Decidability of Inferring Inductive Invariants

Oded Padon
Tel Aviv University, Israel
odedp@mail.tau.ac.il

Neil Immerman
University of Massachusetts, Amherst, USA
immerman@cs.umass.edu

Sharon Shoham
The Academic College of Tel Aviv Yaffo, Israel
sharon.shoham@gmail.com

Aleksandr Karbyshev
Tel Aviv University, Israel
karbyshev@post.tau.ac.il

Mooly Sagiv
Tel Aviv University, Israel
msagiv@post.tau.ac.il

Abstract

Induction is a successful approach for verification of hardware and software systems. A common practice is to model a system using logical formulas, and then use a decision procedure to verify that some logical formula is an inductive safety invariant for the system. A key ingredient in this approach is coming up with the inductive invariant, which is known as invariant inference. This is a major difficulty, and it is often left for humans or addressed by sound but incomplete abstract interpretation. This paper is motivated by the problem of inductive invariants in shape analysis and in distributed protocols.

This paper approaches the general problem of inferring first-order inductive invariants by restricting the language L of candidate invariants. Notice that the problem of invariant inference in a restricted language L differs from the safety problem, since a system may be safe and still not have any inductive invariant in L that proves safety. Clearly, if L is finite (and if testing an inductive invariant is decidable), then inferring invariants in L is decidable. This paper presents some interesting cases when inferring inductive invariants in L is decidable even when L is an infinite language of universal formulas. Decidability is obtained by restricting L and defining a suitable well-quasi-order on the state space. We also present some undecidability results that show that our restrictions are necessary. We further present a framework for systematically constructing infinite languages while keeping the invariant inference problem decidable. We illustrate our approach by showing the decidability of inferring invariants for programs manipulating linked-lists, and for distributed protocols.

Categories and Subject Descriptors D.2.4 [*Software/Program Verification*]: Formal methods; F.3.1 [*Specifying and Verifying and Reasoning about Programs*]: Invariants

Keywords verification, invariant inference, well-quasi-order, effectively propositional logic

POPL'16, January 20–22, 2016, St. Petersburg, FL, USA
© 2016 ACM. 978-1-4503-3549-2/16/01...$15.00
http://dx.doi.org/10.1145/2837614.2837640

1. Introduction

Verifying the safety of infinite-state systems is a central problem for automatic program verification. Given a program which describes a transition relation τ, an assumption *init* on the initial states, and a safety property P, we wish to automatically prove that every state that is τ-reachable from a state satisfying *init*, satisfies P.

One of the most useful techniques for proving safety already advocated by Floyd [17] is using *inductive invariants*. We say that an invariant I is *inductive* for $\langle init, \tau, P\rangle$ if I satisfies the following three conditions: (i) $init \Rightarrow I$ (**initiation**). (ii) $I \Rightarrow P$ (**safety**). (iii) I is closed under τ, i.e., for every execution of τ which starts in a state satisfying I, I also holds after executing τ (**consecution**). It is well known that P holds if and only if there exists such an inductive invariant (in a sufficiently powerful language).

Deductive verification tools such as Dafny [28] accept as input: (i) a program, (ii) safety property and (iii) a candidate inductive invariant I. These tools invoke a theorem prover such as Z3 [15] to check that I is an inductive invariant. This approach raises two technical challenges: (a) coming-up with an inductive invariant, and (b) checking that a given invariant is inductive. Step (b) can be fully automated if the program, the property, and the invariant are expressed in a decidable logic [25, 32]. Many techniques attempt to mechanize step (a) by searching for an inductive invariant. Such tools are only able to infer inductive invariants in a certain language[1], and are hence necessarily incomplete in verifying safety. Their output might be: (i) program is safe (found inductive invariant); or (ii) program is unsafe (found a concrete counterexample); or (iii) don't know or diverge. Since the safety verification problem is undecidable, this incompleteness is expected and therefore mostly accepted by users of such techniques. However, since actual tools search for inductive invariants in a certain language, the underlying decision problem they address is in fact "is there an inductive invariant in a certain language?" A key premise of this work is that this problem is different from the safety problem, and hence might be decidable even in cases where safety verification is not.

This work investigates the decidability of the problem of inferring inductive invariants in a *given* language. Here, the expected outcome is not "safe/unsafe", but rather "inductive invariant exists/does not exist in the given language". Investigating the decidability of this problem is important to better understand the foundation of existing methods for invariant inference (e.g. abstract interpretation [13], PDR [9, 26]): whenever the problem is undecidable, no tool will be

[1] A language here means a (usually infinite) set of potential inductive invariants.

able to be complete even for the language in which it is searching; in contrast, when the problem is decidable, tools builders can aim to have complete algorithms for a restricted language.

This work formulates the general problem of inferring inductive invariants in a restricted language L (Section 3), and applies the technique of [2–5] based on well-quasi-orders (wqo's) to get sufficient conditions for decidability of invariant inference (Section 4). The formalization is parametric in the language L, and associates with each language L a quasi-order $\sqsubseteq_L$ on the state space, such that if $\sqsubseteq_L$ is a wqo, then invariant inference in L is decidable. This leads to a (parametric) connection between languages in first-order logic (presented in Section 5) and the decidability technique based on wqo's. This connection is the basis for the main results of this work.

This work makes the following contributions:

Decidability of universal invariants for linked data structures We prove (Section 6) that when modeling programs manipulating singly-linked-lists with effectively propositional logic as in [25], and restricting to universal invariants, the invariant inference problem is decidable (while safety is still undecidable). The technical proof builds on Kruskal's Tree Theorem [27] to show that the suitable $\sqsubseteq_L$ is a wqo. Being formulated in logic, this result naturally extends to capture programs with additional structure beyond list reachability (e.g. sorting algorithms). The complexity of inferring universal invariants for linked-list is non-elementary (Section 8.4).

Undecidability of alternation-free invariants for linked data structures We show (Section 8.2) that in the same setting, inferring alternation-free invariants is undecidable. This demonstrates that the invariant inference problem is theoretically harder than invariant checking, since in this setting checking inductiveness of an alternation-free invariant is decidable. This also shows that mixing universal and existential information even without alternation makes invariant inference undecidable, so in this setting the restriction to universal invariants is necessary for decidability of invariant inference.

Undecidability of universal invariants for general systems Modeling systems beyond linked-lists requires additional unrestricted relations. However, we show (Section 8.3) that in the presence of a single unrestricted binary relation, invariant inference is undecidable even when restricting to universal invariants. This is done by constructing a safe transition system that has a universal inductive invariant if and only if a given counter machine halts.

New decidability from old To overcome the general undecidability while allowing unrestricted relations, we provide (Section 7) systematic ways to construct classes of systems and languages for invariants for which invariant inference is decidable. These constructions start with some established wqo, (e.g. the linked-lists class with universal invariants) and gradually extend it to construct new systems with suitable wqo's. This process results in systems richer than the original one, while decidability is maintained by further restricting the language of potential invariants. We demonstrate the constructions by obtaining a decidable fragment of the invariant inference problem that captures a nontrivial example of a network learning switch.

2. Overview

In this section we provide a short overview of the problem addressed by the paper, and its main results.

2.1 Motivation and Background

Inductive invariants may be difficult to find either manually or using automatic program analysis. For example, Figure 1 shows a simple loop with a property $P = x > 0$.[2] The program executes the loop an unbounded number of times starting from $x = 1$ and $y = 3$. Obviously, $x < 1000$ is not an invariant at all in this program since it is violated after 500 loop iterations. Interestingly, the required safety property $P = x > 0$ is invariant in this program but it is not inductive. For example, if we execute the program in states in which y is negative, it will be violated. In order to come up with inductive invariants we need to also prove something about y. For example, $x > 0 \land y \geq 2$ and $x > 0 \land y > 0$ are both inductive invariants for P. However, if we restrict the language of inductive invariants to consider x only, then no inductive invariant for P exists.

```
x := 1; y := 2;
while * do
   assert x > 0;
   x := x + y; y := y + 1
```

Figure 1. A simple loop example.

Inferring restricted inductive invariants This paper addresses the decision problem that corresponds to inductive invariant inference, defined as follows:

> given a (usually infinite) family of candidate invariants L, a transition relation, τ, initial assumptions *init*, and a safety property P: does there exist a formula in L which is an inductive invariant for $\langle \mathit{init}, \tau, P\rangle$?

Notice that this problem is relevant for abstract interpretation [13] since L can be viewed as an abstract domain, in which case this question amounts to asking if the abstract domain is precise enough for the given program and property. Also notice that in reality one is interested in efficient decision procedures, which is beyond the scope of this paper. Finally, we note that if L is a finite set and if invariant checking is decidable, then the problem of inferring inductive invariants is trivially decidable.

Working with a restricted language of potential invariants L is beneficial in many situations, as it may lead to decidability both of invariant checking, and of invariant inference. Additionally, we note that a negative answer to the invariant inference problem of the form: "There exists no inductive invariant in L which can be used to verify that your program satisfies P" can be useful for programmers. The programmer may decide to simplify her program such that there will be an inductive invariant in L. In our limited interaction with systems researchers, we observed that they are willing to change their programs if the verifier provides a clear explanation as to why there is no inductive invariant in the restricted language. In any case, such an answer is better than an inconclusive alarm of the form: "Your program may not satisfy P" which even the most sophisticated static analysis tools sometimes provide.

Well-Quasi-Orders To address the decidability of inferring inductive invariants in a restricted language L, we apply the technique of [2–4] based on well-quasi-orders (wqo's).

The set of candidate invariants L naturally defines the following quasi-order on states (models):

$$s \sqsubseteq_L s' \quad \text{iff} \quad \forall I \in L.\, s' \models I \Rightarrow s \models I \qquad (1)$$

Thus, lower states satisfy more formulas from L. Notice that this is a quasi-order, i.e., it is reflexive and transitive. Whenever $\sqsubseteq_L$ is a well-quasi-order (wqo), that is any infinite sequence $s_0, s_1, \ldots$, contains an increasing pair $s_i \sqsubseteq_L s_j$ with $i < j$, then invariant inference in L is decidable. Intuitively, invariant inference in L can be done by backward reachability analysis (which can be seen as

[2] We note that this work does not consider numeric domains, but it is useful to illustrate the notion of inductive invariants using a simple numeric program.

iterative application of weakest precondition). The fact that $\sqsubseteq_L$ is a wqo guarantees the termination of this process.

Universal invariants A natural and useful language for inductive invariants is the set of universal sentences. Universal invariants can be used to prove properties of many infinite-state systems, e.g. parameterized systems and programs with unbounded heap allocation or unbounded arrays. The universal quantification is usually over all nodes in a network, or all the elements in the heap or the array (e.g. [35] for heaps and [14, 34] for arrays). Furthermore, linked-lists can be formulated using a theory of list reachability which allows deciding inductiveness of universal invariants using effectively proportional logic (EPR) [25]. When L is the set of all universal sentences, we denote the quasi-order of Equation (1) by $\sqsubseteq_{\forall^*}$. We note that $\sqsubseteq_{\forall^*}$ is the substructure relation known from model theory (e.g. [12]). That is, $s \sqsubseteq_{\forall^*} s'$ iff s is isomorphic to a substructure of s' (s can be embedded in s').

2.2 Decidability of Inferring Universal Invariants for Linked Data Structures

When modeling programs manipulating singly-linked-lists with effectively propositional logic as in [25], we prove that the structures that arise are well-quasi-ordered by the substructure relation. This leads to decidability of inferring universal invariants for such programs. The proof utilizes Kruskal's Tree Theorem [27], and relates homeomorphic embedding of trees with the substructure relation. The proof using Kruskal's Tree Theorem naturally supports adding any finite number of unary relations, which denote sets of individuals, while still maintaining decidability of invariant inference. This allows adding unary instrumentation relations, which is useful for example in verifying sorting implementations [30].

2.3 Undecidability and Complexity of Invariant Inference

A natural way to extend universal invariants is by allowing Boolean combinations of universal invariants and existential invariants. These are called *alternation-free* invariants since they forbid alternating universal and existential quantifiers. In [25], it was shown that this class of invariants suffices to prove partial correctness of many linked-list manipulating programs. Moreover, checking inductiveness of alternation-free invariants is still decidable.

While checking inductiveness is still decidable, we show that the invariant inference problem is undecidable for alternation-free invariants. This is proven by a reduction from the halting problem of counter machines: given a counter machine we construct a program and property such that the counter machine terminates in k steps if and only if there exists an inductive invariant with $O(k)$ quantifiers for the program. While this result is somehow expected given the unbounded nature of quantified invariants and the complexity of inductive reasoning, we note that it differs from the standard reductions (e.g.,[8]) which show that the *safety* problem for programs with infinite state space is undecidable, as the constructed program is safe whether the counter machine halts or not. This result shows an interesting case where invariant inference is undecidable while invariant checking is decidable.

The undecidability result implies that for linked-list manipulating programs, restricting to universal invariants is necessary for the decidability of invariant inference. By using a similar reduction, we also show that inferring universal invariants has non-primitive complexity. This reduction is from the safety problem of lossy counter machines: given a lossy counter machine, we construct a program such that the program has a universal inductive invariant iff the lossy counter machine is safe.

For modeling systems beyond linked-lists, additional unrestricted relations are needed. However, we show that even when restricting to universal invariants, invariant inference becomes undecidable in the presence of a single unrestricted binary relation. We do this by a similar reduction from the halting problem of counter machines. As before, we construct a system that has a universal invariant with $O(k)$ quantifiers iff a given counter machine halts in k steps. This shows that for many general systems beyond linked-lists, inferring universal invariants is undecidable.

2.4 Systematic Constructions for Decidability

The most exciting part of our work is the ability to show that it is decidable to infer restricted universal invariants in many parametric systems. We note that despite their inherent limitations, universal invariants can be used to model many aspects of systems by using uninterpreted relations. This includes relations with high-arity, partial and total orders. However, as noted above, for the language of all universal invariants, even one unrestricted binary relation makes invariant inference undecidable. To obtain decidability, we further restrict the language of potential universal invariants. To do so, we start from an already established "base" wqo (e.g., the linked-lists class with universal invariants), and present systematic constructions that extend classes of systems and languages for invariants in a limited way to construct new systems for which $\sqsubseteq_L$ is a wqo by construction, and thus invariant inference is decidable (see Section 7). The resulting languages are subsets of the set of all universal invariants.

