 2004. Finite Presentations of Infinite Structures: Automata and Interpretations. *Theory Comput. Syst.* 37, 6 (2004), 641–674. https://doi.org/10.1007/s00224-004-1133-y

Egon Börger, Erich Grädel, and Yuri Gurevich. 2001. *The classical decision problem*. Springer Science & Business Media.

Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. 2006. What's Decidable About Arrays?. In *VMCAI (Lecture Notes in Computer Science, Vol. 3855)*. Springer, 427–442. https://doi.org/10.1007/11609773_28

J Richard Büchi. 1990. On a decision method in restricted second order arithmetic. In *The collected works of J. Richard Büchi*. Springer, 425–435. https://doi.org/10.1007/978-1-4613-8928-6_23

J. Richard Büchi and Lawrence H. Landweber. 1969. Definability in the Monadic Second-Order Theory of Successor. *J. Symb. Log.* 34, 2 (1969), 166–170. https://doi.org/10.2307/2271090

Ernest Chang and Rosemary Roberts. 1979. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Commun. ACM* 22, 5 (1979), 281–283. https://doi.org/10.1145/359104.359108

David C Cooper. 1972. Theorem proving in arithmetic without multiplication. *Machine intelligence* 7, 91-99 (1972), 300. https://doi.org/10.1007/10930755_5

Daniel Danielski and Emanuel Kieronski. 2019. Finite Satisfiability of Unary Negation Fragment with Transitivity. In *MFCS (LIPIcs, Vol. 138)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 17:1–17:15. https://doi.org/10.4230/LIPIcs.MFCS.2019.17

Loris D'Antoni and Margus Veanes. 2021. Automata modulo theories. *Commun. ACM* 64, 5 (2021), 86–95. https://doi.org/10.1145/3419404

Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2007. Efficient E-Matching for SMT Solvers. In *CADE (Lecture Notes in Computer Science, Vol. 4603)*. Springer, 183–198. https://doi.org/10.1007/978-3-540-73595-3_13

Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 4963)*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

David Detlefs, Greg Nelson, and James B. Saxe. 2005. Simplify: a theorem prover for program checking. *J. ACM* 52, 3 (2005), 365–473. https://doi.org/10.1145/1066100.1066102

Claire Dross, Sylvain Conchon, Johannes Kanig, and Andrei Paskevich. 2016. Adding Decision Procedures to SMT Solvers Using Axioms with Triggers. *J. Autom. Reason.* 56, 4 (2016), 387–457. https://doi.org/10.1007/s10817-015-9352-2

Bruno Dutertre and Leonardo De Moura. 2006. The yices smt solver. *Tool paper at http://yices. csl. sri. com/tool-paper. pdf* 2, 2 (2006), 1–2.

Neta Elad, Oded Padon, and Sharon Shoham. [n. d.]. An Infinite Needle in a Finite Haystack: Finding Infinite Counter-Models in Deductive Verification (Artifact). https://doi.org/10.5281/zenodo.8404103

Neta Elad, Oded Padon, and Sharon Shoham. 2023a. An Infinite Needle in a Finite Haystack: Finding Infinite Counter-Models in Deductive Verification (Artifact). https://doi.org/10.5281/zenodo.10125136

Neta Elad, Oded Padon, and Sharon Shoham. 2023b. An Infinite Needle in a Finite Haystack: Finding Infinite Counter-Models in Deductive Verification. https://doi.org/10.48550/arXiv.2310.16762 arXiv:2310.16762 [cs.PL]

Yotam M. Y. Feldman, Oded Padon, Neil Immerman, Mooly Sagiv, and Sharon Shoham. 2017. Bounded Quantifier Instantiation for Checking Inductive Invariants. In *TACAS (1) (Lecture Notes in Computer Science, Vol. 10205)*. 76–95. https://doi.org/10.1007/978-3-662-54577-5_5

Yeting Ge and Leonardo Mendonça de Moura. 2009. Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5643)*, Ahmed Bouajjani and Oded Maler (Eds.). Springer, 306–320. https://doi.org/10.1007/978-3-642-02658-4_25

Rob Gerth, Doron A. Peled, Moshe Y. Vardi, and Pierre Wolper. 1995. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV (IFIP Conference Proceedings, Vol. 38)*. Chapman & Hall, 3–18. https://doi.org/10.1007/978-0-387-34892-6_1

Kurt Gödel. 1932. Ein Spezialfall des Entscheidungsproblems der theoretischen Logik. *Ergebnisse eines mathematischen Kolloquiums* 2 (1932), 27–28.