Symmetric lifting We show that a decidability result for some class of programs and language which relies on an underlying theory (e.g., the theory of singly-linked-lists [25]) can be lifted to systems that are modeled by an unbounded number of instances of the theory, by creating a language of restricted universal sentences, that do not correlate the different instances. We call this operation *symmetric lifting*, and it can be used to model systems beyond linked-lists by using high-arity relations. For example, the routing tables in a network of switches may be modeled by a ternary relation $route$, where $route(d, m, n)$ denotes that messages sent to destination d which arrive at switch m will be forwarded to n. Essentially, the routing tables can be viewed as an unbounded number of linked-lists — one for each destination d. In many cases the invariants do not need to correlate the different instances of the original theory (e.g., it is unnecessary to relate the routing tables for different destinations). For such cases, we show that an established wqo (e.g. that of linked-lists) can be lifted to obtain a wqo for a system where the relations have an increased arity, which corresponds to an unbounded number of instances of the base theory. The fact that wqo's are preserved by symmetric lifting is proved using Higman's Lemma [23].

Bounded occurrences of unrestricted relations It is sometimes necessary for the invariant to mention unrestricted relations (i.e., relations that do not obey any background theory). For example, a model of a distributed protocol such as a learning switch may use a relation $pending$ of arity 4, where $pending(s, d, m, n)$ denotes that a packet with source s and destination d is pending on the link from n to m. Moreover, the invariant might need to relate $pending$ messages to reachability information such as forwarding paths. To handle such cases, we show that wqo and thus decidability of invariant inference is preserved by extending a language of universal invariants to include unrestricted relations, as long as only a *bounded* number of occurrences of these relations appear in each universal clause.

This is not trivial since the occurrences of the unrestricted relations are allowed to create correlations with relations that may appear an unbounded number of times under an unbounded number of quantifiers (e.g., to correlate $pending$ messages to paths in a forwarding graph). For this reason, this result cannot be obtained as a straightforward cartesian product with a finite domain, and the proof also relies on Higman's Lemma to preserve the wqo.

3. The Inductive Invariant Inference Problem

In this section we formalize the inductive invariant decision problem parameterized by the language of invariants, and contrast it with the classical safety decision problem. We start with basic definitions related to the problem at hand.

Transition systems and safety properties A *transition system* is a triple $TS = (S, S_0, R)$ where S is a set of states, $S_0 \subseteq S$ is a set of *initial* states, and $R \subseteq S \times S$ is a *transition relation*. A *trace* of *TS* is a sequence of states $s_0, s_1, \ldots, s_n$ such that $s_0 \in S_0$ and for every $0 \leq i \leq n-1$, $(s_i, s_{i+1}) \in R$. A state $s' \in S$ is *reachable* if there exists a trace $s_0, s_1, \ldots, s_n$ such that $s_n = s'$. Given a set $A \subseteq S$, the *image* of A denoted by $R(A)$ is given by $R(A) = \{s' \in S \mid \exists s \in A. (s, s') \in R\}$.

A *safety property* is defined by a set of states $P \subseteq S$. *TS* is *safe with respect to* P, denoted $TS \models P$, if the set of all reachable states of *TS* is a subset of P. With overloading of terminology, we sometimes use the phrase *transition system* to mean a transition system and a safety property: (S, S_0, R, P).

Classes of transition systems and properties We are interested in formulating decision problems where the input consists of a transition system and a safety property. In the following sections we investigate decidability of these problems for various *classes* of transition systems and safety properties. We use $\mathcal{C}$ to denote such a class. For example, the class of digital circuits describes the set of states of a transition system by Boolean variables, and describes the set of initial states, the transition relation and the property using propositional formulas. Another example is that of programs in some programming language, where the property might be defined via assertions within the program. We write $(TS, P) \in \mathcal{C}$ to denote that (TS, P) is an instance of class $\mathcal{C}$.

The safety decision problem The classical problem of determining the safety of a transition system taken from a given class $\mathcal{C}$ of transition systems can be formulated as the following decision problem:

$$\texttt{SAFE}[\mathcal{C}] = \{(TS, P) \in \mathcal{C} \mid TS \models P\}$$

Recall that for many classes of infinite-state transition systems, $\texttt{SAFE}[\mathcal{C}]$ is undecidable (as *TS* can encode a Turing machine).

Inductive invariants Let (S, S_0, R, P) be a transition system with safety property. A set $I \subseteq S$ is an *inductive invariant* for the transition system if (i) $S_0 \subseteq I$, (ii) $R(I) \subseteq I$, and (iii) $I \subseteq P$. It is well known that $TS \models P$ if and only if there exists an inductive invariant for (TS, P). Specifically, if $TS \models P$, then the set of all reachable states of *TS* is an example of an inductive invariant.

Definition 1 (Counterexample to Inductiveness)**.** Let (S, S_0, R, P) be a transition system with safety property, and let $I \subseteq S$ be a set of states. A state $s \in S$ is a *counterexample to inductiveness* of I if (i) $s \in S_0$ but $s \notin I$, or (ii) $s \in I$, but there exists $s' \notin I$ such that $(s, s') \in R$, or (iii) $s \in I$ but $s \notin P$. Clearly, a set $I \subseteq S$ is inductive iff there is no counterexample to inductiveness of I.

Languages for expressing inductive invariants We are interested in deciding whether a given transition system has an inductive invariant which is *expressible in a given language*. Therefore, the input to the inductive invariant problem is a transition system with a safety property, (TS, P), and a language $L \subseteq 2^S$ (where S is the set of states of *TS*) that determines the inductive invariants of interest. As such, the inductive invariant problem is defined not only with respect to a class $\mathcal{C}$ of transition systems (and properties), but also with respect to a class $\mathcal{L}$ of languages.

For example, if $\mathcal{C}$ is the class of digital circuits defined via propositional formulas over Boolean variables, then $\mathcal{L}$ might restrict to languages L that contain sets expressible by propositional formulas over some of the variables. In this case, if the formulas can mention all the propositional variables, then every set of states is in L. On the other hand, if $\mathcal{C}$ defines transition systems using first-order logic formulas, and $\mathcal{L}$ restricts to languages L that contain sets expressible by quantifier free formulas (or universal formulas), some sets of states might not be in L. Since the definition of $\mathcal{L}$ might depend on $\mathcal{C}$, we consider them as a pair $(\mathcal{C}, \mathcal{L})$. We write $(TS, P, L) \in (\mathcal{C}, \mathcal{L})$ to denote that (TS, P, L) is an instance of $(\mathcal{C}, \mathcal{L})$. We say that a set $A \subseteq S$ is expressible in L if $A \in L$.

The inductive invariant inference problem Given $\mathcal{C}$ and $\mathcal{L}$, we define the decision problem $\texttt{INV}[\mathcal{C}, \mathcal{L}]$ as follows: given a transition system, *TS*, a safety property, P, and a language, $L \subseteq 2^S$, such that $(TS, P, L) \in (\mathcal{C}, \mathcal{L})$, is there an inductive invariant for (TS, P) which is expressible in L. Formally:

$$\begin{aligned}\texttt{INV}[\mathcal{C}, \mathcal{L}] = \{&(TS, P, L) \in (\mathcal{C}, \mathcal{L}) \mid \\ &\exists I \in L \text{ s.t. } I \text{ is an inductive invariant for } (TS, P)\}\end{aligned}$$

Note that for every $\mathcal{C}$ and $\mathcal{L}$, if $(TS, P, L) \in \texttt{INV}[\mathcal{C}, \mathcal{L}]$ then $(TS, P) \in \texttt{SAFE}[\mathcal{C}]$. That is, if there exists an inductive invariant (in L) for a transition system then it is safe. The converse does not necessarily hold, and there could be cases where $(TS, P) \in \texttt{SAFE}[\mathcal{C}]$ but $(TS, P, L) \notin \texttt{INV}[\mathcal{C}, \mathcal{L}]$. This can happen if the language L is not expressive enough to describe an inductive invariant for (TS, P). This suggests that $\texttt{INV}[\mathcal{C}, \mathcal{L}]$ may be decidable even if $\texttt{SAFE}[\mathcal{C}]$ is undecidable.

Also note that decidability issues are of interest when $\mathcal{C}$ allows the definition of an infinite set of states and $\mathcal{L}$ allows the definition of infinitely many sets, since in the finite case both $\texttt{SAFE}[\mathcal{C}]$ and $\texttt{INV}[\mathcal{C}, \mathcal{L}]$ can be decided by a naive enumeration. This is the case, for example, for the class of digital circuits with invariants expressed as propositional formulas.

Effectiveness assumptions In the sequel we restrict our attention to classes $\mathcal{C}$ and languages $\mathcal{L}$ such that: (i) there is a decision procedure that, given $(TS, P, L) \in (\mathcal{C}, \mathcal{L})$, determines membership in S, S_0, R, P and L, (ii) there exists a decision procedure that checks, given $(TS, P, L) \in (\mathcal{C}, \mathcal{L})$ and a set $I \in L$, whether I is an inductive invariant for (TS, P), and provides a counterexample to inductiveness of I if it is not.

Note that since $\mathcal{C}$ and $\mathcal{L}$ are used to define decision problems, they come with a finite encoding of their instances, and the set of instances of $(\mathcal{C}, \mathcal{L})$ is always enumerable and decidable.

4. Sufficient Conditions for Decidability of $\texttt{INV}[\mathcal{C}, \mathcal{L}]$

In this section, we apply the technique of [2–5] to obtain sufficient conditions for the decidability of $\texttt{INV}[\mathcal{C}, \mathcal{L}]$. To do so, for $(TS, P, L) \in (\mathcal{C}, \mathcal{L})$, we define a quasi-order, denoted $\sqsubseteq_L$, on the states of the transition system *TS*. The quasi-order $\sqsubseteq_L$ has the property that if it is a well-quasi-order, and if $(\mathcal{C}, \mathcal{L})$ has several other simple properties, then $\texttt{INV}[\mathcal{C}, \mathcal{L}]$ is decidable.

Well-founded sets and well-quasi-orders We first recall the definitions of a well-quasi-order (wqo) and a well-founded set. Let $(X, \leq)$ be a quasi-order, i.e., $\leq$ is reflexive and transitive. We say that $(X, \leq)$ is *well-founded* if it does not contain any infinite strictly decreasing chain $x_0 > x_1 > \ldots$. We say that the infinite sequence $x_0, x_1, \ldots$ is an *antichain* if every two elements in it are incomparable, i.e., $x_i \not\leq x_j$ for all $i \neq j$. We say that $(X, \leq)$ is a *well-quasi-order (wqo)* if for every infinite sequence of elements $x_0, x_1, \ldots$ there exists $i < j$ such that $x_i \leq x_j$. Equivalently, $(X, \leq)$ is a wqo if $(X, \leq)$ is well-founded and does not contain antichains.

4.1 Quasi-Order and Exclusion Operator for L

Definition 2 ($\sqsubseteq_L$)**.** For any language $L \subseteq 2^S$, we define a quasi-order $\sqsubseteq_L$ on S given by

$$s_1 \sqsubseteq_L s_2 \text{ iff for all } A \in L : s_2 \in A \text{ implies } s_1 \in A$$

Thinking of A as an L-property, $s_1 \sqsubseteq_L s_2$ says that every L-property satisfied by s_2 is satisfied by s_1. That is, s_1 satisfies more (or the same) L-properties than s_2.

Definition 3 (Avoid$_L$)**.** Let $s \in S$ be a state and let $A \in L$ be a set such that $s \notin A$, and for every $A' \in L$, if $s \notin A'$ then $A' \subseteq A$. Then we denote A by Avoid$_L(s)$.

That is, Avoid$_L(s)$ is the maximum (w.r.t. set inclusion) over all sets in L that do not include s. Note that Avoid$_L(s)$ need not exist. However, if it exists, it is unique and equal to the union of all sets in L that do not include s. Avoid$_L(s)$ and $\sqsubseteq_L$ are strongly related:

Lemma 4.1. *For all s, s' such that Avoid$_L(s)$ exists, $s \sqsubseteq_L s'$ iff $s' \notin$ Avoid$_L(s)$.*

Proof. The "only-if" direction directly follows from the definitions. For the "if" direction, assume $s \not\sqsubseteq_L s'$. Then there exists $A \in L$ such that $s' \in A$, but $s \notin A$. The latter implies $A \subseteq$ Avoid$_L(s)$ and thus, $s' \in$ Avoid$_L(s)$. □

4.2 L-Relaxed Transition System & Decidability of INV[$\mathcal{C}, \mathcal{L}$]

In this section, we formulate sufficient conditions for decidability of INV[$\mathcal{C}, \mathcal{L}$]. Given $(\mathcal{C}, \mathcal{L})$, we say that Avoid$_L$ is *computable in* $(\mathcal{C}, \mathcal{L})$ if there exists a procedure that, given $(TS, P, L) \in (\mathcal{C}, \mathcal{L})$ and $s \in S$, computes Avoid$_L(s)$. (In particular, this implies that Avoid$_L(s)$ exists for every $s \in S$.)

Theorem 4.2. *Let $(\mathcal{C}, \mathcal{L})$ be such that (i) for every $(TS, P, L) \in (\mathcal{C}, \mathcal{L})$, $(S, \sqsubseteq_L)$ is a wqo, (ii) Avoid$_L$ is computable in $(\mathcal{C}, \mathcal{L})$, and (iii) for every $(TS, P, L) \in (\mathcal{C}, \mathcal{L})$, L is closed under finite intersections. Then INV[$\mathcal{C}, \mathcal{L}$] is decidable. Furthermore, Algorithm 1 is a decision procedure for it.*

Algorithm 1: Backward Reachability Analysis

```
1  $I := S$
2  while $I$ is not an inductive invariant for $(TS, P)$ do
3      if $S_0 \not\subseteq I$ then return no inductive invariant in $L$
4      let $s$ be a counterexample to inductiveness of $I$
5      $I := I \cap \text{Avoid}_L(s)$
6  return $I$ is an inductive invariant in $L$
```

We prove Theorem 4.2 by proving partial correctness and termination of Algorithm 1. Intuitively, Algorithm 1 simulatneously searches for an inductive invariant in L and for an indication that such an inductive invariant does not exist. The algorithm uses counterexamples to inductiveness to iteratively strengthen the candidate invariant. Strengthening is performed by excluding the counterexamples using Avoid$_L$ (Line 5). An indication that an inductive invariant in L does not exist comes in the form of a trace of a *relaxed transition system*, which we define below, from an initial state to a "bad" state (violating P). Such a trace does not imply that the original transition system is unsafe, but as we show next, it implies that no inductive invariant exists in L, hence partial correctness of the algorithm follows. Finally, the wqo property is used to rule out an infinite sequence of strengthenings, hence ensuring termination of Algorithm 1 and proving Theorem 4.2.