Aman Goel and Karem A. Sakallah. 2021a. On Symmetry and Quantification: A New Approach to Verify Distributed Protocols. In *NFM (Lecture Notes in Computer Science, Vol. 12673)*. Springer, 131–150. https://doi.org/10.1007/978-3-030-76384-8_9

Aman Goel and Karem A. Sakallah. 2021b. Towards an Automatic Proof of Lamport's Paxos. In *FMCAD*. IEEE, 112–122. https://doi.org/10.34727/2021/isbn.978-3-85448-046-4_20

Susanne Graf and Hassen Saïdi. 1997. Construction of Abstract State Graphs with PVS. In *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings (Lecture Notes in Computer Science, Vol. 1254)*, Orna Grumberg (Ed.). Springer, 72–83. https://doi.org/10.1007/3-540-63166-6_10

Yuri Gurevich. 1976. The decision problem for standard classes. *The Journal of Symbolic Logic* 41, 2 (1976), 460–464. https://doi.org/10.1017/S0022481200051513

Márton Hajdú, Petra Hozzová, Laura Kovács, and Andrei Voronkov. 2021. Induction with Recursive Definitions in Superposition. In *FMCAD*. IEEE, 1–10. https://doi.org/10.34727/2021/isbn.978-3-85448-046-4_34

Travis Hance, Marijn Heule, Ruben Martins, and Bryan Parno. 2021. Finding Invariants of Distributed Systems: It's a Small (Enough) World After All. In *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021*, James Mickens and Renata Teixeira (Eds.). USENIX Association, 115–131. https://www.usenix.org/conference/nsdi21/presentation/hance

Matthias Horbach and Viorica Sofronie-Stokkermans. 2013. Obtaining Finite Local Theory Axiomatizations via Saturation. In *Frontiers of Combining Systems - 9th International Symposium, FroCoS 2013, Nancy, France, September 18-20, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 8152)*, Pascal Fontaine, Christophe Ringeissen, and Renate A. Schmidt (Eds.). Springer, 198–213. https://doi.org/10.1007/978-3-642-40885-4_14

Heidi Howard, Dahlia Malkhi, and Alexander Spiegelman. 2016. Flexible Paxos: Quorum Intersection Revisited. In *20th International Conference on Principles of Distributed Systems, OPODIS 2016, December 13-16, 2016, Madrid, Spain (LIPIcs, Vol. 70)*, Panagiota Fatourou, Ernesto Jiménez, and Fernando Pedone (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 25:1–25:14. https://doi.org/10.4230/LIPIcs.OPODIS.2016.25

Petra Hozzová, Laura Kovács, and Andrei Voronkov. 2021. Integer Induction in Saturation. In *CADE (Lecture Notes in Computer Science, Vol. 12699)*. Springer, 361–377. https://doi.org/10.1007/978-3-030-79876-5_21

Shachar Itzhaky, Anindya Banerjee, Neil Immerman, Ori Lahav, Aleksandar Nanevski, and Mooly Sagiv. 2014. Modular reasoning about heap paths via effectively propositional formulas. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, Suresh Jagannathan and Peter Sewell (Eds.). ACM, 385–396. https://doi.org/10.1145/2535838.2535854

Shachar Itzhaky, Anindya Banerjee, Neil Immerman, Aleksandar Nanevski, and Mooly Sagiv. 2013. Effectively-Propositional Reasoning about Reachability in Linked Data Structures. In *CAV (Lecture Notes in Computer Science, Vol. 8044)*. Springer, 756–772. https://doi.org/10.1007/978-3-642-39799-8_53

Swen Jacobs. 2009. Incremental Instance Generation in Local Reasoning. In *CAV (Lecture Notes in Computer Science, Vol. 5643)*. Springer, 368–382. https://doi.org/10.1007/978-3-642-02658-4_29

Aleksandr Karbyshev, Nikolaj S. Bjørner, Shachar Itzhaky, Noam Rinetzky, and Sharon Shoham. 2017. Property-Directed Inference of Universal Invariants or Proving Their Absence. *J. ACM* 64, 1 (2017), 7:1–7:33. https://doi.org/10.1145/3022187