Definition 4. Given a transition system $TS = (S, S_0, R)$ and a language $L \in 2^S$, we define the *L-relaxed transition system* $TS^L = (S, S_0, R^L)$ by

$$(s, s') \in R^L \text{ iff } (s, s') \in R \text{ or } s' \sqsubseteq_L s.$$

We say that a trace of TS^L is an *L-relaxed trace* of TS.

Lemma 4.3. *Let $(TS, P, L) \in (\mathcal{C}, \mathcal{L})$ be a transition system and language. Then $(TS, P, L) \in$ INV[$\mathcal{C}, \mathcal{L}$] implies $TS^L \models P$.*

Proof. If $(TS, P, L) \in$ INV[$\mathcal{C}, \mathcal{L}$], there is $I \in L$ that is an inductive invariant for (TS, P). Let $s_0, \ldots, s_N$ be an L-relaxed trace. We prove by induction that for all $i \leq N$, $s_i \in I$. Indeed, since $s_0 \in S_0$ and $S_0 \subseteq I$, we have $s_0 \in I$. For the induction step, assume $s_i \in I$ and consider the L-relaxed transition step $(s_i, s_{i+1}) \in R^L$. If $(s_i, s_{i+1}) \in R$, then $s_{i+1} \in I$ by inductiveness of I. If $s_{i+1} \sqsubseteq_L s_i$, then $s_{i+1} \in I$ since $I \in L$. Since $I \subseteq P$, we conclude that any reachable state of TS^L is in P, and thus $TS^L \models P$. □

Lemma 4.4 (Partial Correctness of Algorithm 1)**.** *If Algorithm 1 terminates, then its output is correct.*

Proof. If Algorithm 1 determines that I is an inductive invariant, correctness follows from the loop condition. For the case where Algorithm 1 determines that no inductive invariant exists in L, we prove by induction on the number of loop iterations that for every state $s' \notin I$, there exists an L-relaxed trace of TS leading from s' to some state $\notin P$. Hence, if $S_0 \not\subseteq I$, it follows that $TS^L \not\models P$, and by Lemma 4.3, $(TS, P, L) \notin$ INV[$\mathcal{C}, \mathcal{L}$]. The base case of the induction is clear. For the induction step, let s' be a state that is removed from I since it is not in Avoid$_L(s)$. By Definition 1, either $s \notin P$ or there exists $s'' \notin I$ such that $(s, s'') \in R$. By the induction hypothesis for I, we get that in both cases, s itself has an L-relaxed trace leading to some state $\notin P$. By Lemma 4.1, $s \sqsubseteq_L s'$, so s' also has an L-relaxed trace leading to a state $\notin P$. □

In the general case, Algorithm 1 is not guaranteed to terminate. However, the following lemma gives a natural condition for its termination:

Lemma 4.5 (Termination of Algorithm 1)**.** *If $(L, \subseteq)$ is well-founded, then Algorithm 1 always terminates.*

Proof. For $i \geq 0$, let I_i denote the set I at the i'th loop iteration of Algorithm 1. The sequence $I_0, I_1, \ldots$ is strictly decreasing with respect to set inclusion, so it must be finite by well-foundedness. □

The proof Theorem 4.2 is completed by the following lemma:

Lemma 4.6. *Let $L \subseteq 2^S$ be a language such that $(S, \sqsubseteq_L)$ is a wqo, then $(L, \subseteq)$ is well-founded.*

Proof. Assume to the contrary that there exists a strictly decreasing infinite sequence $A_0 \supset A_1 \supset A_2 \ldots$ of sets in L. For every $i \geq 0$ let s_i be a state in $A_i \setminus A_{i+1}$. For every $i < j$, $A_j \subseteq A_{i+1}$, and therefore $s_i \notin A_j$. Hence, for every $i < j$, $A_j \in L$ and $s_j \in A_j$ but $s_i \notin A_j$. This implies that $s_i \not\sqsubseteq_L s_j$ for every $i < j$, in contradiction to the fact that $(S, \sqsubseteq_L)$ is a wqo. □

In fact, the correctness proof of Algorithm 1 also implies the converse of Lemma 4.3 under the conditions of Theorem 4.2. This will become useful in Section 8.4.

Corollary 4.7. *Let $(\mathcal{C}, \mathcal{L})$ be such that (i) for every $(TS, P, L) \in (\mathcal{C}, \mathcal{L})$, $(S, \sqsubseteq_L)$ is a wqo, and (ii) Avoid$_L$ is computable in $(\mathcal{C}, \mathcal{L})$. Then $(TS, P) \in$ INV[$\mathcal{C}, \mathcal{L}$] if and only if $TS^L \models P$.*

Proof. If $TS^L \models P$, then a L-relaxed trace leading from an initial state to a state $\notin P$ does not exist. By the proof of Lemma 4.4, this implies that Algorithm 1 does not terminate in Line 3. Thus, it necessarily finds an inductive invariant. □

5. FOL Classes of Transition Systems and Languages of Invariants

In the rest of the paper we focus on classes of transition systems and properties and on languages of inductive invariants that are based on first-order logic, as we describe next.

5.1 FOL Classes of Transition Systems and Properties

FOL classes use first-order logic to specify transition systems and their properties. A transition system *TS* $= (S, S_0, R)$ is specified by a tuple *TS* $= (\Sigma, \Gamma, \varphi_0, \tau)$ where Σ is a *vocabulary* which consists of constants and relation symbols, Γ is a *background theory* given as a set of closed formulas over Σ,[3] φ_0 is a closed formula over Σ representing the initial states formula, and τ is a *two-vocabulary* closed formula representing the transition relation. In more detail, τ has vocabulary $\Sigma \cup \Sigma'$ where $\Sigma' = \{v' \mid v \in \Sigma\}$. A safety property is also represented by a formula, φ_P, over Σ.

In the sequel, unless explicitly stated otherwise, all formulas we consider are *closed* formulas (sentences).

A structure, $s = (D, \mathcal{I}) \in$ STRUCT$[\Sigma]$ consists of a *finite* domain, D, and an interpretation function, $\mathcal{I}$, mapping each symbol of Σ to its meaning in s.

The set of states of the transition system, $S =$ STRUCT$[\Sigma, \Gamma]$, is the set of all (finite) structures in STRUCT$[\Sigma]$ that satisfy Γ. The set of initial states, S_0, is the set of states satisfying φ_0. Similarly, for the set of "safe" states represented by φ_P.

The two-vocabulary transition formula τ, represents transitions from $s = (D, \mathcal{I})$ to $s' = (D, \mathcal{I}')$. The set of transition pairs is

$$R = \{((D, \mathcal{I}_1), (D, \mathcal{I}_2)) \in S \times S \mid (D, \mathcal{I}_1 \cup \mathcal{I}_2') \models \tau\}$$

where $\mathcal{I}_2(v) = \mathcal{I}_2'(v')$.

FOL classes differ in the restrictions they impose on Σ, and on the formulas in Γ, φ_0, τ and φ_P.

EPR classes Of special interest are formulas expressed in the effectively-propositional (EPR) fragment of first-order logic, also known as the Bernays-Schönfinkel-Ramsey class. EPR is restricted to relational first-order formulas (i.e., formulas over a relational vocabulary that may contain constant symbols and relation symbols but no function symbols) with a quantifier prefix $\exists^*\forall^*$ when written in prenex normal form (i.e., when written as a prefix of quantifiers followed by a quantifier-free formula). Satisfiability of EPR formulas is reducible to Boolean SAT, hence decidable [25]. Moreover, theories in this fragment enjoy the *small model property*, meaning that a satisfiable formula is guaranteed to have a finite model of a size proportional to the depth of quantifier nesting. An EPR class is a FOL class which imposes the following restrictions:

Definition 5. An *EPR transition system* is a transition system (*TS*, φ_P), such that *TS* $= (\Sigma, \Gamma, \varphi_0, \tau)$ where Γ is a finite set of EPR formulas, φ_0 and τ are EPR formulas, and φ_P has a quantifier prefix $\forall^*\exists^*$. (In particular, Σ contains no function symbols.)

The class $\mathcal{C}_{\mathbf{n}^*}$ of programs manipulating singly-linked-lists In this section we define the EPR class $\mathcal{C}_{n^*}$, which uses the surprisingly useful but simple theory introduced in [25] to describe properties of programs manipulating singly-linked-lists in EPR. As described in [25], this theory can be used to model both acyclic and cyclic linked-lists, even though the underlying formulation assumes acyclicity.

Each instance of $\mathcal{C}_{n^*}$ consists of an EPR transition system defined over vocabulary Σ_{n^*} and background theory Γ_{n^*}, defined as follows.

[3] A background theory defines a set of models. In this work, we follow the standard approach where the theory is expressed by a set of formulas. However, the definitions naturally extend to the more general case.

Let the vocabulary Σ_{n^*} contain one binary relation symbol n^*, a finite number of unary relation symbols $u_1, ..., u_k$, and a finite number of constant symbols $c_1, ..., c_m$. The idea is that we want to model a next pointer, n, by only mentioning the binary relation symbol n^*, that encodes the reflexive, transitive closure of n.

We use the following background theory, Γ_{n^*} [25]:

$$\begin{aligned} \forall x, y, z.\, (&n^*(x,y) \wedge n^*(y,x) \leftrightarrow x = y) \wedge \\ &(n^*(x,y) \wedge n^*(y,z) \rightarrow n^*(x,z)) \wedge \\ &(n^*(x,y) \wedge n^*(x,z) \rightarrow n^*(y,z) \vee n^*(z,y)) \end{aligned} \tag{2}$$

which says that n^* is reflexive, anti-symmetric, transitive, and semi-linear in the sense that the set of all points reachable from any given point is linearly ordered.

The following two formulas will be useful:

$$\begin{aligned} \varphi_{next}(x,y) &\equiv n^*(x,y) \wedge x \neq y \wedge \\ &\qquad \forall z.\, n^*(x,z) \rightarrow x = z \vee n^*(y,z) \\ \varphi_{root}(x) &\equiv \forall y.\, n^*(x,y) \rightarrow x = y \end{aligned}$$

Intuitively, $\varphi_{next}(x,y)$ recovers the edge relation n from its reflexive, transitive closure, and $\varphi_{root}(x)$ says that x has no outgoing edges, i.e., it is a root.

The reader should convince herself that the finite models of Γ_{n^*} are directed forests, with the edges directed towards the roots.

5.2 FOL Languages of Inductive Invariants

As we consider classes of transition systems expressed in FOL, we also wish to express inductive invariants as logical formulas in languages based on first-order logic. Of special interest are the following two classes of languages:

- $\mathcal{L}_{\forall^*}$**: class of universal invariants.** The $\mathcal{L}_{\forall^*}$ class of languages restricts formulas to closed universal formulas (i.e., formulas with a $\forall^*$ prefix). Each vocabulary Σ induces another language of class $\mathcal{L}_{\forall^*}$. Since Σ is typically clear from the context, we omit it from the notation, and simply write $\forall^*$ to denote a language from $\mathcal{L}_{\forall^*}$.
- $\mathcal{L}_{\mathbf{AF}}$**: class of alternation-free invariants.** The $\mathcal{L}_{AF}$ class of languages restricts formulas to alternation-free formulas. These are formulas obtained by conjunctions and disjunctions of closed universal formulas (with a $\forall^*$ prefix) and closed existential formulas (with a $\exists^*$ prefix).

5.3 Effectiveness Assumptions in FOL

When FOL classes of transition systems and languages are considered, the effectiveness assumption (i) formulated in Section 3 corresponds to decidability of checking whether a given structure satisfies a formula, and assumption (ii) corresponds to decidability of satisfiability, as explained below.

For the transition system *TS* $= (\Sigma, \Gamma, \varphi_0, \tau)$ and the safety property P given by a first-order formula φ_P, checking whether the first-order formula φ represents an inductive invariant amounts to checking whether (i) $\varphi_0 \Rightarrow_\Gamma \varphi$, (ii) $\varphi \wedge \tau \Rightarrow_{\Gamma \cup \Gamma'} \varphi'$, and (iii) $\varphi \Rightarrow_\Gamma \varphi_P$, where φ' denotes φ except that every $v \in \Sigma$ is substituted by $v' \in \Sigma'$, and similarly for Γ', and $\varphi_1 \Rightarrow_{\tilde{\Gamma}} \varphi_2$ denotes that for every structure $s \in$ STRUCT$[\Sigma, \tilde{\Gamma}]$, if $s \models \varphi_1$, then $s \models \varphi_2$ as well. When $\tilde{\Gamma}$ is clear from the context, we simply write $\Rightarrow$ instead of $\Rightarrow_{\tilde{\Gamma}}$. Recall that validity of a formula is equivalent to unsatisfiability of its negation.

A counterexample to inductiveness of φ (see Definition 1) is now a structure $s \in$ STRUCT$[\Sigma, \Gamma]$ such that (i) $s \models \varphi_0 \wedge \neg\varphi$, or (ii) there exists $s' \in$ STRUCT$[\Sigma, \Gamma]$ such that $(s, s') \models \varphi \wedge \tau \wedge \neg\varphi'$, or (iii) $s \models \varphi \wedge \neg\varphi_P$.

In this paper, all classes we consider are EPR classes, and all languages are alternation-free, usually universal. Note that for an

EPR transition system, checking inductiveness of an alternation-free formula amounts to checking unsatisfiability of EPR formulas, hence it is decidable. In addition, if one of the checks fails, the satisfying *finite* structure provides a counterexample to inductiveness. Therefore, all the effectiveness requirements are satisfied.

5.4 $\sqsubseteq_L$ and Avoid_L in FOL

Finally, we recast in FOL the definitions of $\sqsubseteq_L$ and $\text{Avoid}_L(s)$, which are used to formulate the sufficient conditions for decidability in Theorem 4.2.

A FOL language L of inductive invariants corresponds to a set of formulas, each of which represents a set of states. In this setting, $s_1 \sqsubseteq_L s_2$ iff for all $\varphi \in L$, $s_2 \models \varphi$ implies $s_1 \models \varphi$. Similarly, $\text{Avoid}_L(s)$ is the weakest formula $\varphi \in L$ such that $s \not\models \varphi$. That is, $s \not\models \text{Avoid}_L(s)$, and for every $\varphi' \in L$, if $s \not\models \varphi'$ then $\varphi' \Rightarrow \text{Avoid}_L(s)$.

We note that for FOL languages, requirement iii of Theorem 4.2 corresponds to closure of L under (finite) conjunctions. This property holds for all languages considered in this paper.