Matt Kaufmann, Panagiotis Manolios, and J Strother Moore. 2013. *Computer-aided reasoning: ACL2 case studies*. Vol. 4. Springer Science & Business Media. https://doi.org/10.1007/978-1-4757-3188-0

Matt Kaufmann and J. Strother Moore. 1997. An Industrial Strength Theorem Prover for a Logic Based on Common Lisp. *IEEE Trans. Software Eng.* 23, 4 (1997), 203–213. https://doi.org/10.1109/32.588534

Bakhadyr Khoussainov and Anil Nerode. 1994. Automatic Presentations of Structures. In *Logical and Computational Complexity. Selected Papers. Logic and Computational Complexity, International Workshop LCC '94, Indianapolis, Indiana, USA, 13-16 October 1994 (Lecture Notes in Computer Science, Vol. 960)*, Daniel Leivant (Ed.). Springer, 367–392. https://doi.org/10.1007/3-540-60178-3_93

Bartek Klin and Michal Szynwelski. 2016. SMT Solving for Functional Programming over Infinite Structures. In *MSFP (EPTCS, Vol. 207)*. 57–75. https://doi.org/10.4204/EPTCS.207.3

Jason R. Koenig, Oded Padon, Sharon Shoham, and Alex Aiken. 2022. Inferring Invariants with Quantifier Alternations: Taming the Search Space Explosion. In *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13243)*, Dana Fisman and Grigore Rosu (Eds.). Springer, 338–356. https://doi.org/10.1007/978-3-030-99524-9_18

Leslie Lamport. 1998. The Part-Time Parliament. *ACM Trans. Comput. Syst.* 16, 2 (1998), 133–169. https://doi.org/10.1145/279227.279229

Leslie Lamport. 2019. The Paxos Algorithm - or How to Win a Turing Award. https://lamport.azurewebsites.net/tla/paxos-algorithm.html

K. Rustan M. Leino. 2012. Automating Induction with an SMT Solver. In *VMCAI (Lecture Notes in Computer Science, Vol. 7148)*. Springer, 315–331. https://doi.org/10.1007/978-3-642-27940-9_21

Tal Lev-Ami and Shmuel Sagiv. 2000. TVLA: A System for Implementing Static Analyses. In *SAS (Lecture Notes in Computer Science, Vol. 1824)*. Springer, 280–301. https://doi.org/10.1007/978-3-540-45099-3_15

Harry R. Lewis. 1980. Complexity Results for Classes of Quantificational Formulas. *J. Comput. Syst. Sci.* 21, 3 (1980), 317–353. https://doi.org/10.1016/0022-0000(80)90027-6

Martin Löb. 1967. Decidability of the monadic predicate calculus with unary function symbols. *Journal of Symbolic Logic* 32, 4 (1967), 563.

Christof Löding, P. Madhusudan, and Lucas Peña. 2018. Foundations for natural proofs and quantifier instantiation. *Proc. ACM Program. Lang.* 2, POPL (2018), 10:1–10:30. https://doi.org/10.1145/3158098

Christopher Lynch. 2013. Constructing Bachmair-Ganzinger Models. In *Programming Logics (Lecture Notes in Computer Science, Vol. 7797)*. Springer, 285–301. https://doi.org/10.1007/978-3-642-37651-1_12

Christopher Lynch and Stephen Miner. 2023. Complete Trigger Selection in Satisfiability Modulo First-Order Theories. In *Proceedings of the 21st International Workshop on Satisfiability Modulo Theories (SMT 2023) co-located with the 29th International Conference on Automated Deduction (CADE 2023), Rome, Italy, July, 5-6, 2023 (CEUR Workshop Proceedings, Vol. 3429)*, Stéphane Graham-Lengrand and Mathias Preiner (Eds.). CEUR-WS.org, 18–32. https://doi.org/10.48550/arXiv.2306.09436

Haojun Ma, Aman Goel, Jean-Baptiste Jeannin, Manos Kapritsos, Baris Kasikci, and Karem A. Sakallah. 2019. I4: incremental inference of inductive invariants for verification of distributed protocols. In *SOSP*. ACM, 370–384. https://doi.org/10.1145/3341301.3359651

Umang Mathur, P. Madhusudan, and Mahesh Viswanathan. 2019. Decidable verification of uninterpreted programs. *Proc. ACM Program. Lang.* 3, POPL (2019), 46:1–46:29. https://doi.org/10.1145/3290359

Umang Mathur, P. Madhusudan, and Mahesh Viswanathan. 2020. What's Decidable About Program Verification Modulo Axioms?. In *TACAS (2) (Lecture Notes in Computer Science, Vol. 12079)*. Springer, 158–177. https://doi.org/10.1007/978-3-030-45237-7_10

Kenneth L. McMillan and Oded Padon. 2018. Deductive Verification in Decidable Fragments with Ivy. In *SAS (Lecture Notes in Computer Science, Vol. 11002)*. Springer, 43–55. https://doi.org/10.1007/978-3-319-99725-4_4

Kenneth L. McMillan and Oded Padon. 2020. Ivy: A Multi-modal Verification Tool for Distributed Algorithms. In *CAV (2) (Lecture Notes in Computer Science, Vol. 12225)*. Springer, 190–202. https://doi.org/10.1007/978-3-030-53291-8_12

Michael Mortimer. 1975. On languages with two variables. *Mathematical Logic Quarterly* 21, 1 (1975), 135–140. https://doi.org/10.1002/malq.19750210118

Adithya Murali, Lucas Peña, Eion Blanchard, Christof Löding, and P. Madhusudan. 2022. Model-guided synthesis of inductive lemmas for FOL with least fixpoints. *Proc. ACM Program. Lang.* 6, OOPSLA2 (2022), 1873–1902. https://doi.org/10.1145/3563354

Daniel Neider, Pranav Garg, P. Madhusudan, Shambwaditya Saha, and Daejun Park. 2018. Invariant Synthesis for Incomplete Verification Engines. In *TACAS (1) (Lecture Notes in Computer Science, Vol. 10805)*. Springer, 232–250. https://doi.org/10.1007/978-3-319-89960-2_13

Oded Padon, Giuliano Losa, Mooly Sagiv, and Sharon Shoham. 2017. Paxos made EPR: decidable reasoning about distributed protocols. *Proc. ACM Program. Lang.* 1, OOPSLA (2017), 108:1–108:31. https://doi.org/10.1145/3140568

Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. 2016. Ivy: safety verification by interactive generalization. In *PLDI*. ACM, 614–630. https://doi.org/10.1145/2908080.2908118

Oded Padon, James R. Wilcox, Jason R. Koenig, Kenneth L. McMillan, and Alex Aiken. 2022. Induction duality: primal-dual search for invariants. *Proc. ACM Program. Lang.* 6, POPL (2022), 1–29. https://doi.org/10.1145/3498712

Julian Parsert, Chad E. Brown, Mikolas Janota, and Cezary Kaliszyk. 2023. Experiments on Infinite Model Finding in SMT Solving. In *LPAR 2023: Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Manizales, Colombia, 4-9th June 2023 (EPiC Series in Computing, Vol. 94)*, Ruzica Piskac and Andrei Voronkov

(Eds.). EasyChair, 317–328. https://doi.org/10.29007/slrm

Nicolas Peltier. 2003. Building Infinite Models for Equational Clause Sets: Constructing Non-Ambiguous Formulae. *Log. J. IGPL* 11, 1 (2003), 97–129. https://doi.org/10.1093/jigpal/11.1.97

Mathias Preiner, Aina Niemetz, and Armin Biere. 2017. Counterexample-Guided Model Synthesis. In *TACAS (1) (Lecture Notes in Computer Science, Vol. 10205)*. 264–280. https://doi.org/10.1007/978-3-662-54577-5_15

Michael O Rabin. 1969. Decidability of second-order theories and automata on infinite trees. *Transactions of the american Mathematical Society* 141 (1969), 1–35. https://doi.org/10.2307/1995086

Frank P Ramsey. 1930. On a Problem of Formal Logic. *Procedures of London Mathematical Society* 30 (1930), 264–285. https://doi.org/10.1007/978-0-8176-4842-8_1

Giles Reger, Martin Suda, and Andrei Voronkov. 2016. Finding Finite Models in Multi-sorted First-Order Logic. In *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9710)*, Nadia Creignou and Daniel Le Berre (Eds.). Springer, 323–341. https://doi.org/10.1007/978-3-319-40970-2_20