6. Decidability of Inferring Universal Invariants for Programs Manipulating Linked-Lists

In this section we consider the class of programs manipulating singly-linked-lists, described in Section 5.1, and use Theorem 4.2 to prove that inferring universal invariants for these programs is decidable:

Theorem 6.1. *$INV[\mathcal{C}_{n^*}, \mathcal{L}_{\forall^*}]$ is decidable.*

Technically, we first prove the requirement of Theorem 4.2 that refers to the computability of $\text{Avoid}_{\forall^*}$ with respect to the more general class of EPR transition systems. We then prove that the first-order representation of linked-lists using $\text{STRUCT}[\Sigma_{n^*}, \Gamma_{n^*}]$ from [25], described in Section 5.1, forms a wqo via $\sqsubseteq_{\forall^*}$ (Theorem 6.4). Since all effectiveness assumptions also hold (and universal invariants are closed under conjunction), it follows that for such programs, inferring universal inductive invariants is decidable. This is in contrast to the fact that the safety problem for such programs is undecidable, as they are Turing Complete. A later result (Theorem 8.1) shows that restricting to universal invariants is necessary for decidability, since inferring alternation-free invariants for such programs is undecidable.

6.1 Computability of $\text{Avoid}_{\forall^*}$ for EPR Classes

To show that $\text{Avoid}_{\forall^*}$ is computable, we use the model theoretic notion of a *diagram* (e.g. [12])

Definition 6 (Diagram)**.** Let $s = (D, \mathcal{I})$ be a finite structure over Σ and let $D = \{e_1, \ldots, e_{|D|}\}$. The *diagram* of s, denoted by *Diag*(s), is the following formula over Σ:

$$\exists x_1 \ldots x_{|D|}.\, \mathtt{distinct}(x_1 \ldots x_{|D|}) \wedge \psi$$

where ψ is the conjunction of:

- $x_i = c$ for every constant symbol c such that $\mathcal{I}(c) = e_i$, and
- $r(x_{i_1}, \ldots, x_{i_k})$ for any relation r of arity k in Σ and any $i_1, \ldots, i_k$ s.t. $(e_{i_1}, \ldots, e_{i_k}) \in \mathcal{I}(r)$, and
- $\neg r(x_{i_1}, \ldots, x_{i_k})$ for any relation r of arity k in Σ and any $i_1, \ldots, i_k$ s.t. $(e_{i_1}, \ldots, e_{i_k}) \notin \mathcal{I}(r)$.

Intuitively, one can think of *Diag*(s) as the formula produced by treating individuals in s as existentially quantified variables and explicitly encoding the interpretation of every constant and every relation symbol. Note that the diagram is well defined since we consider *finite* structures. Recall that $s_1 = (D_1, \mathcal{I}_1)$ is a substructure of $s_2 = (D_2, \mathcal{I}_2)$ if $D_1 \subseteq D_2$ and for every $v \in \Sigma$, $\mathcal{I}_1(v)$ is the restriction of $\mathcal{I}_2(v)$ to D_1. It is well known that $s_2 \models$ *Diag*(s_1) iff s_1 is isomorphic to a substructure of s_2. Moreover, for every closed existential formula φ over Σ, $s \models \varphi$ iff *Diag*$(s) \Rightarrow \varphi$. This immediately implies the following two lemmas:

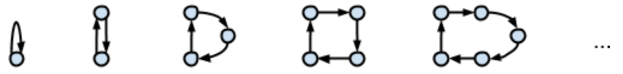
Figure 2. Infinite sequence of incomparable models w.r.t. $\sqsubseteq_{\forall^*}$.

Lemma 6.2. *$s_1 \sqsubseteq_{\forall^*} s_2$ iff s_1 is isomorphic to a substructure of s_2.*

That is, for finite structures, $\sqsubseteq_{\forall^*}$ is the same as the substructure relation (up to isomorphism).

Lemma 6.3. *For every $s = (D, \mathcal{I})$, $\text{Avoid}_{\forall^*}(s)$ is given by the prenex normal form of $\neg$Diag(s). In particular, it is computable.*

6.2 Linked-Lists are WQO via $\sqsubseteq_{\forall^*}$

The proof of Theorem 6.1 is completed by the following theorem:

Theorem 6.4. *$(STRUCT[\Sigma_{n^*}, \Gamma_{n^*}], \sqsubseteq_{\forall^*})$ is a wqo.*

The proof uses Kruskal's Tree Theorem (which we explain next). Note that in general, $(\text{STRUCT}[\Sigma], \sqsubseteq_{\forall^*})$ is not a wqo if Σ contains a binary relation symbol. For example, all the models from the infinite sequence in Figure 2 are incomparable w.r.t. $\sqsubseteq_{\forall^*}$, since none of them can be isomorphically embedded into another.

Kruskal's Tree Theorem Let X be a set. A *labeled graph over X* is a finite undirected graph $G = (V, E, \ell)$ that includes a vertex labeling function $\ell : V \to X$. If G is a tree (undirected connected acyclic graph), then it is called a *labeled tree over X*.

Definition 7 (Tree homeomorphic embedding)**.** Suppose that $(X, \leq)$ is an ordered set. Let $(\mathcal{T}(X), \preceq)$ be the set of all labeled trees over X, with the following ordering: $T_1 \preceq T_2$ iff if T_1 can be obtained from T_2 by a finite number of the following operations:

- Removing a node of degree 1 (and the corresponding edge).
- Removing a node of degree 2 (and the corresponding edges), and adding an edge between its two neighbors.
- Changing the label of a node to a lower value.

Note that the degree of a node is the number of adjacent edges (and not the number of children—since we consider unrooted trees, the notion of children of a node is not well defined).

Fact 6.5 (Kruskal's Tree Theorem, [27, 33].)**.** *If $(X, \leq)$ is a wqo, then so is $(\mathcal{T}(X), \preceq)$.*

Proof of Theorem 6.4. We will encode the directed forests of $(\text{STRUCT}[\Sigma_{n^*}, \Gamma_{n^*}], \sqsubseteq_{\forall^*})$ into the undirected trees of $(\mathcal{T}(X), \leq)$ where X is a certain finite set under the trivial wqo $(X, =)$. We will then apply Fact 6.5 to obtain that $(\mathcal{T}(X), \leq)$ is a wqo. The properties of the encoding will guarantee that this implies that $(\text{STRUCT}[\Sigma_{n^*}, \Gamma_{n^*}], \sqsubseteq_{\forall^*})$ is a wqo.

We first define a function $f : \text{STRUCT}[\Sigma_{n^*}, \Gamma_{n^*}] \to \mathcal{T}(X)$ which encodes each ordered forest F as an undirected tree $f(F)$. The mapping f adds a special new root, v_{root}, connects v_{root} to each root of F, and makes all the n-edges determined by n^*, undirected.

Let $X = 2^{\{u_1, \ldots, u_k, c_1, \ldots, c_m\}} \cup \{l_{root}\}$. We use this finite set X to label each vertex in $f(F)$ according to whether the corresponding vertex from F satisfies each unary predicate, u_i and whether it is equal to the constant c_j. The new vertex v_{root} is labeled l_{root}. The mapping f is illustrated in Figure 3.

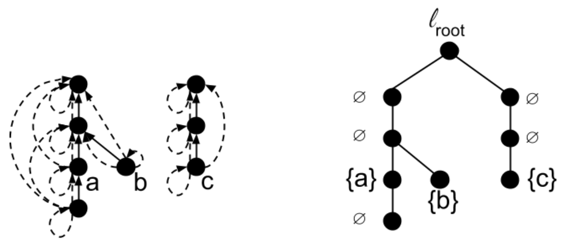

Figure 3. The transformation between STRUCT$[\Sigma_{n^*}, \Gamma_{n^*}]$ and $\mathcal{T}(X)$. Left: a structure with three constants: a, b, c; dashed edges depict n^*; solid edges illustrate $next$. Right: an undirected labeled tree corresponding to the structure on the left.

The following equations explicitly define the mapping f. Given $F = (D, \mathcal{I})$, we put $f(F) = (V, E, \ell)$ where:

$$V = D \cup \{v_{root}\} \text{ where } v_{root} \notin D$$

$$E = \{\{a, b\} \mid a, b \in D \text{ and } F, a, b \models \varphi_{next}\} \cup \{\{a, v_{root}\} \mid F, a \models \varphi_{root}\}$$

$$\ell(a) = \begin{cases} \{u_i \mid a \in \mathcal{I}(u_i)\} \cup \{c_i \mid a = \mathcal{I}(c_i)\} & a \neq v_{root} \\ l_{root} & a = v_{root} \end{cases}$$

The labeling clearly maintains all the information contained in the structure about the constants and the unary predicates, and the new node v_{root} (made distinct by the label l_{root}). It is easy to regain F from the labeled undirected tree, $f(F)$. To formalize this, we define $\mathcal{T}_{n^*} \subseteq \mathcal{T}(X)$ to be the set of all labeled trees, (V, E, ℓ), over X such that there is exactly one node labeled l_{root}, and for every constant symbol c_i there is exactly one node $a \in V$ such that $c_i \in l(a)$. Then, f is one-to-one and onto (up to isomorphism) from STRUCT$[\Sigma_{n^*}, \Gamma_{n^*}]$ to $\mathcal{T}_{n^*}$.

Define the inverse mapping $g \colon \mathcal{T}_{n^*} \to$ STRUCT$[\Sigma_{n^*}, \Gamma_{n^*}]$ as follows. Given $T = (V, E, \ell) \in \mathcal{T}_{n^*}$, let v_{root} be the unique element in V labeled l_{root}, and let $g(T) = (D, \mathcal{I}) \in$ STRUCT$[\Sigma_{n^*}, \Gamma_{n^*}]$, where:

$$D = V \setminus \{v_{root}\}$$

$$\mathcal{I}(c_i) = v_{c_i} \text{ such that } c_i \in \ell(v_{c_i})$$

$$\mathcal{I}(u_i) = \{v \in D \mid u_i \in \ell(v)\}$$

$$\mathcal{I}(n^*) = \{(u, v) \in D^2 \mid \text{the path in } T \text{ from } u \text{ to } v_{root} \text{ contains } v\}$$

Note that for any $T \in \mathcal{T}_{n^*}$, $g(T) \models \Gamma_{n^*}$, and $g = f^{-1}$.

To complete the proof that (STRUCT$[\Sigma_{n^*}, \Gamma_{n^*}], \sqsubseteq_{\forall^*}$) is a wqo, let $s_1, s_i, \ldots$ be an infinite sequence of structures in STRUCT$[\Sigma_{n^*}, \Gamma_{n^*}]$. Consider the infinite sequence $f(s_1), f(s_2), \ldots$ of labeled trees. Since $(\mathcal{T}(X), \preceq)$ is a wqo, there exists $i < j$ such that $f(s_i)$ can be obtained from $f(s_j)$ by the three operations of Def. 7. Since the labels are ordered by equality, there are only two operations to consider: removing a node of degree 1, and replacing a node of degree 2 by an edge. Note that since both $f(s_i), f(s_j) \in \mathcal{T}_{n^*}$, the node labeled by l_{root} and nodes representing constants cannot be removed. Now, consider any $T \in \mathcal{T}_{n^*}$. If T' is obtained from T by removing a node v of degree 1 that is not labeled by l_{root} and does not represent a constant, then clearly $g(T')$ is isomorphic to a substructure of $g(T)$, obtained by removing v from the domain of $g(T)$. This is also true for removing a node of degree 2 and replacing it by an edge, since this operation preserves n^* between all remaining nodes. Since $f(s_i)$ can be obtained from $f(s_j)$ by a finite sequence of such operations, we conclude that s_i is isomorphic to a substructure of s_j, i.e., $s_i \sqsubseteq_{\forall^*} s_j$. Thus, (STRUCT$[\Sigma_{n^*}, \Gamma_{n^*}], \sqsubseteq_{\forall^*}$) is a wqo. □

```
$pending := \emptyset;\ route := \emptyset;\ learned := \emptyset;\ route^* := \{(D, X, X)\}$
while $*$ :
  # choose an arbitrary new or pending packet
  $src, dst, sw_1, sw_2 := *, *, *, *$
  assume $pending(src, dst, sw_1, sw_2) \vee (src = sw_1 = sw_2)$
  if $*$ :
    $pending := pending \setminus \{(src, dst, sw_1, sw_2)\}$
  # if the packet's source is unknown at $sw_2$, learn a new route
  if $\neg learned(src, sw_2) \wedge src \neq sw_2$ :
    # assert that a cycle is not created
    assert $\neg route^*(src, sw_1, sw_2)$
    # update the routing table
    $learned := learned \cup \{(src, sw_2)\}$
    $route := route \cup \{(src, sw_2, sw_1)\}$
    $route^* := route^* \cup \{(src, X, Y) \mid$
                         $route^*(src, X, sw_2) \wedge route^*(src, sw_1, Y))\}$
  # if $dst = sw_2$ consume the packet, otherwise forward it
  if $dst \neq sw_2$ :
    if $\neg learned(dst, sw_2)$ :
      # if no route to $dst$ is known, flood the packet
      $pending := pending \cup \{(src, dst, sw_2, V) \mid link(sw_2, V) \wedge V \neq sw_1)\}$
    else : # otherwise forward according to route
      $pending := pending \cup \{(src, dst, sw_2, V) \mid route(dst, sw_2, V)\}$
```

Figure 4. Model of a learning switch that learns the first connection.

7. Systematic Constructions of Decidable Classes

The result of Section 6 implies that inference of universal invariants is decidable for programs manipulating linked-lists. To model systems beyond linked-lists, one has to use relations of higher arity and/or relations unrestricted by any background theory. However, in Section 8.3 we show that this quickly leads to undecidability of inferring universal invariants. In this section, we develop constructions that maintain decidability of invariant inference imposing further syntactic restrictions on the potential invariants. When combined, the constructions we develop are quite general, and can capture the interesting example of a network learning switch.

Motivating example: network learning switch As a motivating example, we consider the network learning switch protocol. Learning switches maintain routing tables, and learn routes as they receive packets. When a packet first arrives from an unknown source, the switch learns a route to its source through the incoming link. It then checks if it has a route to the packet's destination, and either forwards the packet to a known route or floods it to all but the incoming link. For this protocol, we consider the safety property that the created routing tables do not contain forwarding loops. The learning switch is a parameterized distributed system with infinite-state, as there is an unbounded number of switches, and the routing table of each switch contains an unbounded number of entries. The system and property can be modeled as an EPR transition system. Figure 4 provides a description of such a model.