Andrew Reynolds and Jasmin Christian Blanchette. 2017. A Decision Procedure for (Co)datatypes in SMT Solvers. *J. Autom. Reason.* 58, 3 (2017), 341–362. https://doi.org/10.1007/s10817-016-9372-6

Andrew Reynolds and Viktor Kuncak. 2015. Induction for SMT Solvers. In *VMCAI (Lecture Notes in Computer Science, Vol. 8931)*. Springer, 80–98. https://doi.org/10.1007/978-3-662-46081-8_5

Andrew Reynolds, Cesare Tinelli, Amit Goel, and Sava Krstic. 2013a. Finite Model Finding in SMT. In *CAV (Lecture Notes in Computer Science, Vol. 8044)*. Springer, 640–655. https://doi.org/10.5555/2958031.2958040

Andrew Reynolds, Cesare Tinelli, Amit Goel, Sava Krstic, Morgan Deters, and Clark W. Barrett. 2013b. Quantifier Instantiation Techniques for Finite Model Finding in SMT. In *CADE (Lecture Notes in Computer Science, Vol. 7898)*. Springer, 377–391. https://doi.org/10.1007/978-3-642-38574-2_26

Alexandre Riazanov and Andrei Voronkov. 1999. Vampire. In *CADE (Lecture Notes in Computer Science, Vol. 1632)*. Springer, 292–296. https://doi.org/10.1007/3-540-48660-7_26

Kristin Y. Rozier and Moshe Y. Vardi. 2007. LTL Satisfiability Checking. In *SPIN (Lecture Notes in Computer Science, Vol. 4595)*. Springer, 149–167. https://doi.org/10.1007/s10009-010-0140-3

Shmuel Sagiv, Thomas W. Reps, and Reinhard Wilhelm. 1999. Parametric Shape Analysis via 3-Valued Logic. In *POPL*. ACM, 105–118. https://doi.org/10.1145/514188.514190

Johannes Schoisswohl and Laura Kovács. 2021. Automating Induction by Reflection. In *LFMTP (EPTCS, Vol. 337)*. 39–54. https://doi.org/10.4204/EPTCS.337.4

Saharon Shelah. 1977. Decidability of a portion of the predicate calculus. *Israel Journal of Mathematics* 28, 1 (1977), 32–44. https://doi.org/10.1007/BF02759780

Marcelo Taube, Giuliano Losa, Kenneth L. McMillan, Oded Padon, Mooly Sagiv, Sharon Shoham, James R. Wilcox, and Doug Woos. 2018. Modularity for decidability of deductive verification with applications to distributed systems. In *PLDI*. ACM, 662–677. https://doi.org/10.1145/3192366.3192414

Moshe Y. Vardi. 1995. An Automata-Theoretic Approach to Linear Temporal Logic. In *Banff Higher Order Workshop (Lecture Notes in Computer Science, Vol. 1043)*. Springer, 238–266. https://doi.org/10.1007/3-540-60915-6_6

Niki Vazou, Anish Tondwalkar, Vikraman Choudhury, Ryan G. Scott, Ryan R. Newton, Philip Wadler, and Ranjit Jhala. 2018. Refinement reflection: complete verification with SMT. *Proc. ACM Program. Lang.* 2, POPL (2018), 53:1–53:31. https://doi.org/10.1145/3158141

Marco Voigt. 2019. *Decidable fragments of first-order logic and of first-order linear arithmetic with uninterpreted predicates*. Ph. D. Dissertation. Saarland University, Saarbrücken, Germany.

Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. 2009. SPASS Version 3.5. In *CADE (Lecture Notes in Computer Science, Vol. 5663)*. Springer, 140–145. https://doi.org/10.1007/978-3-642-02959-2_10

Jianan Yao, Runzhou Tao, Ronghui Gu, and Jason Nieh. 2022. DuoAI: Fast, Automated Inference of Inductive Invariants for Verifying Distributed Protocols. In *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022*, Marcos K. Aguilera and Hakim Weatherspoon (Eds.). USENIX Association, 485–501. https://www.usenix.org/conference/osdi22/presentation/yao

Jianan Yao, Runzhou Tao, Ronghui Gu, Jason Nieh, Suman Jana, and Gabriel Ryan. 2021. DistAI: Data-Driven Automated Invariant Learning for Distributed Protocols. In *OSDI*. USENIX Association, 405–421.

Received 2023-07-11; accepted 2023-11-07