We now describe the relations used by the model. The relation *link*^{2} describes the links in the network. The relation *pending*^{4} describes pending packets, and *pending*(s, d, sw_1, sw_2) denotes the fact that a packet with source field s and destination field d is pending on the link from switch sw_1 to switch sw_2. The relations *learned*^{2}, *route*^{3}, *route*^{*3} store information about the current routing tables of the switches. *learned*(d, sw) denotes that switch sw has learned a route to destination d. *route*(d, sw_1, sw_2) denotes that a packet with destination field d will be routed by switch sw_1 to switch sw_2. Thus for any d, *route*$(d, \cdot, \cdot)$ holds the *forwarding graph* for d, that describes how packets with destination d will be forwarded in the network. We wish to verify that this graph is acyclic for all d. To this end, the relation *route** describes paths in this graph: *route*$^*(d, sw_1, sw_2)$ holds if *route*$(d, \cdot, \cdot)$ contains a path from sw_1 to sw_2. Formally, *route*$^*(d, \cdot, \cdot)$ is the reflexive transitive closure of *route*$(d, \cdot, \cdot)$ for any d. The modelling maintains this fact by the standard technique of updating transitive closure (e.g. [25]). The `assert` statement asserts that whenever a switch learns a new route, it does not introduce a cycle in the forwarding graph.

Gradual extensions While the model of Figure 4 represents an EPR transition system, it is not apparent how the results of Section 4 and Section 6 can be applied to obtain decidability of invariant inference for it. In the rest of this section we develop constructions that obtain this by limiting the invariants to universal sentences that satisfy further syntactic restrictions. We do so by gradual extensions of classes of transition systems and languages. The extensions start from an established decidable class, and each step extends the expressive power of the transition system or the language for invariants in a limited way that preserves the fact that $\sqsubseteq_L$ is a wqo, and Avoid$_L$ is computable (L is the language of invariants).

Formally, each extension starts with an EPR class paired with a language class of universal invariants $(\mathcal{C}, \mathcal{L})$ which satisfy the conditions of Theorem 4.2, and constructs a new EPR class and a corresponding language class $(\mathcal{C}', \mathcal{L}')$ that also satisfy these conditions, ensuring that $\texttt{INV}[\mathcal{C}', \mathcal{L}']$ is decidable. We define $(\mathcal{C}', \mathcal{L}')$ by describing the vocabulary Σ', the theory Γ', and the language L' used in instances of $(\mathcal{C}', \mathcal{L}')$. For each extension, we show that $\sqsubseteq_{L'}$ is a wqo and Avoid$_{L'}$ is computable (note that since this section only considers EPR classes and languages of universal sentences, all the effectiveness assumptions are trivially satisfied).

Notation. For the remainder of this section, let us fix a vocabulary Σ, a theory Γ consisting of universal sentences (note that this is the case for Γ_{n^*}), and a base language L of universal sentences, taken from $(\mathcal{C}, \mathcal{L})$ that satisfy the conditions of Theorem 4.2. Let $cl(L)$ be the closure of L under conjunction, disjunction, and rewriting into an equivalent formula. Note that $\sqsubseteq_L = \sqsubseteq_{cl(L)}$. Thus, we assume w.l.o.g. that L is closed, i.e., $L = cl(L)$.

Every universal sentence can be written as a conjunction of closed *universal clauses*, each of which has the form $\forall x_1 \ldots x_r.\, \beta$ where β is the *body*, consisting of a disjunction of literals over $x_1 \ldots x_r$. From now on we will just talk about these disjunctive bodies, understanding that $L = cl(A)$ where A is the set of universal clauses obtained by taking a body formula and universally quantifying all its free variables.

7.1 Basic Extensions

The following basic extensions of wqo's are immediate: if $(\text{STRUCT}[\Sigma, \Gamma], \sqsubseteq_L)$ is a wqo then it remains a wqo if we strengthen the background theory, restrict the language, or extend the vocabulary (while keeping the language the same set of formulas).

Proposition 7.1. *If $(\textit{STRUCT}[\Sigma, \Gamma], \sqsubseteq_L)$ is a wqo then so are:*

1. $(\textit{STRUCT}[\Sigma, \Gamma'], \sqsubseteq_L)$, *if* $\Gamma' \models \Gamma$
2. $(\textit{STRUCT}[\Sigma, \Gamma], \sqsubseteq_{L'})$, *if* $L' \subseteq L$
3. $(\textit{STRUCT}[\Sigma', \Gamma], \sqsubseteq_L)$, *if* $\Sigma \subseteq \Sigma'$

Proof. If $\Gamma' \models \Gamma$, then $\text{STRUCT}[\Sigma, \Gamma] \subseteq \text{STRUCT}[\Sigma, \Gamma']$, and case 1 follows. If $L' \subseteq L$, then $\sqsubseteq_L \subseteq \sqsubseteq_{L'}$, and case 2 follows. Finally, if $\Sigma \subseteq \Sigma'$, let $(s'_i)_{i=1}^{\infty}$ be an infinite sequence of structures in $\text{STRUCT}[\Sigma', \Gamma]$. By projecting each state s'_i to a state s_i over Σ, we obtain the sequence $(s_i)_{i=1}^{\infty}$, for which there exist $i < j$ such that $s_i \sqsubseteq_L s_j$. Since L is defined over Σ, for every $\varphi \in L$, $\varphi \models s_i$ iff $\varphi \models s'_i$. Therefore, $s'_i \sqsubseteq_L s'_j$ as well, and case 3 follows. □

Remark 7.2. In the context of Proposition 7.1, if there is a procedure to compute Avoid$_L$, then the same procedure will work for cases 1 and 3, but not necessarily for case 2.

The wqo property is also preserved under unions of languages:

Proposition 7.3. *If $(\textit{STRUCT}[\Sigma, \Gamma], \sqsubseteq_{L_1})$, $(\textit{STRUCT}[\Sigma, \Gamma], \sqsubseteq_{L_2})$ are wqo's then so is $(\textit{STRUCT}[\Sigma, \Gamma], \sqsubseteq_L)$ where $L = cl(L_1 \cup L_2)$.*

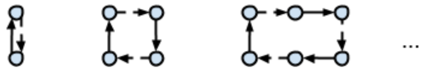

Figure 5. Infinite sequence of incomparable models w.r.t. $\sqsubseteq_{\forall^*}^{n^*, m^*}$. Solid arrows are n-edges, and dashed arrows are m-edges.

Proof. We first prove the claim for $L = L_1 \cup L_2$. As explained above, $\sqsubseteq_L = \sqsubseteq_{cl(L)}$, hence the claim follows for $cl(L_1 \cup L_2)$ as well. Suppose that $s_1, s_2, \ldots$ is an infinite sequence of structures from $(\text{STRUCT}[\Sigma, \Gamma], \sqsubseteq_L)$. Since $(\text{STRUCT}[\Sigma, \Gamma], \sqsubseteq_{L_1})$ is a wqo, there must exist an infinite increasing subsequence under $\sqsubseteq_{L_1}$, $s_{i_1}, s_{i_2}, \ldots$. Since $(\text{STRUCT}[\Sigma, \Gamma], \sqsubseteq_{L_2})$ is a wqo, there exist $j < k$ such that $s_{i_j} \sqsubseteq_{L_2} s_{i_k}$ Thus, every formula in $L_1 \cup L_2$ satisfied by s_{i_k} is satisfied by s_{i_j}. That is, $s_{i_j} \sqsubseteq_L s_{i_k}$, as desired. □

Remark 7.4. If there are procedures to compute Avoid$_{L_1}$ and Avoid$_{L_2}$, then Avoid$_L$ for $L = cl(L_1 \cup L_2)$ is also computable, and given by:

$$\text{Avoid}_L(s) = \text{Avoid}_{L_1}(s) \vee \text{Avoid}_{L_2}(s)$$

Remark 7.5. Proposition 7.3 considers $L = cl(L_1 \cup L_2)$, which contains conjunctions and disjunctions of *closed* universal clauses from L_1 and L_2. Let $L' \supseteq L$ be the universal language containing also clauses whose bodies are obtained by disjunctions of bodies of L_1 with bodies of L_2 (within the scope of the quantifier prefix). It is tempting to try to prove that $\sqsubseteq_{L'}$ is a wqo. However, in general this is not the case. Consider a language $\sqsubseteq_{\forall^*}^{n^*, m^*}$ of universal formulas for shared lists structures in the signature $\{n^*, m^*\}$ and the background theory $\Gamma_{n^*} \wedge \Gamma_{m^*}$ that is a conjunction of two singly-linked-list theories (2). Then we can construct an infinite sequence of models that represent cycles of even length formed by interchanging n and m edges, see Figure 5. This shows that $\sqsubseteq_{\forall^*}^{n^*, m^*}$ is not a wqo.

We can also extend L by adding any ground atom, g (this is applicable for example if case 3 of Proposition 7.1 was applied to add the predicate and/or constants in g to Σ without extending L).

Proposition 7.6. *If $(\textit{STRUCT}[\Sigma, \Gamma], \sqsubseteq_L)$ is a wqo and g is a ground atom of Σ, then $(\textit{STRUCT}[\Sigma, \Gamma], \sqsubseteq_{L'})$ is a wqo where L' has the bodies $\beta, g \vee \beta, \neg g \vee \beta$ for each body β of L.*

Proof. Let $(s_i)_{i=1}^{\infty}$ be an infinite sequence of structures in $\text{STRUCT}[\Sigma, \Gamma]$. Since g is a ground atom, every structure gives it a valuation of true or false, so $(s_i)_{i=1}^{\infty}$ contains an infinite subsequence where all structures give g the same valuation. Therefore there exist $i < j$ such that $s_i \sqsubseteq_L s_j$ and s_i, s_j give g the same valuation. It follows that $s_i \sqsubseteq_{L'} s_j$, and thus $(\text{STRUCT}[\Sigma, \Gamma], \sqsubseteq_{L'})$ is a wqo. □

Remark 7.7. If there is a procedure to compute Avoid$_L$, then for L' of Proposition 7.6, Avoid$_{L'}$ is also computable, and given by

$$\text{Avoid}_{L'}(s) = \begin{cases} \text{Avoid}_L(s) \vee \neg g & s \models g \\ \text{Avoid}_L(s) \vee g & s \not\models g \end{cases}$$

By combining Propositions and Remarks 7.1, 7.2, 7.6, 7.7 we get the following corollary:

Corollary 7.8. *Extending the vocabulary Σ and the language L by adding to Σ any number of new relations and adding to any body of L any number of disjunctions of ground literals constructed from the new relations maintains wqo and computability of Avoid$_L$.*

An operation needed for the constructions that follow later is extension by a new constant symbol. This requires some uniformity from the base language, formalized in the following definition:

Definition 8 (Constant-extendable)**.** Let $(\text{STRUCT}[\Sigma, \Gamma], \sqsubseteq_L)$ be a wqo with Avoid$_L$ computable. We say that L is *constant-extendable* (in the context of Σ and Γ), if for any finite set of fresh constant symbols S:

- $(\text{STRUCT}[\Sigma \cup S, \Gamma], \sqsubseteq_{L'})$ is also a wqo, where the bodies of L' are obtained from the bodies of L by any number of substitutions of variables by constants from S.
- Avoid$_{L'}$ is computable over STRUCT$[\Sigma \cup S, \Gamma]$.

Remark 7.9. Since Theorem 6.4 allows an arbitrary number of constants to begin with, all languages in $(\mathcal{C}_{n^*}, \mathcal{L}_{\forall^*})$ are constant-extendable. Also, the reader should convince herself that if the base languages are constant-extendable, all constructions presented so far result in constant-extendable languages. Obviously, in the context of Definition 8, if L is constant-extendable then so is L'.

7.2 Symmetric Lifting

In this section, we show that a decidability result for some vocabulary, theory and language can be lifted to a vocabulary which describes an unbounded number of instances of the original theory, by parameterizing the theory and creating a language of *symmetric* sentences, that do not correlate the different instances. As an example for this, consider the routing tables of learning switches. For each destination d, each switch has a single "next" pointer for packets destined to d, which is described by the $route$ relation. Thus, the routing tables can be seen as an unbounded number of linked-lists, parameterized by the destination of packets. The extension developed in this section can be used to lift the results of Section 6 to capture the ternary relation $route^*$, and allow invariants to refer to paths in the forwarding graphs with unbounded quantification, as long as they do not to correlate forwarding graphs of different destinations.

The basis for lifting a wqo from a theory to an unbounded number of instances of the theory relies on the following corollary of Higman's Lemma [23]:

Fact 7.10 (Higman's Lemma for finite sets [23]). *If $(X, \leq)$ is a wqo, then so is $(\mathcal{P}_{fin}(X), \preceq)$ where $\mathcal{P}_{fin}(X)$ is the set of finite subsets of X, and $A \preceq B$ iff $\forall s \in A. \exists t \in B. s \leq t$.*

We start by using Fact 7.10 to show that we can perform symmetric lifting and preserve wqo's and computability of Avoid$_L$. We define *symmetric lifting* as the removal of a constant symbol a from the vocabulary Σ, while replacing a by a new universally quantified variable v, both in the theory Γ and in the formulas of L. The latter operation is denoted r_a (e.g. $r_a(\forall x. P(a) \vee Q(a, x)) = \forall v, x. P(v) \vee Q(v, x)$). The next proposition shows that symmetric lifting preserves wqo's and computability of Avoid$_L$.

Proposition 7.11 (Symmetric lifting). *If $(STRUCT[\Sigma, \Gamma], \sqsubseteq_L)$ is a wqo and a is a constant symbol in Σ, then $(STRUCT[\Sigma', \Gamma'], \sqsubseteq_{L'})$ is a wqo where*

$$\Sigma' = \Sigma \backslash \{a\}, \quad \Gamma' = \{r_a(\varphi) \mid \varphi \in \Gamma\}, \quad L' = \{r_a(\varphi) \mid \varphi \in L\}$$

Proof. Define the function $A_a : \text{STRUCT}[\Sigma'] \to \mathcal{P}_{fin}(\text{STRUCT}[\Sigma])$ by

$$A_a((D, \mathcal{I})) = \{(D, \mathcal{I}[a \mapsto d]) \mid d \in D\}$$

That is, A_a maps a structure of Σ' to the set of structures of Σ in which we interpret the new constant symbol, a, in all possible ways. Note that since all our structures are finite, there are only finitely many ways to interpret a in any given structure.

The semantics of a universal quantifier tells us that for any structure $s' \in \text{STRUCT}[\Sigma']$ and any formula φ of vocabulary Σ,

$$s' \models r_a(\varphi) \quad \text{iff} \quad s \models \varphi \text{ for each } s \in A_a(s'). \qquad (3)$$

Thus, if $s' \models \Gamma'$ then for any $s \in A_a(s')$ we have $s \models \Gamma$. Thus, A_a maps STRUCT$[\Sigma', \Gamma']$ to $\mathcal{P}_{fin}(\text{STRUCT}[\Sigma, \Gamma])$.

To prove that $(\text{STRUCT}[\Sigma', \Gamma'], \sqsubseteq_{L'})$ is a wqo, let $(s'_i)_{i=1}^{\infty}$ be an infinite sequence of structures in STRUCT$[\Sigma', \Gamma']$. Now, consider the infinite sequence $(A_a(s'_i))_{i=1}^{\infty}$. This is an infinite sequence of elements of $\mathcal{P}_{fin}(\text{STRUCT}[\Sigma, \Gamma])$, which is a wqo by Fact 7.10 and the fact that $(\text{STRUCT}[\Sigma, \Gamma], \sqsubseteq_L)$ is a wqo. Thus, we have $i < j$ such that:

$$\forall s_1 \in A_a(s'_i)\ \exists s_2 \in A_a(s'_j)\ s_1 \sqsubseteq_L s_2 \qquad (4)$$

To show that $s'_i \sqsubseteq_{L'} s'_j$, let φ' be an arbitrary formula from L' such that $s'_j \models \varphi'$. Thus $\varphi' = r_a(\varphi)$ for some $\varphi \in L$. By eq. (3), $\forall s_2 \in A_a(s'_j)\ s_2 \models \varphi$. Thus, by eq. (4), $\forall s_1 \in A_a(s'_i)\ s_1 \models \varphi$. Thus, again by eq. (3), $s'_i \models \varphi$. □

Remark 7.12. In the setting of Proposition 7.11, if Avoid$_L$ is computable for structures of STRUCT$[\Sigma, \Gamma]$, then Avoid$_{L'}$ is computable for structures of STRUCT$[\Sigma', \Gamma']$ and is given by:

$$\text{Avoid}_{L'}(s') = \bigvee_{s \in A_a(s')} r_a(\text{Avoid}_L(s))$$

The correctness of this definition follows from the definitions of L' and $A_a(s')$, the properties of Avoid$_L$ and Equation (3). Furthermore, if L is constant-extendable (in the context of Σ and Γ), then L' is constant-extendable (in the context of Σ' and Γ').

We now show that Proposition 7.11 can be used to increase the arity of relations and maintain wqo's (e.g. to go from n^* to $route^*$). To do this, we start from a constant-extendable language L, and extend it by a fresh constant symbol a. We use a to replace an n-ary relation r with a relation r' of arity $n+1$. Define $e_a^{r \mapsto r'}$ to denote the substitution of $r(\bar{t})$ by $r'(a\bar{t})$, where $\bar{t}$ is a tuple of variables and constants, and $a\bar{t}$ denotes the tuple consisting of a as its first element, followed by $\bar{t}$ (e.g. $e_a^{n^* \mapsto route^*}(n^*(x, y)) = route^*(a, x, y)$). Then, the next proposition is straightforward:

Proposition 7.13 (Arity extension). *If $(STRUCT[\Sigma, \Gamma], \sqsubseteq_L)$ is a wqo, $r \in \Sigma$ is a relation symbol of arity n, $r' \notin \Sigma$ is a new relation symbol of arity $n+1$, and $a \in \Sigma$ is a constant symbol, then $(STRUCT[\Sigma', \Gamma'], \sqsubseteq_{L'})$ is a wqo where $\Sigma' = \Sigma \setminus \{r\} \cup \{r'\}$, $\Gamma' = \{e_a^{r \mapsto r'}(\varphi) \mid \varphi \in \Gamma\}$, $L' = \{e_a^{r \mapsto r'}(\varphi) \mid \varphi \in L\}$.*

Remark 7.14. In the setting of Proposition 7.13, if Avoid$_L$ is computable for structures of STRUCT$[\Sigma, \Gamma]$, then Avoid$_{L'}$ is computable for structures of STRUCT$[\Sigma', \Gamma']$, and is given by:

$$\text{Avoid}_{L'}(s') = e_a^{r \mapsto r'}(\text{Avoid}_L(s))$$

where, for $s' = (D', \mathcal{I}')$, we define $s = (D', \mathcal{I})$ where $\mathcal{I}$ (defined over $\Sigma' \setminus \{r'\} \cup \{r\}$) is the same as $\mathcal{I}'$, except for $\mathcal{I}(r)$ which is obtained from $\mathcal{I}'(r')$ by truncating the first element in each tuple. Furthermore, if L is constant-extendable (in the context of Σ and Γ), then L' is constant-extendable (in the context of Σ' and Γ').

Extending L by a constant and using Proposition 7.13 followed by Proposition 7.11 results in a vocabulary, theory and language where a relation r has been replaced by a relation r' with increased arity (e.g. replacing n^* by $route^*$). The obtained language L' contains universal sentences, where the occurrences of r' are *symmetric* in their first argument: every universal clause in L' can only use one universally quantified variable as the first argument of r' in all its appearances. Therefore, formulas in L' cannot correlate values of r' for different elements as the first argument. Note however, that the variable used for the first argument of r' can appear elsewhere in the clause, and can be correlated to other relations, including other occurrences of r' (see Section 7.4 for a concrete example).

7.3 Adding Occurrences of Arbitrary Relation Symbols

It is sometimes necessary for the invariant to mention relations that do not obey any background theory (e.g. the $pending$ relation in the learning switch example). This section shows that such relations can be allowed in the language for potential invariants while maintaining decidability, as long as only a *bounded* number of occurrences of these relations appear in each universal clause.

We note that the clauses of the original language may contain unbounded quantification, and the bounded occurrences of the new relations can correlate to other literals in these clauses (the new relations are added to the bodies, i.e. within the scope of the universal quantifiers). For this reason, this result requires the use of Higman's Lemma, and cannot be obtained as a straightforward cartesian product with a finite domain. We prove this result more concisely by building on the operation of symmetric lifting and Proposition 7.11 (which was proven using Higman's Lemma).

Let $r \in \Sigma$ be a relation symbol. We are interested in extending L by adding one occurrence of r to any body, β, of L. Let $A_r(\beta)$ be the set of bodies of the form β, $r(\bar{t}) \vee \beta$, or $\neg r(\bar{t}) \vee \beta$, where $\bar{t}$ is a tuple of variables and constants (including free variables that appear in β). Let $A_r(L)$ have exactly the bodies $A_r(\beta)$ for β a body of L.

Proposition 7.15. *If* ($STRUCT[\Sigma, \Gamma], \sqsubseteq_L$) *is a wqo, Avoid$_L$ is computable, L is constant-extendable, and $r \in \Sigma$ is a relation symbol, then:*

- ($STRUCT[\Sigma, \Gamma], \sqsubseteq_{A_r(L)}$) *is a wqo,*
- *Avoid$_{A_r(L)}$ is computable, and*
- *$A_r(L)$ is constant-extendable.*

Proof. Let n be the arity of r. Let $c_1, \ldots, c_n$ be n fresh constant symbols. Let $\Sigma_1 = \Sigma \cup \{c_1, \ldots, c_n\}$ and let L_1 be L extended by the constant symbols $c_1, \ldots, c_n$ as in Definition 8. Then, since L is constant-extendable, we get that (STRUCT$[\Sigma_1, \Gamma], \sqsubseteq_{L_1}$) is a wqo, Avoid$_{L_1}$ is computable, and L_1 is constant-extendable.

Let L_2 have the bodies, β, of L_1 plus $\beta \vee g$ and $\beta \vee \neg g$ for the ground atom $g = r(c_1, \ldots, c_n)$ Then by Proposition 7.6, (STRUCT$[\Sigma_1, \Gamma], \sqsubseteq_{L_2}$) is a wqo, Avoid$_{L_2}$ is computable, and L_2 is constant-extendable.

Finally, let L_3 be the language obtained by applying Proposition 7.11 and Remark 7.12 n times to remove the constants $c_1, \ldots, c_n$, from the vocabulary (replacing them by universally quantified variables). By Proposition 7.11 and Remark 7.12 (STRUCT$[\Sigma, \Gamma], \sqsubseteq_{L_3}$) is a wqo, Avoid$_{L_3}$ is computable, and L_3 is constant-extendable. By the constructions of L_1, L_2, L_3, we get that $L_3 = A_r(L)$, which completes the proof. □

It immediately follows from Proposition 7.15 that:

Corollary 7.16. *Extending the vocabulary Σ by adding an* arbitrary *relation (i.e., with* any arity*) and extending L by adding to the bodies of L any number $\leq k$ of occurrences of the new relation symbol, for some* fixed $k \geq 0$, *maintains the wqo and computability of Avoid$_L$.*

7.4 Putting It All Together: Application to Learning Switch

We now illustrate the results of this section by applying it to the learning switch model of Figure 4, and obtaining a decidable class for invariant inference that captures it. The class we obtain contains an inductive invariant that proves the absence of forwarding loops.

Recall that for any d, *route*$^*(d, \cdot, \cdot)$ describes the reflexive transitive closure of *route*$(d, \cdot, \cdot)$, which is a functional relation (as explained above), and thus *route*$^*(d, \cdot, \cdot)$ obeys Γ_{n^*} when replaced for n^*. Thus, we start with the result of Section 6, and apply the construction of Section 7.2 to lift n^* to *route**. We denote the resulting theory Γ_{route^*} and the resulting language L_0. L_0 contains universal clauses with any number of occurrences of *route** that are symmetric with respect to its first argument. For the vocabulary $\Sigma_0 = \{route^*\}$, we have that (STRUCT$[\Sigma_0, \Gamma_{route^*}], \sqsubseteq_{L_0}$) forms a wqo with Avoid$_{L_0}$ computable.

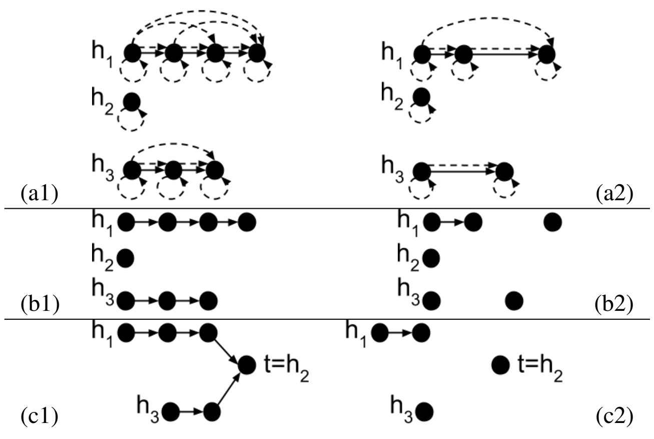

Figure 6. Depiction of the structures obtained by different encodings of counters for $c_1 = 3, c_2 = 0, c_3 = 2$: a structure of $\mathcal{E}_{n^*}$ (a1) and its substructure (a2); a structure of an unsuccessful encoding $\mathcal{E}_n$ (b1) and its substructure (b2); a structure of $\mathcal{E}_n$ (c1) and its substructure (c2).

For any $k > 0$, we obtain L_k by applying Proposition 7.15 $4k$ times (starting with L_0) to allow at most k occurrences of any of the relations *link*, *learned*, *pending* and *route*. Thus, for $\Sigma_{\mathtt{ls}} = \{link, pending, learned, route, route^*\}$, we have that (STRUCT$[\Sigma_{\mathtt{ls}}, \Gamma_{route^*}], \sqsubseteq_{L_k}$) is a wqo with Avoid$_{L_k}$ computable.

For $k = 1$, L_k contains an inductive invariant for the learning switch, that contains clauses such as:

$$\forall x, y, z.\, route^*(x, y, z) \wedge y \neq z \rightarrow learned(x, y)$$
$$\forall w, x, y, z.\, pending(w, x, y, z) \wedge w \neq y \rightarrow learned(w, y)$$

Note that these clauses create correlations between the first argument of *route** and the other relations, and also between the other relations and themselves. These correlations are allowed by the constructions of this section. An example for a clause that would not be allowed is: $\forall x, y, z.\, route^*(x, y, z) \rightarrow route^*(z, y, x)$, as it creates correlations of *route** with different first arguments.

8. Undecidability and Complexity of $\mathtt{INV}[\mathcal{C}, \mathcal{L}]$

In this section, we present several hardness results for $\mathtt{INV}[\mathcal{C}, \mathcal{L}]$. We first present a general scheme for proving undecidability by an interesting reduction from the halting problem of counter machines. We use this scheme to prove that allowing alternation-free invariants for $\mathcal{C}_{n^*}$ leads to undecidability. We then use similar arguments to show that even if we allow only universal invariants, but consider $\mathcal{C}$ that allows a single "unrestricted" binary relation (i.e., not restricted to admit the background theory of $\mathcal{C}_{n^*}$) then $\mathtt{INV}[\mathcal{C}, \mathcal{L}_{\forall^*}]$ is undecidable. This is in contrast to the fact that verifying inductiveness of a given invariant *is* decidable in these cases.

We conclude this section by adapting the reduction from counter machines to a reduction from lossy counter machines, and applying it to $\mathcal{C}_{n^*}$ and $\mathcal{L}_{\forall^*}$, which proves that the decidable problem $\mathtt{INV}[\mathcal{C}_{n^*}, \mathcal{L}_{\forall^*}]$ has non-elementary complexity.

8.1 Reduction from Counter Machines to $\mathtt{INV}[\mathcal{C}, \mathcal{L}]$

This subsection presents a reduction scheme from the halting problem of Minsky (2-counter) machines to $\mathtt{INV}[\mathcal{C}, \mathcal{L}]$. The scheme is later instantiated to obtain two undecidability results. The reduction is parameterized by an encoding for counters, denoted by $\mathcal{E}$. We first present the reduction, and then the conditions on $\mathcal{E}$ needed for it to be correct.

Input We are given an arbitrary Minsky machine, $M = (Q, c_1, c_2)$, where c_1, c_2 are counters, both initially 0, and $Q = q_1, \dots, q_n$ is a finite sequence of instructions, where q_1 is the first instruction, and q_n is the halting instruction. The possible instructions are:

i_k: increment counter c_k

d_k: decrement counter c_k

$t_k(j)$: if counter c_k is 0 go to instruction j

where in each case, control is passed to the next instruction except when the tested counter is 0 and thus the branch is taken.

Idea The reduction constructs $(TS, P, L) \in (\mathcal{C}, \mathcal{L})$, such that

$$(TS, P, L) \in \mathtt{INV}[\mathcal{C}, \mathcal{L}] \quad \text{iff} \quad M \text{ halts}$$

The idea is for *TS* to simulate M, and in parallel simulate a third counter c_3 that is initially 0, and will always contain an even value. Specifically, each transition of *TS* will simulate one step of M, and will non deterministically increment or decrement c_3 by 2. The safety property P will assert that c_3 does not contain the value 1. Notice that $TS \models P$ regardless of whether M halts. Both *TS* and P will be encoded in first-order logic, and the encoding will be constructed such that the two correctness conditions hold:

1. If M halts, *TS* will have finitely many reachable configurations, and there will be an inductive invariant in L, constructed by a disjunction over formulas in L representing these configurations.
2. If M does not halt, there will be no inductive invariant in L, since L will not be able to express the fact that the value of c_3 is even, which is needed for inductuveness.

Construction To have *TS* simulate M, we use nullary relations $q_1, \dots, q_n$ to keep track of M's current instruction (we overload the q_i's for instructions and nullary relation symbols). We also need to encode the value of the three counters, that have infinitely many possible values. To do so, we will use an encoding $\mathcal{E}$ of the counters over vocabulary $\Sigma_{\mathcal{E}}$, which will be provided by each instantiation of the reduction (see Section 8.2 and Section 8.3). The formulas for *TS* and P can easily be constructed if $\mathcal{E}$ provides the following formulas (for $i = 1, 2, 3$):

- inc_i a transition formula for increasing the value of c_i by 1
- dec_i a transition formula for decreasing the value of c_i by 1
- id_i a transition formula for keeping the value of c_i unchanged
- $zero_i$ a formula for testing if the value of c_i is 0
- $init$ a formula for the initial state s.t. $init \Rightarrow \bigwedge_{i=1}^{3} zero_i$

The output of the reduction is $(TS, P, L) \in (\mathcal{C}, \mathcal{L})$, given by the vocabulary $\Sigma_{\mathcal{E}} \cup \{q_1, \dots, q_n\}$, and the formulas τ, φ_0 and φ_P that are constructed from the formulas that $\mathcal{E}$ provides.[4]

Correctness conditions For the reduction to be correct, $\mathcal{E}$ must guarantee both correctness conditions. For the first correctness condition, assuming M halts, there are finitely many reachable configuration, each defined by the current instruction of M and the values of c_1, c_2, c_3. Denote by $Reach \subseteq Q \times \mathbb{N}^3$ the finite set of reachable configurations. An inductive invariant $\varphi_I \in L$ for *TS* can be defined as follows:

$$\varphi_I = \bigvee_{(q_i, \ell_1, \ell_2, \ell_3) \in Reach} q_i \wedge \varphi_{\mathcal{E}}(\ell_1, \ell_2, \ell_3),$$

where for any $\ell_1, \ell_2, \ell_3 \in \mathbb{N}$, $\varphi_{\mathcal{E}}(\ell_1, \ell_2, \ell_3)$ is a "witness" formula *in the language* L that is specific to $\mathcal{E}$. φ_I will be inductive if $\mathcal{E}$ guarantees that for any $\ell_1, \ell_2, \ell_3 \in \mathbb{N}$ the following holds:

$$\begin{aligned}
&\bigwedge_{i=1}^{3} zero_i \Rightarrow \varphi_{\mathcal{E}}(0,0,0) \\
&\varphi_{\mathcal{E}}(\ell_1, \ell_2, \ell_3) \wedge inc_1 \wedge id_2 \wedge id_3 \Rightarrow \varphi_{\mathcal{E}}(\ell_1 + 1, \ell_2, \ell_3)' \\
&\varphi_{\mathcal{E}}(\ell_1, \ell_2, \ell_3) \wedge dec_1 \wedge id_2 \wedge id_3 \Rightarrow \varphi_{\mathcal{E}}(\ell_1 - 1, \ell_2, \ell_3)' \text{ for } \ell_1 \geq 1 \\
&\quad \dots \text{ similarly for } inc_2, dec_2, inc_3, dec_3 \dots \\
&\varphi_{\mathcal{E}}(\ell_1, \ell_2, \ell_3) \Rightarrow \neg dec_i \text{ for } \ell_i = 0 \\
&\varphi_{\mathcal{E}}(\ell_1, \ell_2, \ell_3) \Rightarrow \neg zero_i \text{ for } \ell_i \neq 0
\end{aligned} \tag{5}$$

The first requirement guarantees that the initial state will satisfy φ_I, and the others guarantee that φ_I will be inductive (i.e. closed under transitions of *TS*), and imply the safety property.

8.2 Undecidability of $\mathtt{INV}[\mathcal{C}_{n^*}, \mathcal{L}_{AF}]$

Recall that $\mathcal{L}_{AF}$ allows only alternation-free invariants. In this subsection we instantiate the reduction scheme of Section 8.1 to prove the following theorem:

Theorem 8.1. *INV$[\mathcal{C}_{n^*}, \mathcal{L}_{AF}]$ is undecidable.*

To instantiate the reduction scheme of Section 8.1 for $\mathtt{INV}[\mathcal{C}_{n^*}, \mathcal{L}_{AF}]$, we present an encoding $\mathcal{E}_{n^*}$ of the counters using n^* and linked-lists as per $\mathcal{C}_{n^*}$, and show witness formulas $\varphi_{\mathcal{E}}$ that are *alternation-free* as per $\mathcal{L}_{AF}$, that prove the correctness of the reduction.

Encoding We encode the 3 counters c_1, c_2, c_3 using 3 disjoint linked-lists, where the length of list i encodes the value of c_i. The vocabulary $\Sigma_{\mathcal{E}}$ contains the binary relation n^* and 3 constant symbols h_1, h_2, h_3 for the heads of the lists. Figure 6(a1) provides an example of a structure that arises in this encoding for $c_1 = 3, c_2 = 0$ and $c_3 = 2$. The encoding formulas will be:

- inc_i will prepend a new node to list i
- dec_i will remove a node from the start of list i, assuming there is a $next$ edge from h_i (otherwise dec_i is not satisfied)
- id_i will keep h_i unchanged
- $zero_i$ will test if there is no $next$ edge from h_i
- $init$ will assert n^* is the identity relation and the h_i's are distinct

These formulas are all easy to write in EPR using n^*, as presented in Section 5.1.

Correctness For the second correctness condition of the reduction, we need to show that if M does not halt, there is no alternation-free inductive invariant. Intuitively, this is true for this encoding, due to the fact that it is not expressible in first-order logic to say that the length of a list is even using n^*.

More formally, any inductive invariant must be true of all the reachable states. If M does not halt, then these states include segments corresponding to c_3 that are line graphs of unbounded even length. Suppose that our inductive invariant, φ, has quantifier depth k. Let s be any state satisfying φ such that the length of the list encoding c_3 is $\ell > 2^k$. If we modify s to s' only by adding one more segment to c_3's list leaving everything else the same, then it is easy to show using Ehrenfeucht-Fraïssé games [24] that $s \equiv_k s'$, i.e., that they agree on all first-order formulas of quantifier depth k. Thus $s' \models \varphi$. But this leads to a contradiction because s' is not safe.

Witness formulas For the first correctness condition (namely that if M halts there is an alternation-free inductive invariant), we need to present the existence of alternation-free witness formulas that meet the conditions of Equation (5). For any $\ell_1, \ell_2, \ell_3 \in \mathbb{N}$, the witness formula $\varphi_{\mathcal{E}}(\ell_1, \ell_2, \ell_3)$ will be the conjunction of the following:

[4] Note that if $\mathcal{E}$ provides the above formulas in $\exists^*\forall^*$ form, then *TS* can be constructed to be an EPR transition system.

- A universal formula asserting the h_i's point to disjoint lists: $\forall x.\, n^*(h_i,x) \wedge n^*(h_j,x) \rightarrow i=j$
- An existential formula asserting the length of list i is at least ℓ_i: $\exists x_1 \dots x_{\ell_i}.\, \mathtt{distinct}(h_i, x_1, \dots, x_{\ell_i}) \wedge \bigwedge_{j=1}^{\ell_i} n^*(h_i, x_j)$
- A universal formula asserting the length of list i is at most ℓ_i: $\forall x_0 \dots x_{\ell_i}.\, \neg(\mathtt{distinct}(h_i, x_0, \dots, x_{\ell_i}) \wedge \bigwedge_{j=0}^{\ell_i} n^*(h_i, x_j))$

The conjunction is clearly alternation-free. These witness formulas guarantee the conditions of Equation (5). Intuitively, this is because $\varphi_{\mathcal{E}}(\ell_1,\ell_2,\ell_3)$ is strong enough to guarantee that in any model of it, the h_i's points to 3 disjoint linked-lists of exactly the correct lengths.

8.3 Undecidability of $\mathtt{INV}[\mathcal{C}_n, \mathcal{L}_{\forall^*}]$

For the second undecidability proof that we present, we first define the class $\mathcal{C}_n$ of transition systems which allows a single binary relation that is not restricted by a background theory. We show that $\mathtt{INV}[\mathcal{C}_n, \mathcal{L}_{\forall^*}]$ is undecidable, and thus $\mathtt{INV}[\mathcal{C}, \mathcal{L}_{\forall^*}]$ is undecidable for any extension $\mathcal{C}$ of $\mathcal{C}_n$ as well.

Definition 9 ($\mathcal{C}_n$)**.** We denote by $\mathcal{C}_n$ the class of EPR transition systems where Σ contains a single binary relation symbol n, any finite number of nullary relation symbols, and 4 constant symbols, and there is no background theory.

Therefore, the state space of *TS* $\in \mathcal{C}_n$ is STRUCT[Σ] (no background theory). This subsection proves the following:

Theorem 8.2. *INV*$[\mathcal{C}, \mathcal{L}_{\forall^*}]$ *is undecidable for every class* $\mathcal{C}$ *that extends* $\mathcal{C}_n$*.*

To show undecidability of $\mathtt{INV}[\mathcal{C}_n, \mathcal{L}_{\forall^*}]$, we present an encoding $\mathcal{E}_n$, which uses the single unrestricted binary relation n allowed by $\mathcal{C}_n$ to encode the counters. For the correctness of the reduction we must show *universal* witness formulas $\varphi_{\mathcal{E}}$, as per $\mathcal{L}_{\forall^*}$, that meet the conditions of Equation (5).

We use the same idea of Section 8.2, namely to encode the counters c_1, c_2, c_3 using 3 disjoint linked-lists whose heads are h_1, h_2, h_3, and to use the list lengths to encode counter values. We first explain why the fact that $\mathcal{L}_{\forall^*}$ allows only universal invariants (as opposed to alternation-free) makes the reduction trickier, and then present the encoding that can be used for this reduction.

Encoding A zero-attempt is to make n be n^*, and use the transition formulas and witness formulas to enforce the theory Γ_{n^*}. This approach is bound to fail, since $\mathtt{INV}[\mathcal{C}_{n^*}, \mathcal{L}_{\forall^*}]$ is decidable, but it is useful to understand where exactly it fails. The second direction of the reduction (if M does not halt then there is no inductive invariant) will be correct exactly as it was in Section 8.2, but the first direction will fail: since we are targeting $\mathcal{L}_{\forall^*}$, we must present universal witness formulas and these do not exist.

To make the reduction work, we must take advantage of the fact that n is not restricted by a background theory. We do this by using n, to encode $next$ edges directly. It is easy to use n to express linked-lists operations such as prepending to a linked-list, removing an element from the head of a list, and checking if the head has a $next$ edge. Thus, we can use n to write the formulas inc_i, dec_i, id_i, $zero_i$, and $init$ to express the same linked-lists operations as in Section 8.2, but this time using n and not n^*. Figure 6(b1) provides an example of a structure that arises in this encoding for $c_1 = 3, c_2 = 0$ and $c_3 = 2$.

However, this also fails. The reason is that any universal formula is closed under substructure (unlike alternation-free formulas), and a structure that contains a linked-list of length > 0, has a substructure where the head of the list does not have a $next$ edge (if we remove the first node after the head from the domain), and this substructure will satisfy $zero_i$. This is demonstrated by Figure 6(b2) which is a substructure of Figure 6(b1), but $zero_3$ (wrongfully) holds in it. For this reason, any universal $\varphi_{\mathcal{E}}$ cannot guarantee that $\varphi_{\mathcal{E}}(\ell_1,\ell_2,\ell_3) \Rightarrow \neg zero_i$ for $\ell_i \neq 0$.

This can be fixed by adding another constant symbol t, to represent a common tail of the linked-lists, and changing the formulas to say:

- inc_i will prepend a new node to list i (creating a new n edge)
- dec_i will remove a node from the start of list i, assuming $h_i \neq t$
- id_i will keep h_i unchanged
- $zero_i$ will test if $h_i = t$
- $init$ will say that n is empty and $head_i = t$ for $i = 1, 2, 3$

The key idea is that since the common tail is guarded by a constant, no "transition to substructure" can transform a list of length > 0 to a list of length 0. Note however, that a transition to substructure can remove elements that are part of the n-path from h_i to t from the domain, thus making the list disconnected.

Figure 6(c1) provides an example of the resulting encoding $\mathcal{E}_n$ for $c_1 = 3, c_2 = 0$ and $c_3 = 2$. In this case, the substructure in Figure 6(c2) no longer satisfies $zero_3$.

Correctness The second correctness condition of the reduction (if M does not halt then there is no universal inductive invariant) is correct for this encoding by the same arguments provided for $\mathcal{E}_{n^*}$ (it is not expressible in first-order logic to say that the length of an n-path is even). For the first correctness condition (if M halts there is a universal inductive invariant), we present universal witness formulas that meet the conditions of Equation (5).

Witness formulas For any $\ell_1, \ell_2, \ell_3 \in \mathbb{N}$, the witness formula $\varphi_{\mathcal{E}}(\ell_1,\ell_2,\ell_3)$ will be the conjunction of the following:

- A q.f. formula asserting $h_i \neq h_j$ unless they are both equal to t
- A universal formula saying the h_i's have no incoming n edges
- A universal formula encoding that list i is of length ℓ_i:

$$\forall x_0 \dots x_{\ell_i}.$$
$$\bigwedge_{j=0}^{\ell_i} \big((h_i = x_0 \wedge \bigwedge_{k=0}^{j-1} n(x_k, x_{k+1})) \rightarrow (x_j = t \leftrightarrow j = \ell_i)\big)$$

The conjunction is clearly universal, and these witness formulas guarantee the conditions for inductiveness of Equation (5). Intuitively, the first two conjuncts make sure the 3 lists do not interfere with each other, and the last one makes sure that all n-paths of length at most ℓ_i starting from h_i are consistent with the fact that list i is of length ℓ_i: paths shorter than ℓ_i must not end in t, and paths of length ℓ_i must end in t.

8.4 Complexity of $\mathtt{INV}[\mathcal{C}_{n^*}, \mathcal{L}_{\forall^*}]$ is Non-Elementary

As we saw in Section 6, $\mathtt{INV}[\mathcal{C}_{n^*}, \mathcal{L}_{\forall^*}]$ is decidable. In this section, we show that the complexity of $\mathtt{INV}[\mathcal{C}_{n^*}, \mathcal{L}_{\forall^*}]$ is non-elementary. Therefore this is also true for any $(\mathcal{C}, \mathcal{L})$ extending $(\mathcal{C}_{n^*}, \mathcal{L}_{\forall^*})$ with any of the constructions presented in Section 7.

Theorem 8.3. *The complexity of INV*$[\mathcal{C}_{n^*}, \mathcal{L}_{\forall^*}]$ *is non-elementary.*

We prove the theorem by a reduction from the reachability problem of lossy counter machines, which is known to have non-elementary complexity [36, 37], to the complement of $\mathtt{INV}[\mathcal{C}_{n^*}, \mathcal{L}_{\forall^*}]$. The reachability problem for lossy counter machines is: given a counter machine $M = (Q, c_1, \dots, c_k)$ and $q_{\text{goal}} \in Q$, is there a trace of M under the lossy semantics that leads from the initial configuration $(q_1, 0, \dots, 0)$, to a configuration with q_{goal}. Recall that the lossy semantics lets the value of any counter decrease non-deterministically in any transition.

The input of the reduction is (M, q_{goal}), and the output is $(TS, P, L) \in (\mathcal{C}_{n^*}, \mathcal{L}_{\forall^*})$ such that:

M has a lossy trace to $q_{\text{goal}} \Leftrightarrow (TS, P, L) \notin \texttt{INV}[\mathcal{C}_{n^*}, \mathcal{L}_{\forall^*}]$

According to Section 6, for any language in the context of $(\mathcal{C}_{n^*}, \mathcal{L}_{\forall^*})$, $\sqsubseteq_{\forall^*}$ is a wqo and Avoid$_{\forall^*}$ is computable. Therefore, we can apply Corollary 4.7 and thus:

$$(TS, P, L) \notin \texttt{INV}[\mathcal{C}_{n^*}, \mathcal{L}_{\forall^*}] \Leftrightarrow TS^{\forall^*} \not\models P$$

Therefore, it suffices to show that

$$M \text{ has a lossy trace to } q_{\text{goal}} \Leftrightarrow TS^{\forall^*} \not\models P \qquad (6)$$

To construct *TS* that satisfies Equation (6), we use the same encoding presented in Section 8.2. The only difference is that for this reduction we do not need the special counter that always holds even values, and we just model the counters of the input machine M. We encode Q with nullary relations and the values of the k counters using n^* and k disjoint linked-lists whose heads are $h_1, \ldots, h_k$, where the lengths of the lists keep the values of the counters. The formulas for the initial state and the transition relation are constructed as in Section 8.1 and Section 8.2, and the safety property is given by $\varphi_P = \neg q_{\text{goal}}$.

To see that the described *TS* satisfies Equation (6), recall that $TS^{\forall^*} \not\models P$ iff *TS* has a $\forall^*$-relaxed trace to $S \setminus P$, where a $\forall^*$-relaxed trace can make transitions to substructures (since $\sqsubseteq_{\forall^*}$ is the substructure relation). The punch line is that for the encoding of Section 8.2, these transitions exactly correspond to decrements of the counters. The reader should convince herself that for a linked-list represented by n^* and a constant for the head, any substructure is a linked-list with less elements. (For an example, see Figure 6(c1) and its substructure Figure 6(c)). Therefore, by the construction of *TS* we get that $\forall^*$-relaxed traces of *TS* exactly correspond to lossy traces of the counter machine M, and thus Equation (6) holds, which completes the proof.

9. Related Work

Decidability for infinite-state systems via wqo's Following [2–5], well-quasi-orders and well-structured (monotonic) transition systems (WSTS's) have become a standard technique for proving decidability of various problems over infinite-state systems (e.g. [16]). Often a WSTS is obtained by relaxing the semantics of a transition system to a *lossy* semantics. The classical examples of this are lossy channel systems [1], lossy counter machines (e.g. [31]) and vector addition systems or Petri nets (e.g. [29]). In [10, 20], this technique is applied to array-based transition systems. States are identified with finitely-generated models, and the quasi-order on states is the substructure relation.

The common approach in these works is to consider a system under lossy semantics or via an abstraction, and to analyze the decidability of the *safety* of the lossy/abstract system. Our work focuses on transition systems formalized using first-order logic, and takes the viewpoint of restricting potential inductive invariants to some language L, and then analyzing the decidability of *invariant inference* in L. The two views are connected: every particular language L can be regarded as an abstraction, where the abstract domain is formulas in L; or as a lossy semantics, with lossy transitions according to $\sqsubseteq_L$[5].

Our use of first-order logic together with a formalism which is parameterized by L, allows the systematic constructions presented in Section 7, that are able to construct restricted languages and wqo's that are applicable to complicated systems like the learning switch. From a practical perspective, using logic enables to use existing decision procedures in the invariant inference process. The operation Avoid$_L$ corresponds to concepts found both in Property Directed Reachability algorithms [9, 26] and in decision procedure based abstract interpreters, e.g., [39], that automate the process of inferring inductive invariants. Our work can be seen as studying the conditions required for termination of such procedures, and the conditions under which the underlying problem is undecidable and thus divergence occurs for infinitely many instances.

Linked-lists & counter machines In [8], it was observed that linked-list programs are counter machines. While the safety of counter machines is undecidable, the safety of lossy counter machines is decidable with non-elementary complexity [36]. In the case of linked-lists, our work can be seen as combining these two interesting results: "Linked-lists with *alternation-free* invariants are counter machines" (and therefore undecidable), and "Linked-lists with *universal* invariants are *lossy* counter machines" (and therefore decidable with non-elementary complexity).

In [6, 7], monotonic abstractions are defined for linked data structures. The wqo on heaps defined there is semantically similar to $\sqsubseteq_{\forall^*}$ with n^*, but it is defined via tree operations and not via logic. This makes the result of Section 6 more powerful because it also provides unlimited unary predicates, as well as being amenable to the constructions of Section 7. The wqo of [6] results in a slightly less precise abstraction compared to that of Section 6, as it also allows edge deletion (which is not allowed in $\sqsubseteq_{\forall^*}$ with n^*). Interestingly, the proof in [6] does not use Kruskal's Tree Theorem, which is the basis for our result.

Undecidability There are many undecidability results for safety of infinite state systems, e.g., [8, 11]. For the related problem `INV` of inferring inductive invariants we show two interesting undecidability results: for universal invariants over arbitrary domains; and for alternation free formulas over the theory of linked-list reachability.

Completeness of abstract interpretation The theoretical study of the precision of abstract interpretation is an ongoing research area. Usually *completeness* for abstract interpretation means that the abstract domain is precise enough to prove all interesting safety properties, e.g., [21]. In our terms, this means that $\texttt{INV} = \texttt{SAFE}$, that is that all safe programs have an inductive invariant expressible in the abstract domain. Since we are interested in automatically analyzing Turing Complete programs (for which `SAFE` is undecidable), we consider the completeness problem at a later binding time. Essentially, `INV` asks if the abstract domain is precise enough for a *given* property in a *given* program.

A very interesting twist on completeness of abstract interpreters is considered in [22] which algorithmically studies the completeness of certain abstract domains for certain questions.

For numeric domains (e.g. intervals), techniques such as widening and policy iteration (e.g. [19]) are used to prove the correctness of many infinite state systems. Surprising results in [18, 38] show that for some numeric domains and programs, precise information can be computed, thus solving `INV`.

Acknowledgments

This work was funded by the European Research Council under the European Union's Seventh Framework Program (FP7/2007–2013) / ERC grant agreement no. [321174-VSSC], and by a grant from the Israel Science Foundation (652/11). Immerman was partially supported by NSF grant CCF 1115448. We thank Yaron Velner, Nikolaj Bjørner and the anonymous referees for insightful comments which improved this paper.

[5] Indeed, TS^L can alternatively be formulated as a monotonic transition system, and this can be viewed as monotonic abstraction as in [5].

References

[1] P. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Logic in Computer Science, 1993. LICS'93., Proceedings of Eighth Annual IEEE Symposium on*, pages 160–170. IEEE, 1993.

[2] P. A. Abdulla and B. Jonsson. Ensuring completeness of symbolic verification methods for infinite-state systems. *Theoretical Computer Science*, 256(1):145–167, 2001.

[3] P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Logic in Computer Science, 1996. LICS'96. Proceedings., Eleventh Annual IEEE Symposium on*, pages 313–321. IEEE, 1996.

[4] P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1):109–127, 2000.

[5] P. A. Abdulla, G. Delzanno, N. B. Henda, and A. Rezine. Regular model checking without transducers (on efficient verification of parameterized systems). In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 721–736. Springer, 2007.

[6] P. A. Abdulla, A. Bouajjani, J. Cederberg, F. Haziza, and A. Rezine. Monotonic abstraction for programs with dynamic memory heaps. In *CAV'08*, pages 341–354, 2008.

[7] P. A. Abdulla, J. Cederberg, and T. Vojnar. Monotonic abstraction for programs with multiply-linked structures. *Int. J. Found. Comput. Sci.*, 24(2):187–210, 2013.

[8] A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar. Programs with lists are counter automata. *Formal Methods in System Design*, 38(2):158–192, 2011.

[9] A. R. Bradley. Sat-based model checking without unrolling. In *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, pages 70–87, 2011.

[10] A. Carioni, S. Ghilardi, and S. Ranise. Automated termination in model-checking modulo theories. *Int. J. Found. Comput. Sci.*, 24(2): 211–232, 2013.

[11] V. T. Chakaravarthy. New results on the computability and complexity of points - to analysis. In *Conference Record of POPL 2003: The 30th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, New Orleans, Louisisana, USA, January 15-17, 2003*, pages 115–125, 2003.

[12] C. Chang and H. Keisler. *Model Theory*. Studies in Logic and the Foundations of Mathematics. Elsevier Science, 1990. ISBN 9780080880075.

[13] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Symp. on Princ. of Prog. Lang.*, pages 269–282, New York, NY, 1979. ACM Press.

[14] P. Cousot, R. Cousot, and F. Logozzo. A parametric segmentation functor for fully automatic and scalable array content analysis. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 105–118, 2011.

[15] L. De Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, 2008.

[16] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1):63–92, 2001.

[17] R. W. Floyd. Assigning meanings to programs. In *Proceedings of Symposium on Applied Mathematics*, number 32, 1967.

[18] T. Gawlitza, J. Leroux, J. Reineke, H. Seidl, G. Sutre, and R. Wilhelm. Polynomial precise interval analysis revisited. In *Efficient Algorithms, Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, pages 422–437, 2009.

[19] T. M. Gawlitza and D. Monniaux. Invariant generation through strategy iteration in succinctly represented control flow graphs. *Logical Methods in Computer Science*, 8(3), 2012.

[20] S. Ghilardi and S. Ranise. Backward reachability of array-based systems by SMT solving: Termination and invariant synthesis. *Logical Methods in Computer Science*, 6(4), 2010. . URL `http://dx.doi.org/10.2168/LMCS-6(4:10)2010`.

[21] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361–416, 2000.

[22] R. Giacobazzi, F. Logozzo, and F. Ranzato. Analyzing program analyses. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 261–273, 2015.

[23] G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, pages 326–336, 1952.

[24] N. Immerman. *Descriptive Complexity*. Graduate Texts in Computer Science. Springer, 1999.

[25] S. Itzhaky, A. Banerjee, N. Immerman, A. Nanevski, and M. Sagiv. Effectively-propositional reasoning about reachability in linked data structures. In *CAV*, volume 8044 of *LNCS*, pages 756–772, 2013.

[26] A. Karbyshev, N. Bjorner, S. Itzhaky, N. Rinetzky, and S. Shoham. Property-directed inference of universal invariants or proving their absence. In *CAV*, 2015.

[27] J. Kruskal. Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95(2), May 1960.

[28] K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 348–370. Springer, 2010.

[29] J. Leroux. The general vector addition system reachability problem by Presburger inductive invariants. In *Logic In Computer Science, 2009. LICS'09. 24th Annual IEEE Symposium on*, pages 4–13. IEEE, 2009.

[30] T. Lev-Ami, T. Reps, M. Sagiv, and R. Wilhelm. Putting static analysis to work for verification: A case study. In *Proc. of the Int. Symp. on Software Testing and Analysis*, 2000.

[31] R. Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science*, 297(1):337–354, 2003.

[32] A. Møller and M. I. Schwartzbach. The pointer assertion logic engine. In *Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Snowbird, Utah, USA, June 20-22, 2001*, pages 221–231, 2001.

[33] C. Nash-Williams. On well-quasi-ordering finite trees. In *Proc. Of the Cambridge Phil. Soc. 59*, 1963.

[34] V. Perrelle and N. Halbwachs. An analysis of permutations in arrays. In *Verification, Model Checking, and Abstract Interpretation, 11th International Conference, VMCAI 2010, Madrid, Spain, January 17-19, 2010. Proceedings*, pages 279–294, 2010.

[35] S. Sagiv, T. W. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Trans. Program. Lang. Syst.*, 24(3):217–298, 2002.

[36] P. Schnoebelen. Lossy counter machines decidability cheat sheet. In *Reachability Problems, 4th International Workshop, RP 2010, Brno, Czech Republic, August 28-29, 2010. Proceedings*, pages 51–75, 2010.

[37] P. Schnoebelen. Revisiting Ackermann-hardness for lossy counter machines and reset petri nets. In *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, pages 616–628, 2010. . URL `http://dx.doi.org/10.1007/978-3-642-15155-2_54`.

[38] Z. Su and D. Wagner. A class of polynomially solvable range constraints for interval analysis without widenings. *Theor. Comput. Sci.*, 345(1):122–138, 2005.

[39] A. V. Thakur, A. Lal, J. Lim, and T. W. Reps. Posthat and all that: Automating abstract interpretation. *Electr. Notes Theor. Comput. Sci.*, 311:15–32, 2015.